TUM Data Innovation Lab

Munich Data Science Institute (MDSI)

Technical University of Munich

&

PwC Deutschland

Final report of project:

Innovative Machine Learning Algorithm meets Carrera for auto piloting

Authors	Dohyun Jeong, Daniel-Jordi Regenbrecht, Stefan Tremmel
Mentors	M.Sc. Oliver Kobsik, M.Sc. Stephan Bautz, B.A. Manuel Kuhlin, M.Sc. Daniel Reider, M.Sc. Philipp Düpree
Co-Mentor	M.Sc. Cristina Cipriani
Project Lead	Dr. Ricardo Acevedo Cabra (MDSI)
Supervisor	Prof. Dr. Massimo Fornasier (MDSI)

Feb 2023

Abstract

In the European motor insurance market, risk assessment is largely based on vehicle and user groups without taking into account actual driving behaviour. In recent years, the possibilities of real-time recording of vehicle movements have improved, as even smartphones are able to capture the relevant information. Telematics insurance utilizes this data and attempts to create a behavior based risk profile with an adjusted premium. Various metrics can be shown to the policy holder as immediate feedback on the current trip to incentivize safe driving with the ambition to reduce overall accidents. To make the telematics concept more accessible, we aim to transfer this real world problem to a Carrera slot car track and combine it with state of the art reinforcement learning for autonomous driving.

In order to provide a complete and flexible model, we build an Internet of Things (IoT) system upon the original Carrera slot car with additional components. The hand-held controller is replaced by electronic devices to control the slot car precisely. Physical sensors are added to receive the data about the current position and movement of the slot car. This sensor data is sent to a database in a cloud service with a predefined format. We present the IoT system together with experiments conducted to determine the optimal adjustment of the environment.

Risk quantification is a key part of the behavior based insurance. To identify relevant data, we conduct detailed data analyses and use the findings to define a scoring function which is suitable for the Carrera environment. We further suggest a dashboard for real-time monitoring to provide users with information about driving safety.

A related trend is seen in the number of advances made in autonomous vehicles. Wide-scale acceptance of these technologies hinges on proving that driving safety is at least comparable or preferably superior to human drivers. Our setting provides a well-controlled model scenario to illustrate these concepts. However, the performance of reinforcement learning techniques for many real-world scenarios with no simulation environment have suffered from limited availability of live interaction. Benefiting from recent advances in the field of batch reinforcement learning, we present an agent for autonomous driving on the race track, which is trained completely without real-world interaction and achieves competitive results compared to human drivers.

Contents

1 Introduction

1.1 Motivation

Motor insurance is the largest property and casualty business line in Europe with premiums of €149.4bn in 2020 [1]. It includes, for example, mandatory motor third-party liability coverage. Standard pricing methods depend on many covariates such as type of vehicle, regional classification, or annual mileage performance. Another prominent and influential factor is the bonus-malus level, those tariff criteria are then used to create tariff classes for customers. However, using this approach the actual driving behavior is not reflected, only vehicle and user group [2]. In recent years, more sensor data is collected by modern cars used for new features on the way to autonomous driving. This also enables innovations in the insurance industry.

The so-called usage-based insurance rates individual driving behavior. Calculation strategies might include information about car velocity, acceleration, road type, time of day, weather conditions, and more [2]. Thus, in a basic form, even a mobile phone or a GPS device is sufficient to gather the required data. To get a realistic approximation of the driving style, high-frequency data with a sampling rate of 1 Hz is commonly used for monitoring [3].

The benefits for average safe driving customers are premium reductions and real-time feedback on driving behavior. Risky drivers might be disciplined with higher premia. Insurers have advantages as it is expected to promote safe driving and allow an individual pricing approach. A study from the Insurance Research Council in 2015 shows that telematics insurance indeed leads to a change in driving behavior for more than half of the participants [4]. This illustrates the potential of telematics to boost safe driving and reduce the number of road accidents and fatalities.

To illustrate the current differences in acceptance, we investigate two European countries: France and Italy. Recently, an article on the French automotive market noted that telematics is hardly used and that parts of the insurance industry do not see profitability from this branch in the short and medium term [5]. The authors attribute this to inertia on organizational and cognitive levels and a deliberate strategy. As they further mention that even with big data, it is hard to beat the pricing methods developed over the past decades. In Italy, however, motor claims decreased during most of the 2010s due to the introduction of telematic devices [1]. Particular changes were observed in the reduction of fraudulent claims and increased driving responsibility.

Those contrasts of acceptance motivate a project to make the concept of telematics more approachable and illustrate the functional principle. For this, we use a Carrera slot car track as a model of the real world. Slot car tracks are widely known and preserve features of driving, such as speed control, diverse routes, and the possibility of crashes. On the Carrera track, it is also feasible to establish an Internet of Things (IoT) environment with multiple sensors and connections for real-time interaction. This allows us

to add and illustrate another important concept simultaneously: self-driving cars. Current advances [6] in reinforcement learning (RL) enable us to apply them to our Carrera environment.

1.2 Project Goals and Contributions

With our project, we aim to provide an approachable real-world model of telematics insurance and autonomous driving to make them more accessible. Our main contributions are threefold, we present

- A flexible, modular IoT Ecosystem with a unified architecture for human and autonomous control, featuring a complete data pipeline from sensor data to high-level, cloud-based data analysis delivered in real-time,
- A risk-based scoring function to measure driving safety that is adaptable to a real-world setting, and
- A prototype of a Reinforcement Learning Agent trained for optimal control of the IoT system.

For all parts, we conduct and present extensive experiments to better understand the relevance and interactions of the system parts and analyze current shortcomings.

2 IoT Environment

Though there does not exist a generally-accepted rigorous definition of the term "Internet of Things", common characteristics of IoT systems have been identified in recent literature [7, 8, 9]. An IoT system comprises physical objects operating in a dynamical environment, often equipped with sensors and controllers, which are heterogenous in hardware platforms and networks but interconnected, i.e. able to exchange information and control signals as well as connected to global information and communication infrastructure.

In order to improve the understanding on matter and work towards a comprehensive definition, several attempts have been made to provide a general model for IoT ecosystems [10]. One of the most widely adopted models is the one presented at the 2014 IoT World Forum that separates an IoT environment into 7 vertical layers, as shown in Figure 1.

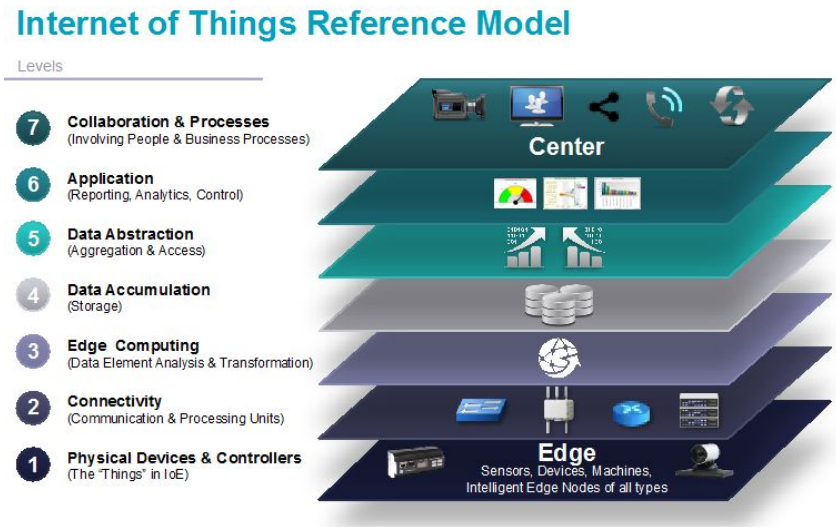


Figure 1: IoT Reference Model Published by the IoT World Forum [8].

2.1 Method: Implementation of Layers

The setting in which we operate fits well into the general characterization of IoT systems: The race track with a moving car constitute a dynamical system which we aim to equip with sensors to control the car. In doing so, it is indispensable to employ different devices with different hardware platforms that need to exchange information. This information should also be accessible to authorized users globally for purposes of data analysis and preparation.

The above described model provides a guideline to decompose the system into smaller parts, allowing to develop solutions independently of each other and optimally for their demands, while maintaining a unified interface to connect layers from bottom to top. We therefore decide to implement our system as an instance of this model. In the following,

we present our implementation for the first six layers: The final layer is omitted in this project, as it depends heavily on business processes which is outside our scope.

2.1.1 Devices and Controllers

In order to understand our overall setup of physical parts (shown in Figure 2), in particular, the choice of controlling devices, we need to lay out the working principles of a slot car. Figure 3 shows the basic principle how a slot car works. High alternating current (AC) is supplied via a power pack and converted into low direct current (DC). If one presses or releases the handle of the controller, its internal resistance decreases or increases, respectively. A DC motor is embedded in the slot car and the speed is controlled by the voltage due to the changes of the internal resistance of the controller. In summary, it is not different from a potentiometer and a DC motor simply connected in series.

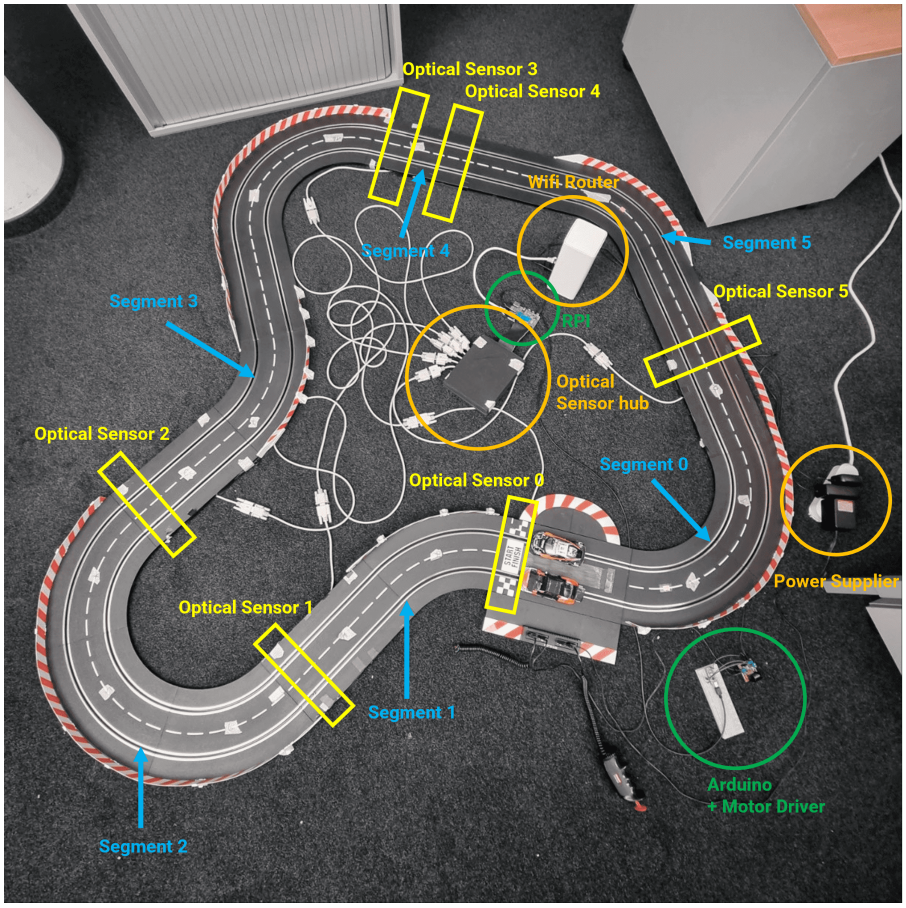


Figure 2: Overview of the Carrera track setting.

To steer the slot car without a controller, we need to identify the electric circuit of the Carrera track, the controller and the slot car. For the track, a current test was conducted with a multimeter, and for the controller, we decomposed it physically and observed the physical wire connections. Detailed information of the identified electric circuit and of the wiring between the devices are included in Appendix A.1.1.

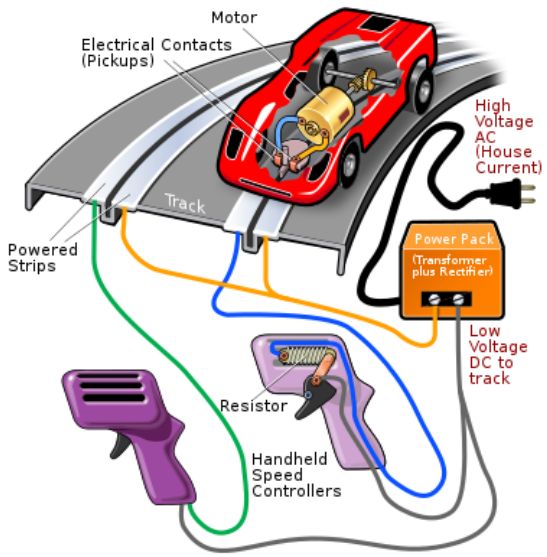


Figure 3: Basic principle of a slot car and the track [11].

Controllers To control the slot car precisely, we replace the standard controller for human control provided by the manufacturer with three different electronic devices, namely a Raspberry Pi, Arduino, and motor driver. In the following, we describe each of the electronic devices.

Raspberry Pi The Raspberry Pi (RPI) is a small single-board computer that operates in a Linux environment. In our project, we use a Raspberry Pi 4 Model B. All the Python codes are contained and executed on the RPI. The RPI is connected with a Wi-Fi router, allowing users to remotely connect to it.

Arduino The Arduino is a microcontroller that is widely used, because it provides a well-documented and easy to use programming environment. It can be used to send low-voltage control signals to the motor driver to regulate the motor speed. In our project, we use the Arduino Nano. As seen in Figure 4, the Arduino Nano is fixed on a breadboard and connected with the motor driver via cable.

Motor Driver A motor driver is an electronic component that is used to control the motor direction and speed precisely. It receives voltage that is supplied from a power supplier and control signals from a microcontroller. With this control signals, the output voltage is regulated leading to different motor directions and speeds. In our project, we use the L289N motor driver, because the voltage and current that are supplied from the Carrera power supplier are in the allowed range of this model.

Sensors To capture the movement of the slot car, we use two kinds of sensors, optical sensors and an Inertial Measurement Unit (IMU). In the following, we describe each of the sensors.

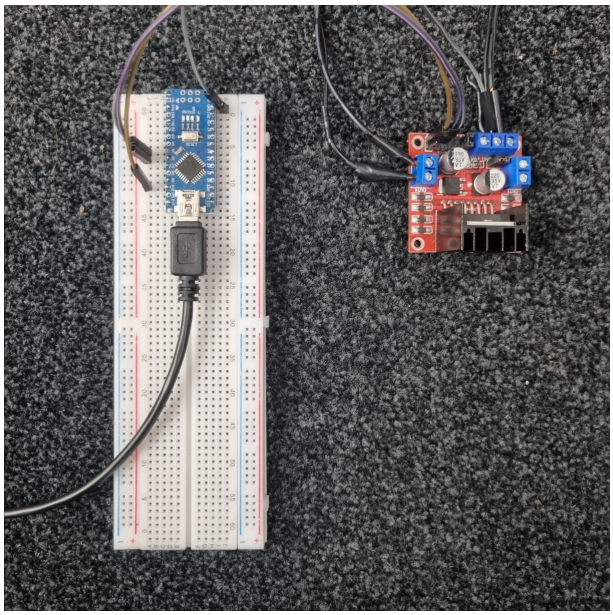


Figure 4: Arduino and the motor driver.

Optical sensors An optical sensor is a sensor that can detect an object when the object passes through the sensor. In total, 6 sensors have been built into the Carrera track as Figure 5a to detect where the slot car is currently driving. The sensors are directly connected to Raspberry Pi via an optical sensor hub. Detailed sensor positions on the track are depicted in Figure 2.

Inertial Measurement Unit With the optical sensors, we can identify the position of the car the instant an optical sensor is passed, but it cannot provide information of the movement in-between sensors. In order to capture the movement of the slot car more precisely, an additional sensor is needed.

For our project, the physical sensor must satisfy the following requirements in Table 1.

Requirement	**Reasoning**
Capture physical data (e.g. acceleration, angular velocity, etc...)	To represent the movement of the slot car.
Sufficiently high output rate	To receive detailed movement information.
Consistent output rate	To ensure consistent interpretability of the data.
Small physical size	To allow fixation on the small slot car.
Built-in battery	To keep volume and required wiring limited.
Wireless communication	To allow free movement of the car.
Sufficient resources (e.g. official APIs or open source repositories)	To access the sensor data accurately.

Table 1: The requirements for a physical sensor to capture the movement of the slot car and the reasoning

As the best fit for these requirements, we identified the 9-axis IMU Sensor BWT901CL of WitMotion. Not only x-, y-, and z-direction acceleration data, but also angle and angular velocity data can be measured, so that we can get more information of the motion of the slot car. Furthermore, the output rate is consistent and there are many open source repositories, so that we can access to the sensor data accurately.

We fix the IMU on the middle top of the slot car as Figure 5b, because the size of the sensor is not enough to set inside the slot car. Furthermore, if the sensor is positioned on the front or rear part of the car, it could effect the stability of the car.

(a) Optical sensors *(b) IMU*

Figure 5: Optical sensors embedded in the track and IMU positioned on top of the slot car.

2.1.2 Connectivity

The RPI is the core device that stands in the middle of the connections between the components. It receives sensor data from the optical sensors and the IMU and sends the data to Microsoft Azure SQL Database via a Wi-Fi Router. Control signals regulating the voltage on the track are calculated and transmitted from the RPI to the motor driver via the Arduino Nano. The overall communication between components is summarized in Figure 6.

2.1.3 Edge Computing

On the RPI, we provide an architecture that allows autonomous interaction with our IoT system. We use a modular approach in the software architecture that enables us to develop different solutions independently. We separate the *agent*, which is responsible for computing the appropriate control signal, and the *environment*, an abstraction of the physical system that handles the communication with other devices. In particular, it relays the control signal to the Arduino, which in turn controls the motor driver and also maintains the current state in the form of aggregated sensor information. This separation of agent and environment also lays the foundation to apply reinforcement learning, as discussed in-depth in Section 4. An overview of the classes and their relationship is depicted in Figure 7.

While driving, raw sensor data is constantly provided. The IMU measures at a predefined rate and the optical sensors whenever the car passes them. To not lose any

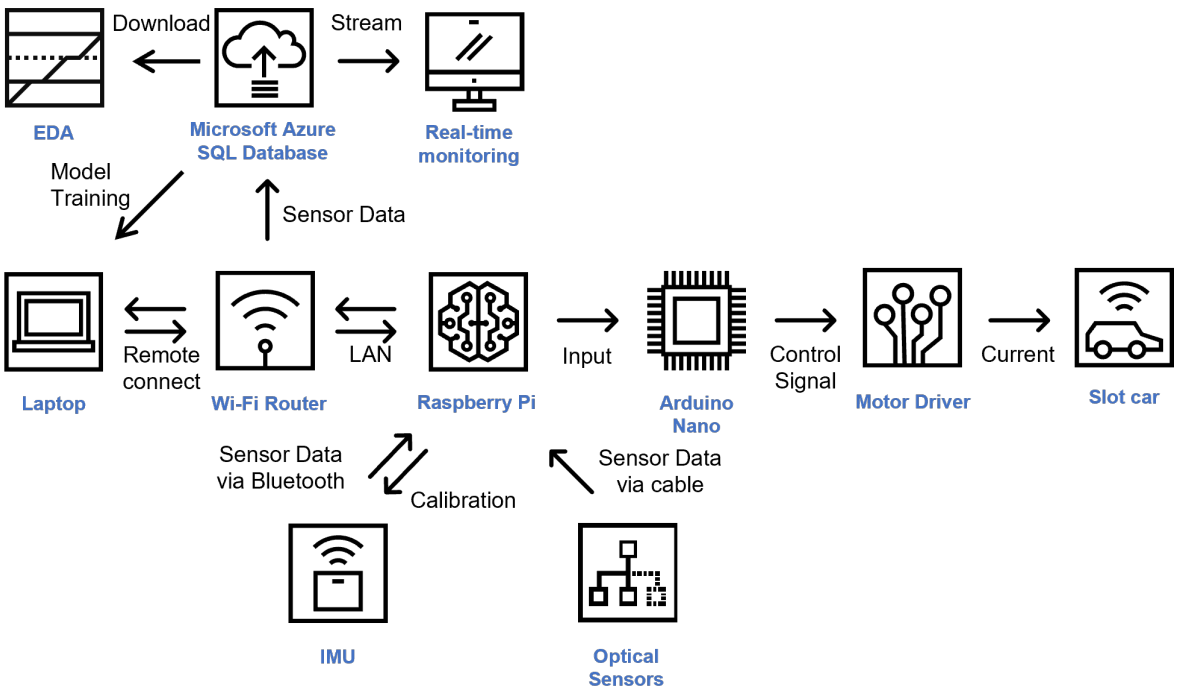


Figure 6: Communication between components and data flow.

information those data streams are bundled on the Raspberry Pi and temporarily stored. Once the sensor data for a segment was gathered, the data is enriched. Further metrics such as the segment time or the segment score are calculated. Basic information about the used trackside or the current time is also captured. Depending on the task, for example, human driving or automated data generation, the data is structured differently. What they have in common is that one data package is collected for one segment and sent to Microsoft Azure after every optical signal.

2.1.4 Data Accumulation and Abstraction

In our project, we combine data accumulation and abstraction to a unified approach. For a reliable data flow, we design a consistent data pipeline for all our workflows. The idea is to use the same infrastructure for any data we capture. In this section, we have a look at how data is stored and then where it is utilized. The journey starts at the sensors to the reinforcement learning and visualizations. The structure of this approach can be seen in Figure 6 extending the communication between parts.

We use an Azure Event Hub that receives data packages and hands them to a Stream Analytics Job within Azure. This process ensures real-time processing of the incoming data. The Stream Analytics query language, a subset of T-SQL syntax, allows for direct interaction with the data stream. However, for our use case, it is advantageous to do all computations directly on the Raspberry Pi as described above. The real-time data stream is then stored in an Azure SQL Database. We create several tables, e.g. for human-generated data or the reinforcement model, which are used depending on the task.

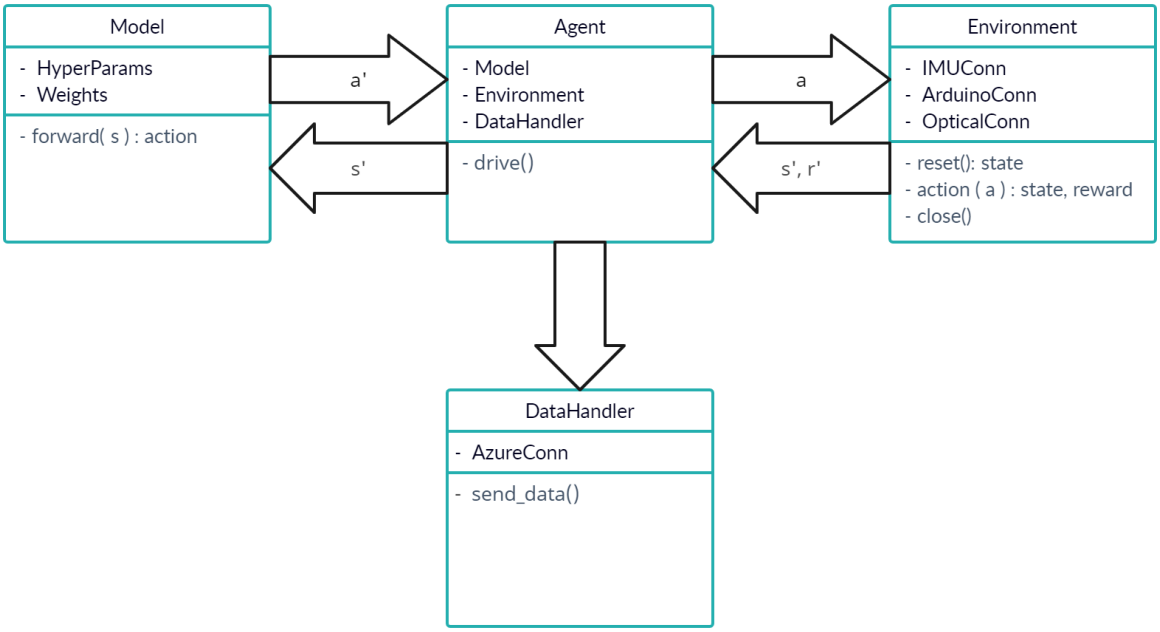


Figure 7: High-level overview of the software architecture.

2.1.5 Applications

Having well-structured databases in place, we can now start with the information extraction of the collected data. For real-time monitoring, we use Microsoft Power BI and via a direct query connection to the Azure SQL database, it is possible to update the information every second. This dashboard is the subject of discussion in Section 5. Detailed analyses are conducted in R Studio, where we created several reports to analyze human driving behavior and calibrate our functions. The quality of the sensor data is also part of those reports and helped us to gain a deeper understanding. Furthermore, we use the SQL database for off-line reinforcement learning, where we connect from our virtual machines to Azure.

2.2 Experiments

Several experiments are conducted to characterize and set the environment.

2.2.1 System Inconsistency

Our project is not based on a simulation environment but on the real world. Therefore, some inconsistencies need to be overcome.

Motor temperature fluctuation For the data collection, we need to drive many laps with the slot car to get enough data for training. Nevertheless, it is impossible to operate the slot car without pauses, as temperature fluctuations occur during the operation of the electric motor [12]. This phenomenon can lead to a reduction in energy efficiency of the electric motor, namely thermal loss [13], [14]. Therefore, we can not consistently achieve

the same velocity even if the same voltage was input. Figure 8 shows the result of the experiment. In the experiment, we input exactly the same signals for 100 laps. The x-axis indicates the lap and the y-axis the lap time in millisecond. In the graph, the lap time increases continuously despite the same inputs.

Thus we set a *break time* for every certain number of laps during the autonomous drive. With the break time, we can reduce the performance degradation, but not completely, because the degradation already starts after driving only one lap.

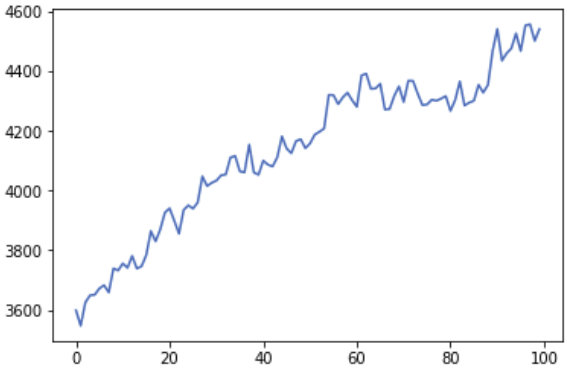


Figure 8: Lap times in milliseconds with constant input signal for 100 laps.

Aging of the parts Aging of the parts is an unavoidable problem in a real-world environment. The more we drive the slot car, the more the parts, e.g. sliding contacts and tires, will be worn. Furthermore, the electric motor of the slot car and the motor driver will age and this can lead to a performance degradation. The change of the friction coefficient of the track will be also observed. From several laps, we could observe that the sliding contact wears significantly fast.

Environmental inconsistency The environment of the place, where the Carrera track is installed, can be inconsistent. For example, temperature, humidity, and air flow are factors that can be changed by the place and time.

Car Model The specific model of the slot car can influence the outcome data. Depending on the model, they have different specifications, e.g. size, weight, and type of the motor, etc. With different physical characteristics, the slot cars show different tendencies in the movement.

2.2.2 Settings

In order to set the optimal environment for our project, we conducted several experiments regarding the Carrera track, electronic devices, and the sensors.

Track Modification The Carrera track for our project had a complex structure with height changes at the beginning. To simplify it, the track structure has modified. It is

expected that after all the processes in the project are set up, they can be transferred to the original track.

IMU Model Our first attempt for choosing the IMU was with the Xiaomi's MiBand 2 fitness band. After checking the ouput data of the sensor, we decided not to use the sensor because of following reasons:

- The acceleration data is the only data that we can receive from the sensor.
- The output rate differs every time, thus we couldn't get consisent data.
- There is no official supported API or enough open source repositories, thus there is no way to accurately access the sensor data.

Sampling Rate In order to decide the sampling rate of the IMU, three experiments with different sampling rates were conducted, namely 20Hz, 100Hz, and 200Hz. In case of 100Hz and 200Hz, many adjacent data showed the same value, i.e. the frequencies were too high. With 20Hz, the movement of the slot car was sufficiently captured with less memory. Therefore, we set the sampling rate of the IMU as 20Hz.

Data Consistency The same experiments from Section 2.2.1 were conducted to compare consistencies of the acceleration data and the angular velocity data from the IMU. In order to take into account the system inconsistency, normalized sensor data were compared. From these experiments, we could observe that the angular velocity data shows better consistency than the acceleration data.

Calibration One of the most common problems with physical sensors is that they need to be calibrated frequently. Some experiments were conducted to check the necessity to calibrate the sensor. After we drove the slot car several laps, the sensor outputs in the static state were compared. In these experiments, we found that the angular velocity data is relatively consistent, but the acceleration data is shifted after only a few laps. Therefore, we decided that the calibration is needed and wrote a calibration code. The code is executed every time the Carrera environment is started.

Detecting Failures In an ideal case, the optical sensors work consistently without any failures, but we observed that the sensors do not send a message occasionally in our case even if the slot car already passed through the sensor. This problem persists even after checking the connecting wires and cleaning the track. Therefore, we conducted analyses on the captured data and reveal that from 3313 samples 174 experience failure of one or more sensors. An important finding is that sensor 4 in particular tends to fail more frequently, as more than half of all faulty events can be traced back to it. However, each of the sensors is affected and sometimes several can fail in succession. In order to not disturb our results, we wrote code to stop generating data if this phenomenon is observed.

Bandwidth and Latency In order to check whether the bandwidth and latency is enough, we had an experiment to measure round-trip delay (RTT) between Arduino and Raspberry Pi. It takes about 35 milliseconds to send and receive a message, thus it is sufficiently fast to control the slot car.

3 Scoring Function

For both of our main fields of interest, telematics and reinforcement learning, we need a function to quantify the current state. Telematics insurance should provide instant feedback to its user, such that the driving behavior is rated in real-time. Because a potential improvement of the current driving style is beneficial for the individual (reduced premium cost) and the insurers (risk decreases). In reinforcement learning defining a proper reward function is crucial to achieving the goal [15]. Within our project, we want to train a model to drive as safely as possible on the race track. Given the similarity of the tasks, we decide to use the same function for scoring and reward. The challenge on the track and with the sensors is to get a robust function for measurement noise and reference values for safe driving.

Current research in the field of telematics motor insurance pricing comprises driving style extraction and its investigation. For the extraction Gao *et al.* [16, 3] propose speed and acceleration heatmaps using the K-means algorithm, singular value decomposition, or bottleneck neural networks. Again for speed and acceleration data, it was also shown that Fourier analysis decomposition can be used [2]. In the following, we explain an approach for our specific environment and present the scoring function. For the Carrera setting, we use the following notation.

Optical sensor The track comprises 6 optical sensors, which are non-uniformly distributed. We denote them as $optical_i$, $i \in \{0, 1, \cdots, 5\}$, where $optical_0$ corresponds to the sensor at the start/finish line and the order is ascending with regard to the driving direction. The respective positions on the track can be seen in Figure 2.

Segment We subdivide the track into 6 disjoint segments Seg_i, where $i \in \{0, 1, \cdots, 5\}$. The latter sensor names the segment, e.g. Seg_4 is the part of the track between $optical_3$ and $optical_4$.

Lap One lap Lap_j, where $j \in \mathbb{N}$, comprises 6 consecutive segments. To distinguish the same segment in different laps, we write $\mathrm{Seg}_{i,j}$ in this case.

3.1 Data Generation and Exploration

To fulfill the requirements of safe driving, we have to analyze human generated data from the track. We refer to this data set as scoring calibration data. Six optical sensors are mounted to the track and trigger the sending of a data sample. Since the IMU has a higher output rate than we pass the optical sensors, we save a history for every segment. To get the important information, we define a measurement as

- **Optical sensor:** An integer $0, \ldots, 5$ representing one of six sensors, respectively segments. 6 is used as a dummy value to indicate, e.g., a time-out.
- **Segment time:** A floating point number that shows the time needed to get from the start of the segment to its end. It is reported in milliseconds.

- **Sensor data:** Acceleration and angular velocity data in x-, y-, and z-direction for the segment, each with a history of up to 40 measurements. This results in an array of dimensions 6x40.
- **Trackside:** Indicating if the right or left lane was measured.
- **Car:** Shows which slot car model was used.
- **Timestamp:** Unique identifier when the observation was made.

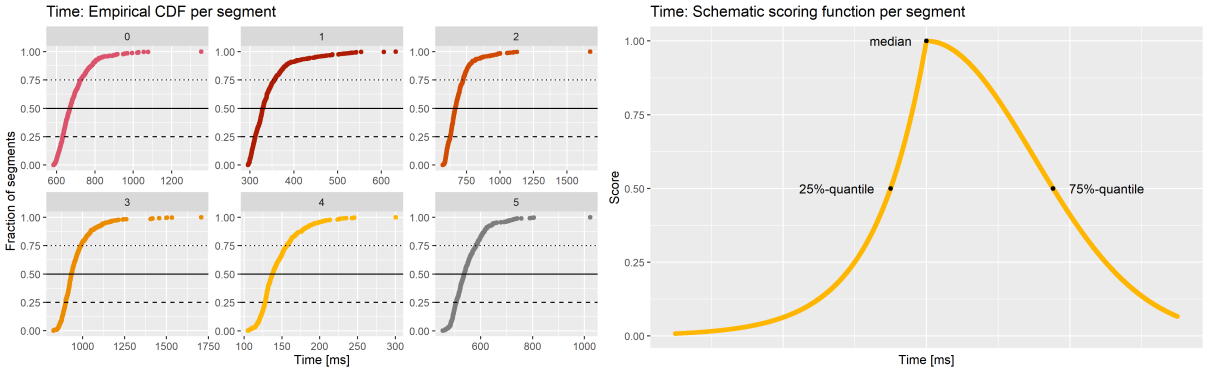
A Python script and Jupyter Notebook on the Raspberry Pi are used to implement the process and send the data to an Azure SQL Database. For data generation, we drive on the track using a standard controller. In total, we gathered over 3,100 human driving samples for the scoring calibration data set. Since the scoring function should be explainable, we decide to use the same family of functions for all segments. However, it should also capture the critical information in as much detail as possible, so the parameters are calibrated individually for every segment. Further, let q^{α} denote an α-quantile of a data set.

Safe driving for a slot car and a real car means some differences: there is no speed limit on the track, no change in weather conditions, and no direct interaction between road users. However, the time needed for a specific and not-too-long route should provide a basic measure in both examples. Faster times appear to be more dangerous due to a higher average speed. But one has to account for corner cases, e.g. stopping the car and then going full speed. To counter such behavior, we introduce a time-out above 2 seconds per segment. Driving too slowly can also pose a risk to other drivers and traffic flow.

Figure 9a visualizes segment times via an empirical cumulative distribution function (ECDF). We observe a similar shape across all segments and a noticeable difference in the time scale that corresponds to the various lengths and difficulties of the segments. The ECDFs are steep up to $q^{25\%}$, between $q^{25\%}$ and $q^{75\%}$ they seem approximately linear and above $q^{75\%}$, meaning for larger times, an edge manifests. This backs the assumption that the median time should reflect safe driving behavior. An additional advantage of the median is its robustness. Times between $q^{25\%}$ and $q^{75\%}$ are still considered to be in a safe range and are assigned values above 0.5. Overall, we want to penalize time deviations from a safe trip, meaning median time, more heavily if they are too fast than too slow. This translates to the following function.

Per Segment Time Score $s_{time,i}$. We use a piece-wise exponential (for being too fast) and scaled Gaussian density (too slow) as a time scoring function. More specifically, we fix the median duration $t_{0,i}$ for Seg_i according to our human driving data and determine which of the functions is used. For an appropriate scaling, we require $s_{time,i}(t_{0,i}) = 1$ showing the maximum value of the score per segment. We define the function as

$$s_{time,i}(t_i) = \begin{cases} \exp\left(\frac{t_i - t_{0,i}}{b_i}\right), & \text{if } t_i < t_{0,i} \\ \exp\left(-\frac{1}{2}\left(\frac{t_i - t_{0,i}}{\sigma_i}\right)^2\right), & \text{otherwise.} \end{cases}$$



(a) Time ECDFs *(b) Time scoring*

Figure 9: (a) Empirical distribution functions showing cleaned per segment data from human driving. 25% (dashed), 50% (solid), 75% (dotted) p values are added. (b) A schematic representation of the time scoring is shown.

For calibration of the parameters b_i and σ_i, we use the human driving data and set

$$s_{time,i}(q^{25\%}_{time,i}) = s_{time,i}(q^{75\%}_{time,i}) = 0.5.$$

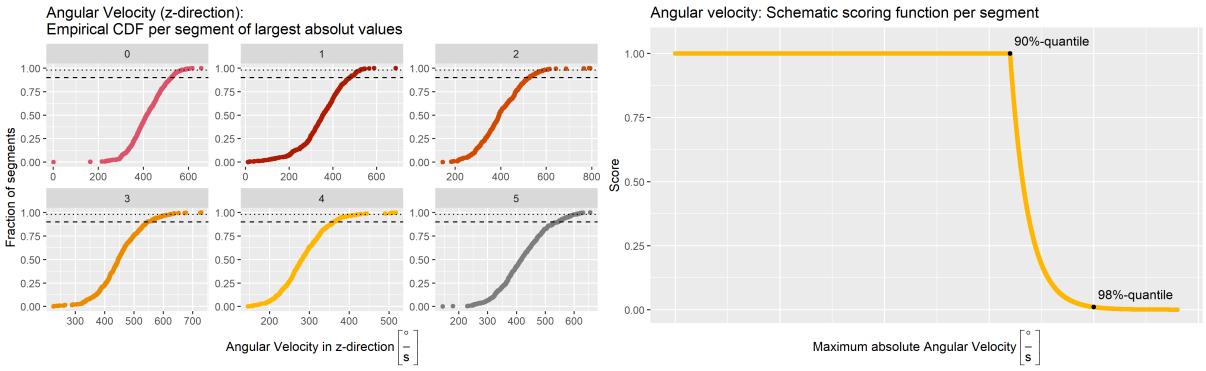
A schematic representation of the scoring shape is shown in Figure 9b and more detailed calculations can be found in Appendix A.2.

Another factor is the angular velocity during a curve. The higher it is, the riskier the driving. On the Carrera track, this is a valuable metric as 5 of the segments include at least one curved part. Angular velocity has up to 40 measurements per segment, so there is a need to summarize this information to have an explainable function. Especially for high absolute values, i.e. for left and right curves, angular velocity provides a good proxy for safety and the risk of derailing. Since the IMU is repeatedly calibrated during data generation, the measurements are reliable and errors are not expected. This allows us to choose the maximum absolute value per segment as a risk indicator and illustrate them in Figure 10a with ECDFs. We observe a beginning edge just above 90% across all six segments, where above 98% angular velocity grows fast. This leads to our assumption that those high values should be considered high risk. In particular, the frequency of angular velocity above 98% would match the number of expected derailings during the data gathering. Overall, a wide range is considered to be relatively safe, and specific penalties for angular velocity below the 90% threshold are hard to define from the data we have. With regards to low values, we assume they are also reflected in higher segment times and sufficiently penalized there. This reasoning is described for the angular velocity score below and the interplay with time in the next Section 3.2.

Per Segment Angular Velocity Score $s_{angv,i}$**.** We define the function to have the best scoring values below the 90%-quantile for maximum absolute angular velocity. Above this critical threshold, we penalize with an exponential decay.

$$s_{angv,i}(a_i) = \begin{cases} 1, & \text{if } a_i < q^{90\%}_{angv,i} \\ \exp\left(\frac{a_i - a_{0,i}}{c_i}\right), & \text{otherwise.} \end{cases}$$

To calibrate c_i, we set $s_{angv,i}(q^{98\%}_{angv,i}) = 0.01$. An example of its shape is depicted in Figure 10b with more detailed calculations in Appendix A.2.



(a) Angular Velocity ECDFs *(b) Angular Velocity scoring*

Figure 10: (a) Empirical distribution functions showing per segment data from human driving. 90% (dashed) and 98 % (dotted) p values are added. (b) A schematic representation of the angular velocity scoring is shown.

3.2 Combined scoring function

With the knowledge of previous sections, we build the combined scoring function. In particular, we utilize the per segment time score as a base measure for safety. It already reflects the average speed and penalizes racing more than slow driving. Therefore, the additional information from angular velocity is included as an indicator for high-risk action in curves. There the risk of derailing is the greatest and should be strictly avoided. To not disturb the extracted information from the time data, we only add the penalty for angular velocity values above $q^{90\%}$.

Per Segment Score $s_{seg,i}$. The total score of Seg_i is the product of time and angular velocity subscores

$$s_{seg,i}(t_i, a_i) = d \cdot s_{time,i}(t_i) \cdot s_{angv,i}(a_i).$$

In case a time-out occurs, the segment score is defined as 0. Thus, $s_{seg,i} \in [0, d]$. In our analysis, we use $d = 100$ throughout the experiments. If we want to emphasize in which lap $j \in \mathbb{N}$ the segment was measured, we write $s_{seg,i,j}$.

There is an interest in not just rating different segments, but the whole track when driving. To achieve a good comparison of several laps, we choose a conservative approach, where low values for some segments affect the entire lap.

Lap score $s_{lap,j}$. We use the geometric mean of the segment scores $s_{seg,i}$, where $i \in \{0, 1, \cdots, 5\}$. For a lap $j \in \mathbb{N}$, we denote

$$s_{lap,j} = \sqrt[5]{\prod_{i=0}^{5} s_{seg,i,j}}.$$

4 Reinforcement Learning

Among the primary objectives of this project was to develop a framework for training a deep neural network to effectively control the steering mechanism of the car aiming for a high score as defined above. The methodology employed to achieve this goal is described in detail in the following section.

4.1 Preliminaries

The Reinforcement Learning Setting Following [17], we outline the fundamental theoretical concepts that our solution builds upon. Formally, we interpret the process of steering the car as a Markov decision process (MDP), defined as a 5-tuple (S, A, T_a, R_a, γ)

- S: The set of states,
- A: The set of actions,
- $T_a(s, s')$: The state transition function,
- $R_a(s, s')$: The reward function, and
- $\gamma \in [0, 1]$: The discount factor.

In this system, a reinforcement learning agent interacts with the environment in a step-by-step fashion: At time step t, the environment is in state s_t, the agent chooses an action $a_t \in A$ and carries it out in the environment, which returns the associated reward r_{t+1} and subsequent state s_{t+1}, according to T and R respectively.

We formalize one iteration of this interaction by defining a transition as a 4-tuple (s, a, r, s'), with

- s: A state at any point in time,
- a: An action carried out after observing o,
- r: The reward received after carrying out action a at state s, and
- s': The state following s.

The goal in reinforcement learning is to find a policy $\pi : S \to A$ that maps states to actions, such that the agent acting according to the policy maximizes the expected discounted future reward

$$\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_{t+1}]$$

Such a policy is called an optimal policy π^*.

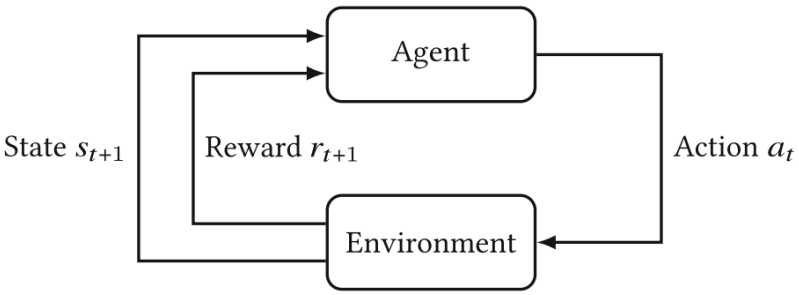


Figure 11: Agent and Environment Interaction [15].

Deep Q-Learning Each policy π has an associated Q-function which maps a state-action pair (s, a) to its *quality*, the expected sum of discounted reward when executing a at state s and following the policy π for all subsequent steps, i.e.

$$Q^{\pi}(s,a) = \mathbb{E}_{T_a(s,s')}[\sum_{i=0}^{\infty} \gamma^i r_{t+i} | s_t = s, a_t = a].$$

Q-Learning approaches the task of reinforcement learning by approximating the optimal Q-function $Q^* := Q^{\pi^*}$. If this function is known, the corresponding optimal policy π^* can easily be recovered as

$$\pi^*(s) = \arg\max_{a \in A} Q(s,a).$$

In the case of both finite observation and action spaces with manageable sizes, this function can be stored as a simple lookup table, storing a value for each pair $(o, a) \in O \times A$.

We refer the reader to [15] for details on this approach, though it is not applicable in our case. This is because the state space is not only high-dimensional but also continuous. Instead, we use a neural network architecture to approximate the Q-function. Mnih *et al.* introduced this method known as Deep Q-Learning [18] and showed that the simple Algorithm 1 converges to the optimal Q-function given sufficient data.

while *not converged* **do**
  Sample transition (s, a, r, s');
  Predict $\hat{q} \leftarrow Q_\theta(s, a)$;
  Compute target $q \leftarrow r_j + \gamma \max_\alpha Q_\theta(s, \alpha)$;
  Perform gradient update on parameters θ using the mean squared error loss $MSE(q, \hat{q})$
end

Algorithm 1: Training a Deep Q-Learning Agent [18].

CQL: A Batch RL Framework Though in the original formulation, new transitions were generated on the fly, i.e. using the current Q_θ to interact with the environment, generating new transitions during training, this approach has been used successfully training

purely on stored transitions generated with an *arbitrary* agent [19].

Though this data can be generated by an agent acting with an arbitrary policy, any methods using such data run into the problem of distributional shift: During training, the distribution of predicted optimal actions differs from the distribution of actions in the training set. This leads to unreliable predictions of the Q-network outside of the support of the training data action distribution. With Conservative Q-Learning (CQL), Kumar *et al.* introduced a simple yet powerful adaptation of the standard Batch Q-learning framework that mitigates the problem by adding a simple regularization term to the loss. We refer the reader to [6] for details.

4.2 Method: Implementing CQL

Traditional reinforcement learning strategies involve the interaction with the environment in real-time and require the exploration of the environment by allowing the agent to perform a wide range of actions on it. Moreover, RL frameworks are notorious for requiring a high number of training iterations until satisfactory results are achieved. These limitations combined make these traditional methods unattractive for our use case: A wide range of actions cannot be carried out safely on the track. If the chosen voltage is too low, the car stalls on the track such that the corresponding step will never be completed. Conversely, setting it too high can easily lead to the car derailing. Apart from potential damages to the IMU in the latter case, both are unacceptable since they would require human intervention in the form of setting the car back on the track and restarting the training, leading to potentially unfeasibly high training time requirements.

On the other hand, producing a high number of training samples for a batch RL setting is simple: After experimentally determining the safe range of actions, data is sampled from an agent behaving according to a policy picking random values in the safe range.

4.2.1 Defining step, action, state and reward

Having established the theoretical we now consider the manner in which the concepts are formally defined within the context of our particular scenario.

Reward The reward we want to optimize for is the score over one segment, as defined in Section 3.

Time Step At each time step, a reward must be returned to the agent, calculated based on the segment-wise scoring methodology outlined in Section 3. The environment operates in a step-by-step manner, where each step begins upon the passage of a light sensor, following the selection of an action by the agent, and ends upon the passage of the subsequent light sensor.

Action Our definition of the action set A is straightforward: Our physical set-up allows us to set the motor driver to multiples of $\frac{1}{256}$ parts of the total available voltage. Aiming for a compromise between a fine-grained control and a reduced action space to ease learning,

we decided to further divide the range of possible voltages into 64 discrete actions $A = \{a_0, ..., a_{63}\}$, where a_i corresponds to setting the voltage to $\frac{4i}{256}$ of maximum.

State Defining the set of states S requires some more involved careful considerations. In particular, this state must contain sufficiently expressive features to capture all relevant information for the agent to make informed decisions. We have identified three factors that influence the behaviour of the car on the next segment, and thus the best action to choose.

Position Different segments require different actions, e.g. a curvy segment requires a lower speed than a straight one. We can include exact information about the car's position on the track via the optical sensor at which a step in the environment starts.

Motion of the Car The faster the car moves into the segment, the lower the voltage setting on the segment needs to be to achieve a desired time and vice versa. Since we cannot directly measure the velocity of the car, we include motion information via the measurements provided by the IMU. More precisely, as soon as a step begins, all readings of the IMU are buffered. The resulting buffer, a six-dimensional finite time series, is provided as a part of the next state.

Global System State As we have discovered in 2.2.1, we cannot assume the system to react the same way even if the same sequence of control signals is provided. For the purposes of controlling the car precisely, the current *efficiency* of the system needs to be taken into account, which expresses itself in a relationship between action and resulting segment time given the same initial velocity. Again, this parameter is not tractable directly in our setting. It is also not sufficient to only include the very last action-time pair, as this is influenced by the initial speed for the previous segment. However, including several past actions together with the resulting times allow the model to infer their relationship in the near past more robustly, providing an indication of the efficiency.

Keeping these requirements in mind we combine them to define a state s as a 4-tuple $(optical_i, H_I, H_A, H_T)$, with:

- $optical_i, i \in \{0, .., 6\}$: Current optical sensor 0-5 (6 as dummy value, i.e. no sensor),
- $H_I \in \mathbb{R}^{6\times k}$: The last k readings of the IMU,
- $H_A \in A^l$: The last l performed actions, and
- $H_T \in (0, \infty)^l$ The last l segment times.

For all applications, we decide to store up to $k = 40$ readings of the IMU, resulting in a history of 2 seconds at our sampling rate of 20 Hz and $l = 10$ segemnt actions and resulting times. Padding is applied for shorter actually observed sequences in either case.

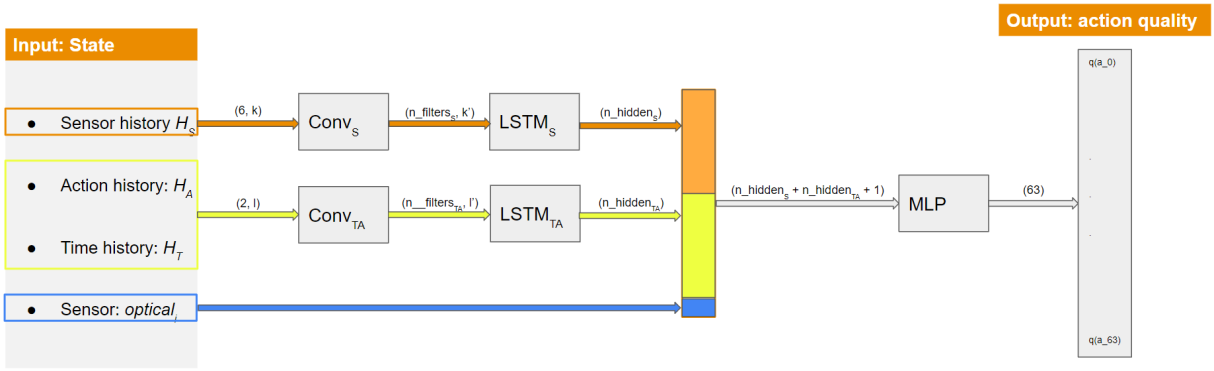


Figure 12: Overview of the model architecture.

4.2.2 Data Generation

Having established the format of the data set, as well as the fact that it can be generated using an arbitrary model, we need to generate a data set for off-line training in the appropriate format. We use our IoT infrastructure as outlined in Section 2 to gather and store the data. To ensure the covering of a wide range of actions we choose a model that picks the actions uniformly at random in an experimentally determined safe range, omitting both stalling and derailing of the car. Driving episodes are limited to 100 laps followed by a cool-down time of 5 minutes to reduce the heating effects observed in 8. Each sample is prepared on the RPI in the correct format and sent into the cloud database on MS Azure.

4.2.3 Model Architecture

An overview of our designed architecture is depicted in Figure 12. Sensor History H_I as well as time history H_T and action history H_A all occur naturally as time-series data. Recently, state of the art approaches have been achieved with CNN-LSTM architectures on time series processing in general [20] and IMU data in particular [21, 22]. This success inspired us to choose an architecture similar to [22]; however, we remove the output layer of this architecture to produce a feature encoding instead of class predictions. We observe that a single convolutional layer followed by a single LSTM is sufficient to produce results competitive with human driving while keeping the network architecture simple enough to allow good training behavior.

We employ two independent branches of the CNN-LSTM pipeline: The first one handles the raw IMU data directly and thus operates on six-dimensional time series measurements. Since the time history and action history are sampled at the same rate, which is different from the one of the IMU and only contains relevant information of the state of the environment via their relationship, we stack the time history and action history to form a two-dimensional time series measurement which is fed into the second branch.

In each of them, first a 1-dimensional convolutional operator is applied. The sequence transformed this way is then fed into the LSTM, whose last cell's hidden state is used as the feature encoding of the entire sequence.

The features of both branches as well as the ID of the last optical sensor are concatenated to produce the full feature encoding of the state. This encoding is fed into a final, small MLP to predict a per-action prediction vector of the action qualities $(q(a_0), ..., q(a_{63}))$.

During training it is required to predict qualities for each action, as the loss is computed with respect to the action a that occurs in the stored transition. At test time, the action with the maximal associated predicted quality is chosen by the agent.

4.2.4 Implementation Details

Used Libraries Our model is implemented in PyTorch [23], the implementation of the environment builds upon OpenAI's Gym library [24]. In order to implement Q-learning we use d3rlpy [25], a high level library implementing the CQL training schedule, while still being flexible enough to use our custom neural network for state feature extraction.

Hyperparameters and Training The CQL training scheme, like many reinforcement learning strategies in general and batch methods in particular is hard to optimize and notoriously suffers from high noise during training. We mitigate this problem by several design choices: We find that even with a small number of parameters the information we aim to include with the definition of the state can be extracted. We expand on these findings in Section 4.3. Further, we follow previous authors in using a normalization of the rewards and do not generate regression targets with the most recent parameters, but instead maintain a copy of the Q-network, the target net, that is only updated after carrying out multiple training steps. The number of steps carried out before updating is called the target update interval. We summarize the hyperparameters with which we achieve the best results in Appendix A.3.

A rather uncommon choice in comparison to other Reinforcement Learning tasks is our low discount factor of $\gamma = 0.3$. This further helps stabilizing the training procedure by placing a higher relative weight to the immediate reward compared to the much more uncertain prediction of the next state value. In many applications, such a low discount factor would be unacceptable, since a wrong action can lead to catastrophic failure of the agent many steps later. A classical example is the use of reinforcement learning for training an agent to play Atari games, where a step typically consists in one new frame, leading to a duration of only a few milliseconds. A wrong input can lead to failing a game many frames later [18]. However, a step in our environment takes quite a long time compared to other methods, leading to an increased possibility to react to whatever state the last segments action results in.

4.3 Experimental Analysis

Of particular interest in our experiments were two questions:

- Does the model take into account each part of the state as intended?
- How well does the model perform in comparison to human drivers?

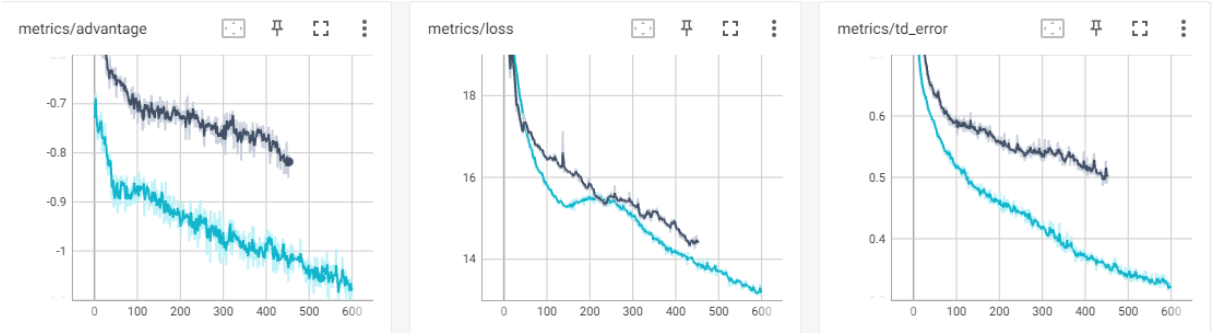


Figure 13: Validation advantage, loss and TD-error of a model trained with ω_z only (black) compared to a model trained with a_x and a_y additionally (blue).

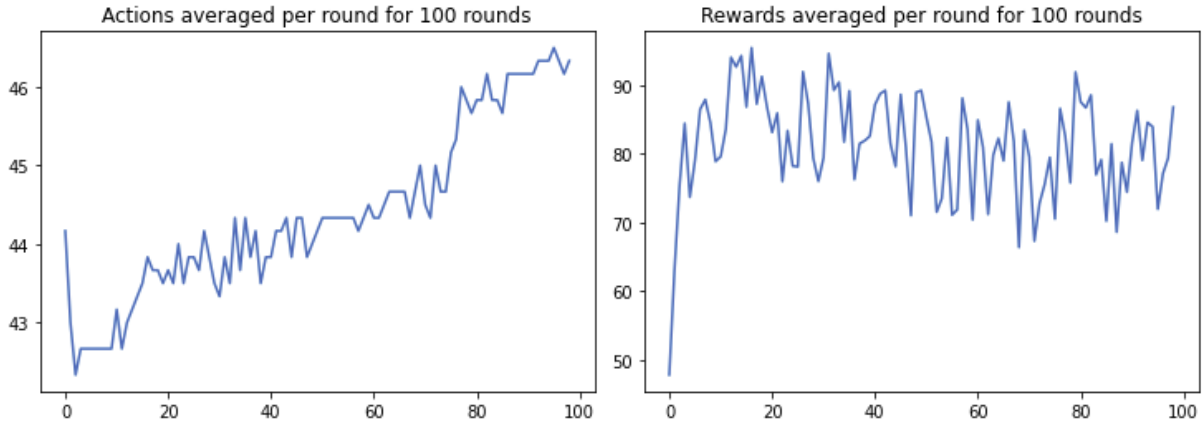
Metrics We conduct both off-line and on-line experiments. For off-line experiments, we mainly take into account the CQL loss as defined in [6], the advantage which intuitively provides an estimate of the amount by which the resulting policy could outperform the agent used in data collection and the temporal difference (TD) error which is a measure of the variability of the predictions over one time step. In both cases, lower values are desirable. We refer the reader to [17] for detailed explanations and definitions of both metrics.

In order to approach the first question, more particularly the question how well the model takes IMU data into account, we trained several variants of the model using only part of the measurements. Here, we denote by a_d, ω_d the acceleration and angular velocity in dimension d, respectively. We achieve the best performance in terms of advantage and TD error computed over the validation set by using a_x, a_y, ω_z. Including the measurements ω_x, ω_y and a_z did not aid the training of the model. This is easily explained from a theoretical point of view: The track the car is moving on is planar and orthogonal to the direction of gravity. Therefore, ignoring noise and physical imperfections of the track, we expect $a_z = 1$ and $\omega_x = \omega_y = 0$ constantly. Conversely, the importance of a_x, a_y, ω_z has been discussed in-depth in Section 3. Figure 13 shows a comparison between two models, one with only angular velocity in z and one with additional acceleration data in x and y. All hyper-parameters were kept the same. The visible performance difference indicates that even though the reward is influenced only by angular velocity, the acceleration in x and y direction as measured in our framework are still highly predictive features for optimal control and capture the motion of the car well despite the noise in measurements we identified.

We also aimed to explore the model's capability of inferring the systems global state as discussed in 2.2.1. To this end, we deployed the model and tracked the actions chosen by the agent over time. Figure 14a shows the average action the agent took per lap during an episode of the maximum episode length in the training set of 100 laps, starting on a track in a cold state. We clearly see the upwards trend similar in shape to the increase in time we observed with a constant voltage setting seen in Figure 8. This shows that the model learns to compensate for the performance degradation that comes with heating by gradually increasing the voltage on the track in order to achieve rewards that are stable on a high level of 70-95 as depicted in 14b. We deduce that it is capable of appropriately

considering the time and action history of a state. We attribute the worse performance on the first few segments to the skew in our data set towards a heated track. Since it is more time-economical to record longer episodes, states from a cold track are underrepresented in the data set, potentially explaining worse performance of the Q-network on these instances.

Finally, we compare the models score with that of human drivers: We observe that the model achieves an average segment score above 86, outperforming 84 percent of segments of the human-controlled samples in our data set.



(a) Average actions chosen during 100 laps. *(b) Average rewards received during 100 laps.*

Figure 14: Performance of the fully trained CQL agent.

5 Dashboard

To visualize the driving behavior of the reinforcement model in real-time, we develop a Microsoft Power BI dashboard. Since the pipeline from Section 2.1.4 proves to be with negligible delay and a lap time typically above three seconds, we can focus on immediate per lap information. One of our goals is to make the concept of telematics insurance approachable and to rate a safe driving behavior. Therefore, we need a simple representation of the key metrics. As the monitored car is driven via the reinforcement model, we also want to include its performance compared to human driving.

In Figure 15 the components of the dashboard are shown. Lap score and time are provided for the last lap and a history of them is plotted as a bar chart. The score is also compared to our databases from the human data generation and the automated, randomized model. It shows the percentage of observations that were below the reinforcement score. Another important aspect is to potentially improve the current performance on a more granular level. For this, we include boxplots of the score for each segment. Additional information about the current car and the trackside is also provided.

Those metrics and visuals will be of further importance, once both tracksides are used and monitored separately. Especially, if one is steered by the reinforcement model and the other one by a human driver. In our Carrera environment, we assume this will add an aspect of gamification and leads to a better understanding of telematics insurance. The dashboard helps to channel the information.

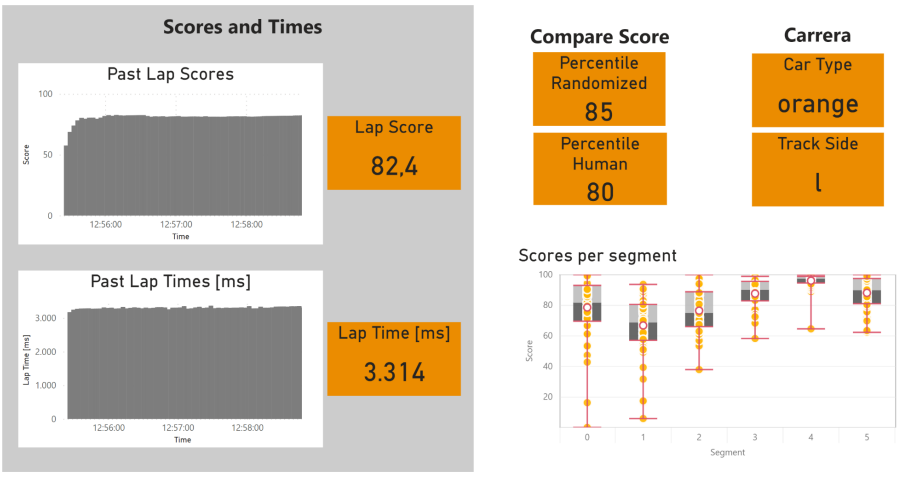


Figure 15: The real-time monitoring dashboard of the reinforcement model.

6 Discussion and Outlook

The IoT system we built poses a successful implementation of the IoT ecosystem layer model. We find that this model provides an abstraction of the components from sensor data to cloud-based solutions while being flexible enough to be tailored to the specific needs of our project. Our implementation has proven to be consistent and powerful enough to allow real-time data monitoring and control of the car. We find that the Conservative Q-Learning framework can be used successfully to leverage a static data set in a real world setting with limited possibilities for live exploration. This work, however, presents a prototype with several chances to improve and build upon.

IoT Environment Our experiments show a significant decrease in speed over time even though the signal sent via the Raspberry Pi was kept constant. We conjecture that this can be attributed mostly to heating of the motor driver, though further experiments are needed to understand and quantify this effect and take appropriate counter-measures, e.g. by replacing affected parts. Another improvement we propose is the placing of the sensor inside the car instead of on top, potentially opening up the possibilities for other track configurations including bridges.

Scoring The biggest limitation of the scoring function as we defined it is that it can only be computed over pre-defined segments. Thus, it can only be evaluated once an optical sensor is passed and is dependent on the configuration of the track, as well as the lane the car is driving in. We believe that future approaches could benefit from restricting the input to the scoring function to IMU data, e.g. by learning to predict for any given IMU history the probability of an imminent crash. Such a solution could provide rewards that can be evaluated at any point in time, allowing for more fine-grained control by defining steps at a higher rate than the passage of optical sensors. However, this also incurs higher demands on latency of the IoT components.

Reinforcement Learning The model we present achieves a competitive score in comparison to human drivers. However, due to the uniqueness of the task no benchmarks or alternative tasks exists to compare the performance with other documented approaches. Experiments that compare different models and training methods are needed to acquire more insights into the benefits and shortcomings of several methods including ours in this setting. Due to time constraints, we were not able to refine the model with real-world interaction using standard reinforcement learning techniques. It remains to be seen whether such a training scheme can further improve performance in our case.

We conclude that telematic policies reveal great potential for both customers and insurance companies. In terms of safety, a broader application is desirable, but this has as a prerequisite a proper quantification of the actual driving risk. Further advances in car connectivity might act as an acceleration for telematics insurance.

References

[1] *European insurance in figures 2020 data*. URL: https://insuranceeurope.eu/publications/2569/european-insurance-in-figures-2020-data.

[2] Wiltrud Weidner, Fabian W.G. Transchel, and Robert Weidner. "Telematic driving profile classification in car insurance pricing". In: *Annals of Actuarial Science* 11.2 (2017), 213â€"236. DOI: 10.1017/S1748499516000130.

[3] Guangyuan Gao, Shengwang Meng, and Mario V. Wüthrich. "Claims frequency modeling using telematics car driving data". In: *Scandinavian Actuarial Journal* 2019.2 (2019), pp. 143–162. DOI: 10.1080/03461238.2018.1523068. eprint: https://doi.org/10.1080/03461238.2018.1523068. URL: https://doi.org/10.1080/03461238.2018.1523068.

[4] *Telematics devices prompt changes in driving behavior - insurance research*. Nov. 2015. URL: https://www.insurance-research.org/sites/default/files/downloads/TelematicsNR11172015%5C%20%5C%282%5C%29.pdf.

[5] Pierre Francois and Théo Voldoire. "The revolution that did not happen: Telematics and car insurance in the 2010s". In: *Big Data & Society* 9.2 (2022). DOI: 10.1177/20539517221142033. eprint: https://doi.org/10.1177/20539517221142033. URL: https://doi.org/10.1177/20539517221142033.

[6] Aviral Kumar et al. "Conservative Q-Learning for Offline Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 1179–1191. URL: https://proceedings.neurips.cc/paper/2020/file/0d2b2061826a5df3221116a5085a6052-Paper.pdf.

[7] Somayya Madakam et al. "Internet of Things (IoT): A literature review". In: *Journal of Computer and Communications* 3.05 (2015), p. 164.

[8] Cisco Press. "IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things". In: (2017).

[9] Pradyumna Gokhale, Omkar Bhat, and Sagar Bhat. "Introduction to IOT". In: *International Advanced Research Journal in Science, Engineering and Technology* 5.1 (2018), pp. 41–44.

[10] Malihe Asemani, Fatemeh Abdollahei, and Fatemeh Jabbari. "Understanding IoT Platforms : Towards a comprehensive definition and main characteristic description". In: *2019 5th International Conference on Web Research (ICWR)*. 2019, pp. 172–177. DOI: 10.1109/ICWR.2019.8765259.

[11] Wikipedia. *Slot car - Wikipedia, The Free Encyclopedia*. [Online; accessed 09-February-2023]. 2023. URL: http://en.wikipedia.org/w/index.php?title=Slot%5C%20car%5C&oldid=1138317880.

[12] Jacek Wernik. "Investigation of Heat Loss from the Finned Housing of the Electric Motor of a Vacuum Pump". In: *Applied Sciences* 7.12 (2017). ISSN: 2076-3417. DOI: 10.3390/app7121214. URL: https://www.mdpi.com/2076-3417/7/12/1214.

[13] Edson da Costa Bortoni. "Are my motors oversized?" In: *Energy Conversion and Management* 50.9 (2009), pp. 2282–2287. ISSN: 0196-8904. DOI: https://doi.org/10.1016/j.enconman.2009.05.004. URL: https://www.sciencedirect.com/science/article/pii/S0196890409001824.

[14] Edison Gundabattini et al. "A review on methods of finding losses and cooling methods to increase efficiency of electric machines". In: *Ain Shams Engineering Journal* 12.1 (2021), pp. 497–505. ISSN: 2090-4479. DOI: https://doi.org/10.1016/j.asej.2020.08.014. URL: https://www.sciencedirect.com/science/article/pii/S2090447920301854.

[15] R.S. Sutton and A.G. Barto. *Reinforcement Learning, second edition: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, 2018. ISBN: 9780262352703. URL: https://books.google.de/books?id=uWV0DwAAQBAJ.

[16] Guangyuan Gao and Mario Wüthrich. "Feature extraction from telematics car driving heatmaps". In: *European Actuarial Journal* 8 (Oct. 2018). DOI: 10.1007/s13385-018-0181-7.

[17] Aske Plaat. *Deep Reinforcement Learning*. Springer, 2022. ISBN: 978-981-19-0637-4. DOI: 10.1007/978-981-19-0638-1. URL: https://doi.org/10.1007/978-981-19-0638-1.

[18] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[19] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. "An Optimistic Perspective on Offline Reinforcement Learning". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 104–114. URL: https://proceedings.mlr.press/v119/agarwal20c.html.

[20] Ioannis Livieris, Emmanuel Pintelas, and P. Pintelas. "A CNN-LSTM model for gold price time series forecasting". In: *Neural Computing and Applications* 32 (Dec. 2020). DOI: 10.1007/s00521-020-04867-x.

[21] Narit Hnoohom, Anuchit Jitpattanakul, and Sakorn Mekruksavanich. "Real-life Human Activity Recognition with Tri-axial Accelerometer Data from Smartphone using Hybrid Long Short-Term Memory Networks". In: *2020 15th International Joint Symposium on Artificial Intelligence and Natural Language Processing (iSAI-NLP)*. 2020, pp. 1–6. DOI: 10.1109/iSAI-NLP51646.2020.9376839.

[22] Ronald Mutegeki and Dong Seog Han. "A CNN-LSTM Approach to Human Activity Recognition". In: *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. 2020, pp. 362–366. DOI: 10.1109/ICAIIC48513.2020.9065078.

[23] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[24] Greg Brockman et al. "OpenAI Gym". In: *CoRR* abs/1606.01540 (2016). arXiv: 1606.01540. URL: http://arxiv.org/abs/1606.01540.

[25] Takuma Seno and Michita Imai. "d3rlpy: An Offline Deep Reinforcement Learning Library". In: *CoRR* abs/2111.03788 (2021). arXiv: 2111.03788. URL: https://arxiv.org/abs/2111.03788.

[26] *Arduino Nano*. URL: https://store.arduino.cc/products/arduino-nano?gclid=CjwKCAiA0JKfBhBIEiwAPhZXD1hlnY01f8PP6tkT2Rc4YxEYeqz4zInkZrR6qqrYiaj7iSLaFSujzRoC-2AQAvD_BwE.

[27] Ryan Chan. *How to use the L298N motor driver*. Dec. 2022. URL: https://www.hackster.io/ryanchan/how-to-use-the-l298n-motor-driver-b124c5.

[28] Codefls et al. *H-bridge with L298N motor driver*. Nov. 2019. URL: https://forum.fritzing.org/t/h-bridge-with-l298n-motor-driver/7711.

A Appendix

A.1 IoT Environment

A.1.1 Electrical Circuit

The structure of the Carrera track and the controller as identified in Section 2.1.1 is depicted in Figure 16. Figure 17 shows the simplified version of the figure with the slot car. The controller handle works like a switch with a potentiometer. If one releases the handle, the electric circuit becomes completely interrupted, thus current doesn't flow on the track. If one presses the handle, the total circuit is connected and the slot car drives. Depending on how much the handle is pressed, the resistance in the controller is changed, thus the voltage for the slot car also changes. This leads to a different speed of the DC motor inside the slot car.

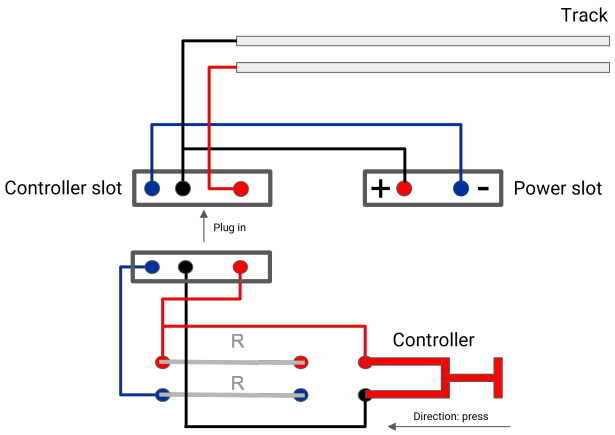


Figure 16: Electric circuit of the Carrera track and the controller.

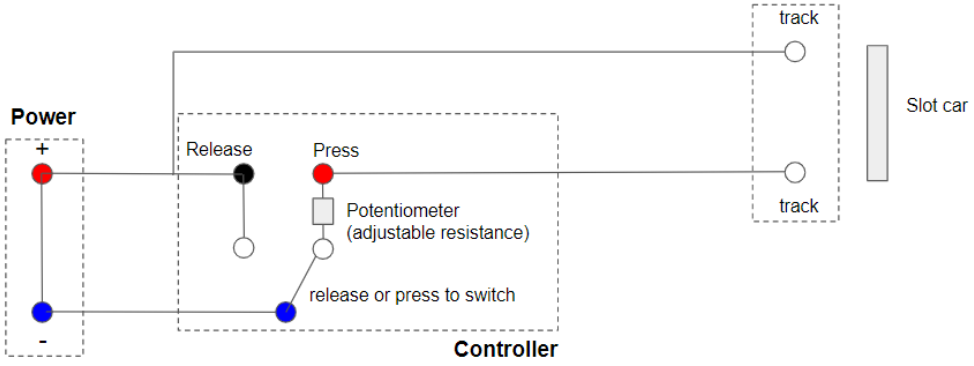


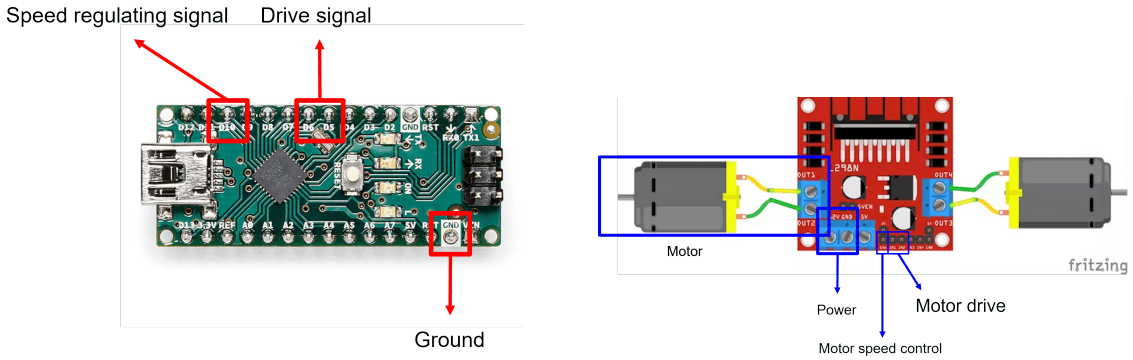
Figure 17: Electric circuit of the Carrera track, the controller and the slot car.

A.1.2 Wiring

This section shows how the electronic components should be wired.

Arduino: Nano We use a total of 3 pins from the Arduino. Two of them are used to send motor drive signals, and the other to send motor speed regulating signals. The pins are depicted in Figure 18a.

Motor Driver: L298N The motor driver L298N receives power from an external power supplier via power pins on the left bottom side. There are in total four logic pins on the right bottom side, and the left two of them decides whether the motor, that is connected via left side of the motor driver, drives or stop, and the right two logic pins the right side connected motor. On both end sides of the logic pins, enable pins are positioned. With the enabled pins, the speed of the motor can be regulated. The pins are depicted in Figure 18b.



(a) Arduino Nano *(b) Motor Driver L298N*

Figure 18: Description of the pins of the Arduino and the L298N that used in the project [26, 27].

Overall wiring Based on Figures 17 and 18, the overall components should be wired as in Figure 19. The Arduino and the motor driver are simply connected with wires. Due to the specific formats of the controller and power slots and the power supplier, the other connections were not straight forward. For convenience, we cut the wire of the original power supplier and the controller as in Figure 20 and used them to connect the components.

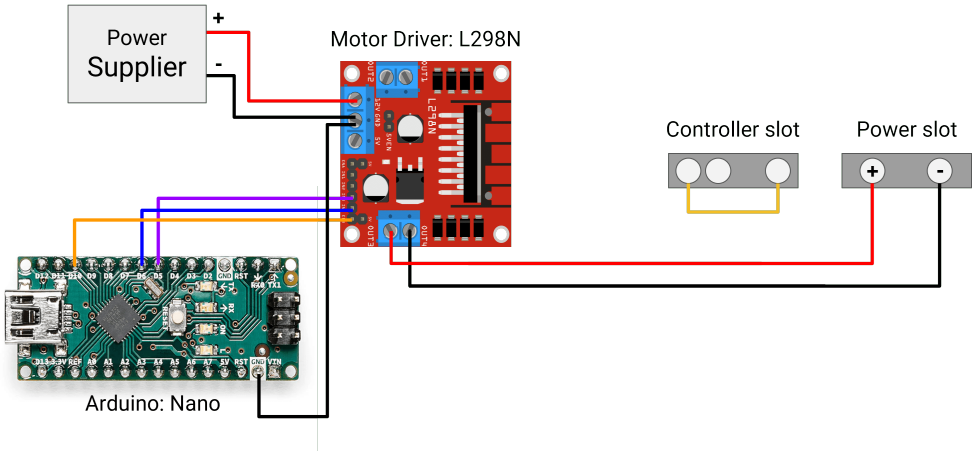


Figure 19: Wiring between Arduino, motor, driver, and power supplier [26, 28].

Figure 20: The power supplier and the controller after cutting.

A.2 Scoring

Calibrating an Exponential We refer to an exponential function as $f : \mathbb{R} \longrightarrow \mathbb{R}$, $t \mapsto \exp\left(\frac{t-t_0}{b}\right)$ with one parameter b and a constraint $f(\tilde{t}) = \beta$, where $t_0, \tilde{t} \in \mathbb{R}$, $b \in \mathbb{R}\backslash\{0\}$, and $\beta \in (0, 1)$. To determine the value of b, we see that

$$\exp\left(\frac{\tilde{t} - t_0}{b}\right) = \beta \iff b = \frac{\tilde{t} - t_0}{\log(\beta)}.$$

Calibrating a Gaussian We use the Gaussian density function $f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma_i}\right)^2\right)$, with $\mu \in \mathbb{R}$ and $\sigma \in \mathbb{R}_{>0}$. To normalize the function (require $f(\mu) = 1$), we divide by $f(\mu)$ and define $g(x) := f(x)/f(\mu) = \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma_i}\right)^2\right)$. Now, given μ and a value $\tilde{x} > \mu$, we want to find σ, such that $g(\tilde{x}) = 0.5$. We use the Full Width Half Maximum from Remark 1 and note that $FWHM = 2 \cdot (\tilde{x} - \mu)$, since

the normal density function is symmetric. Then we see that

$$\frac{FWHM}{2} = \sqrt{2\ln 2} \cdot \sigma \Rightarrow \sigma = \frac{\tilde{x} - \mu}{\sqrt{2\ln 2}}.$$

Remark 1 *Full Width Half Maximum (FWHM) is the difference between two values of the independent variable at which the dependent variable is equal to half of its maximum value. For the density function of the normal distribution, we get*

$$FWHM = 2\sqrt{2\ln 2} \cdot \sigma.$$

A.3 Reinforcement Learning

Conv. filters	Conv. Kernel Size	LSTM Hidden Features
2	2	4

Table 2: Hyperparameter Choices in the Sensor Branch.

Conv. filters	Conv. Kernel Size	LSTM Hidden Features
2	2	12

Table 3: Hyperparameter Choices in the Time-Action Branch.

Hidden Layers	Hidden Size
2	256

Table 4: Hyperparameter Choices in the Final MLP.

Batch Size	Target Update Interval	Discount Factor
64	512	0.3

Table 5: Hyperparameter Choices for Training.

Learnig Rate	Decay Rate (1st Moment)	Decay Rate (2nd Moment)
1e-4	0.9	0.99

Table 6: Hyperparameter Choices for the Adam Optimizer.