F1TENTH: An Over-taking Algorithm Using Machine Learning

Jiancheng Zhang
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh, United Kingdom
jz76@hw.ac.uk

Hans-Wolfgang Loidl
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh, United Kingdom
h.w.loidl@hw.ac.uk

2023 28th International Conference on Automation and Computing (ICAC) | 979-8-3503-3585-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICAC57885.2023.10275226

***Abstract*—In this paper we report on the development of a novel over-taking algorithm of cars in a simulated environment. The algorithm uses machine learning techniques, specifically recurrent neural networks (RNNs) and dense neural networks. We take LiDAR data, current speed, and current steering angle as input and produce control information in the form of output speed and steering angle. We obtain the training data by monitoring a human driver over-taking a car, controlled by a model predictive control (MPC) algorithm.**

After having trained several models (using Keras and Tensorflow), two unseen racetracks are used for evaluating the models. We set up experiments on these two racetracks in the simulator, to test whether the models can overtake in different and unseen cases. The best model (simple RNN) can pass 84 out of 90 cases on both racetracks. We identify faster training and lower risk of overfitting as key advantages for RNNs compared to other NNs we explored.

***Index Terms*—Machine learning, Neural Networks, F1TENTH, Racing, Over-taking.**

I. Introduction

Autonomous control of vehicles is a rich topic for the application of artificial intelligence and machine learning techniques. In this paper we focus on the problem of over-taking a car in the context of a Formula 1 race track simulation engine, F1TENTH. We apply advanced machine learning techniques, and compare their effectiveness in controlled experiments.

The main aim of this work is to develop a machine learning algorithm for over-taking manoeuvres in a multi-agent racing scenario within a simulated environment. To achieve this main aim, we also extend the existing F1TENTH simulator [1] to a multi-agent simulator (see Section III).

Our implementation uses the Keras library [2] in the TensorFlow framework [3] for machine learning. This framework provides numerous useful functions that can build a range of different neural network (NN) models. In this work, we explore three classes of NNs: dense NNs, recurrent NNs, and LSTM NNs. In these NNs, the input value of the over-taking algorithm is LiDAR data, current velocity, and current steering angle The output value is a driving command, which contains desired speed and desired steering angle. We train these neural network (NN) models to perform over-taking in a simulated environment. Finally, we test the over-taking algorithm in the extended F1TENTH Simulator, and we measure how many failures and successes in over-taking are observed.

Our main contributions are[1]:

- We report on the training and testing of several neural-network-based over-taking algorithms.
- We measure success and failure of these algorithms in simulations of over-taking, using two unseen race-tracks.
- We identify the main reasons for the recurrent neural network model performing best.

II. Literature Review

In contrast to *modular systems*, which work as a pipeline that processes information step-by-step [4], *end-to-end planning* is more like a black box. It generates controlling commands directly from sensors' output data [5]. A range of machine learning approached can be used in end-to-end planning, such as deep learning [6] or reinforcement learning [7], [8], [9]. The quality of the machine learning model generated typically depends on the quality of the dataset. The generated model can then solve the specific problem directly. The shortcoming is the difficulty to verify or debug the model. In this paper, we are going to use an end-to-end approach for developing a machine learning model that can perform over-taking manoeuvres, as visualised in Fig. 1(b).

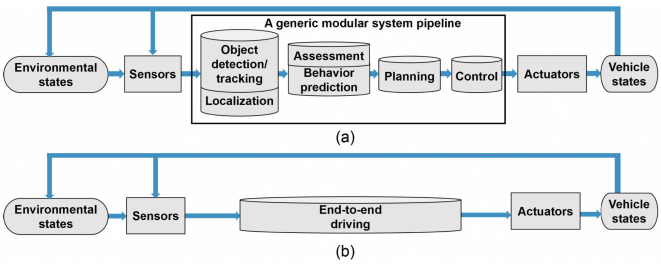


Fig. 1: Difference between modular approach and end-to-end approach (from [10]).

One of the simplest autonomous driving algorithms is the "Follow the Gap Method" (FGM) [11], which is designed without any prior knowledge of the race-track. The FGM will output steering angle by processing LiDAR data (with timestamps) directly. First, after given the data array, it will model obstacles as bubbles, by adding an extra boundary to

[1]A presentation video is available here: https://youtu.be/LjTMYoEPjOk

obstacles of half the size of the vehicle. This reduces the vehicle to just one point and decreases the problem complexity to find the maximum gap in the new array. After the maximum gap is found, the angle between gap centre and current heading direction is calculated, so that the vehicle will start heading toward this gap. The advantage of this algorithm is that its time complexity is linear. The disadvantage is that if the maximum gap is a U-shape dead-end, the algorithm will fail, because it has no knowledge about the places LiDAR data cannot detect. FGM was used as controlling algorithm in the PORTO 2018 F1TENTH head-to-head racing and achieved championship.

For the machine learning approach, the work in [12] combined long short-term memory (LSTM) with convolutional neural network (CNN), which is used to process images from the TORCS simulator and to output driving commands by applying a softmax function to the output of the LSTM. Training data is generated by a human driver and contains 137,500 frames where frames are recorded continuously. Each frame is labelled with the corresponding driving commands, and there are nine types of driving commands. The result shows that the model has 86.94% accuracy on training data and 85.67% accuracy on validation data where accuracy is measured as a match between driving command and a human driver's decision. Compared with no LSTM joined, the result is 84.19% accuracy on training data and 79.76% accuracy on validation data, therefore, with LSTM used, the performance of the algorithm increased.

The work in [13] combined a LSTM with CNN as a model but uses the F1 2019 simulator, which has a more photo-realistic and accurate physics system. The training dataset has 12000 frames, which are collected by manually driving. The frames in both datasets are labeled with synchronised steering angle and acceleration. The result shows that the model has 0.0368 root mean square error on the testing dataset. Moreover, they also applied the model to a real F1tenth car. After collecting human driving data on the real car and training the model again, the model still performs well, which has 0.14 root mean square error on the test racetrack.

III. F1TENTH SIMULATOR

The F1TENTH Simulator [1] is a lightweight two-dimensional simulator designed for a F1TENTH car, and this simulator is developed on top of the Racecar Simulator [14], which is designed for the MIT Racecar [15]. This simulator runs on Ubuntu 16.04 or 20.04 with ROS Melodic or Noetic. It uses C++ and Python 3 as development languages. Our work uses this simulator as a basis. The reason we choose this simulator is that a two-dimensional world is sufficient for testing autonomous racing algorithms because the car's movement is limited to one plane and the LiDAR sensor can only detect one plane. Moreover, the simulator runs on top of the ROS framework, which is the same as the F1Tenth System.

The simulator is based on the F1TENTH car, a $\frac{1}{10}$ size Formula One car with light detection and ranging (LiDAR), camera, and an embedded graphics processing unit. The 2D LiDAR sensor has 270-degree angle of view and one beam with 1081 steps; each step will return the distance between an obstacle and sensor, which is one-dimensional information. In total, the 1081 steps form a two-dimension plane. The camera provides video of the car's point of view to the front but with a 90-degree angle of view. We will not use the camera in this project.

The simulator aims at reducing the time and cost of testing functionalities. Racing car simulators are designed for racing environment simulation, which includes vehicle modelling, racetrack modelling, physics simulation, kinematic simulation, and perception simulation. Fig. 2 gives an illustration of software stack. We developed this after discussion with a group working on the physical car, showing that after a ROS workspace setup, the simulator will work on the top of the ROS workspace, and users can apply their algorithm to the simulator. There are **three main purposes** for using and extending a simulator.

- the user can setup the virtual world in the simulator as they want, such as editing car model, changing parameters of a car, modelling racetrack, etc;
- the user can apply an algorithm to the cars in the simulator;
- the user can collect data that is generated by the sensors or the algorithm.

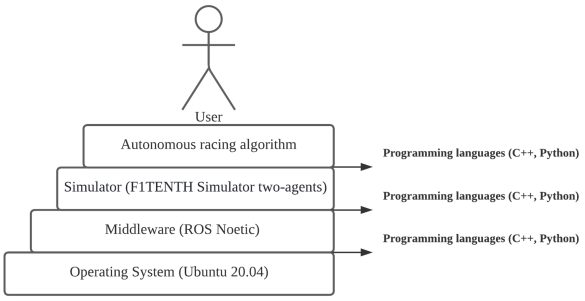


Fig. 2: Software stack for our autonomous car control.

We extend the open-source F1TENTH Simulator [1] to a multi-agents simulator. The C++ code can be found on GitHub[2] including a user guide. The implementation can be divided into four main sections:

- Adding the second vehicle to the simulator.
- Adding a box collider.
- Adding LiDAR pattern.
- Improving vehicle model.

The nodes diagram of the multi-agent F1TENTH Simulator in Fig. 3 shows the main relationships between nodes, which remains the same as the original simulator. The `simulator` node is the place where the computation of simulation is done; the `mux` node subscribes to all sources of driving commands but only publishes the selected driving commands for each car. The `keyboard` and `joystick` nodes are for publishing input information of keyboard and joystick. The `behavior_controller` node will turn the raw input of

[2]https://github.com/JZ76/f1tenth_simulator_two_agents

keyboard and joystick into driving command. The `MPC_red` and `ML_overtaking_red` nodes contain the MPC and the over-taking algorithm respectively. These nodes will generate output driving commands.

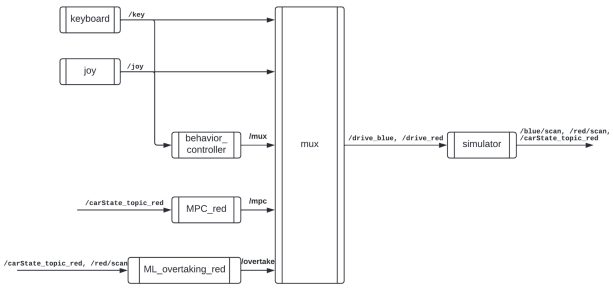


Fig. 3: Nodes diagram with critical ROS topics.

IV. Machine Learning Techniques for Over-taking

The main requirement on the over-taking algorithm is: *the algorithm can over-take an opponent with different speed, i.e., the opponent may drive at a very low speed or a very high speed*, and according to the speed of the opponent, the algorithm needs to take different manoeuvres, such as following, wheel-to-wheel, etc, in order to over-take if possible.

As mentioned in Section II, we used the end-to-end approach as a guideline. While this provides an all-in-one solution, if successful, it is less flexible and explainable than a modular approach. In particular, debugging undesirable behaviour is difficult. However, the developer can tune the hyper-parameters in the models and improve dataset quality. Since there is no freely available dataset for over-taking in the available racing environment, we create our own datasets and use imitation learning. Thus, the over-taking algorithm will try to imitate over-taking behaviour from a human driver and should be able to over-take on unseen testing racetracks.

A. Data Generation

Before starting to create a dataset, we set three principles as boundaries of the different cases to explore. Note that the leading vehicle is the car that will be overtaken, and the ego vehicle is the car that performs over-taking.

- The leading vehicle will only drive with minimum time trajectory.
- The maximum speed of the leading vehicle is always slower than the ego vehicle.
- The leading vehicle cannot defend from the ego vehicle.

Four racetracks are used to create datasets. All of them are from Formula One Grand Prix and they were chosen for varying levels of difficulty: Australian, Chinese, Bahrain, and Malaysian. For each racetrack, we set up several starting points along a racetrack, red dots in Fig. 4, usually in a straight track and before a turn. The green arrow indicates the racing direction. For each starting position, the position of the leading vehicle is always ahead of the position of the ego vehicle. We will run several times in the same beginning position with different maximum speed of the leading vehicle by changing an index in the matrix for the MPC algorithm. There are three speed tiers used for the leading vehicle, high: 6–6.5m/s, medium: 5.5–6m/s, low: 4.5–5.5m/s. The maximum speed of the ego vehicle remains at 7m/s. The leading vehicle (red car) uses the MPC algorithm whilst the ego vehicle (blue car) is manually driven.

After two cars starting at the same time, we start recording data, and we stop recording data once the ego vehicle finishes over-taking, i.e., the leading vehicle cannot be detected by the ego's LiDAR anymore. Recorded data includes LiDAR data and driving commands of the blue car, car state of the blue and red car. As a result, there are 136, 193, 67, and 192 instances created on Australian, Chinese, Bahrain, and Malaysian racetracks, respectively. Each instance has three files: "ML_dataset", "carstate_blue", and "carstate_red". The "ML_dataset" files contain LiDAR data and human driver inputs while the "carstate" files record the state of each car.

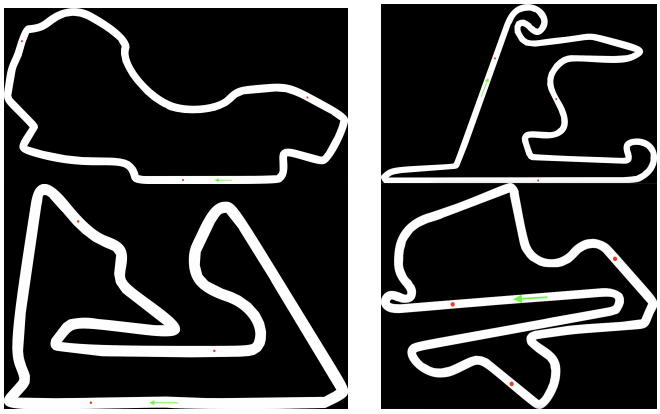


Fig. 4: Australian racetrack (top-left), Chinese racetrack (top-right), Bahrain racetrack (bottom-left), Malaysian racetrack (bottom-right). Green arrow indicates racing direction.

B. Machine Learning Techniques

a) Dense NN version: Our first NN model consists of a very basic, dense layer, which stacks multiple layers into one model. Only LiDAR data from the "ML_dataset" files is used as input. The output data is a pair of two elements: the throttle and the steering angle. The loss function uses mean square root between the output pair and the respective data in the "ML_dataset" files. A fully connected dense layer is suitable for fast training and prototype building, so that we can use it as a base and add other kinds of layers to it. There is no split of the testing data, since higher accuracy on a testing dataset does not necessarily mean better over-taking performance on a testing racetrack. Two testing racetracks are used in the evaluation of the models. We train several models with different hyper-parameters, such as adding more layers, adding more nodes in each layer, and changing the activation function. However, the behaviour of these models is poor: the model car drives in a straight line most of the time and only has a slight steering angle when reaching a curve.

The main reason for this poor performance of the fully dense model is that the model treats each frame of the LiDAR data stream independently. This misses the time-series nature of the data stream with only small changes between frames. Notably, the two important, directly controllable parameters of speed and rotation of the car are not explicit. Hence, we decided to use in the next step recurrent neural networks (RNNs) for processing LiDAR data. The rationale is that RNNs are good at processing sequential, time-series data. The frames in the data stream have relationships and contain more information in the context of the time-series. This nature of the data is akin to, e.g. natural language, video, or voice data. Moreover, the problem can be classified as a sequence-to-sequence problem, i.e. given a sequence of LiDAR data, the output should be a sequence of driving commands.

b) LSTM version: Long Short-Term Memory (LSTM) is a special kind of RNN, which can remember longer periods of data than simple RNNs [16]. The data processing for LSTMs can be divided into four operations: forget gate, input gate, conveyor belt, and output gate. The forget gate is used for deciding which old data needs to be forgotten. In the input gate, new data will be selected and added to the conveyor belt. The conveyor belt contains both old data and new data, whilst it can keep data from a long period ago. Notably, the conveyor belt structure solves the gradient vanishing problem in simple RNNs. The output gate is the place of calculating the result, and the result will be used for updating those four parameter matrices.

Considering that there are thousands of rows/frames of LiDAR data in each instance, as a starting point for our machine learning development we believe the LSTM layer is a good candidate, because it can remember a lot of LiDAR data. With this rationale, we replace the first dense layer by an LSTM layer and keep the other dense layers the same. The performance of this model is better than the previous one. The car tries to avoid collisions if it is too close to the opponent or the racetrack. However, it does not seem to know how to drive forward rather just idling around. Hence, we consider other improvements to the NN design.

c) SimpleRNN version: In the next phase, we make the following **three major improvements** on the neural network models discussed above. The structure of the improved NNs is shown in Fig. 5.

First, because the training time of the LSTM model is too long, we switch to **SimpleRNN**s. Due to the large dimensionality of the input data, 1081 data items per timestamp, if the output shape of the LSTM layer is of a similar size, there will be around one million parameters. These need to be trained in generating a model, and training time for each epoch for such a LSTM model is nearly 4 hours. In contrast, training time for the SimpleRNN layer is only about 1 hour.

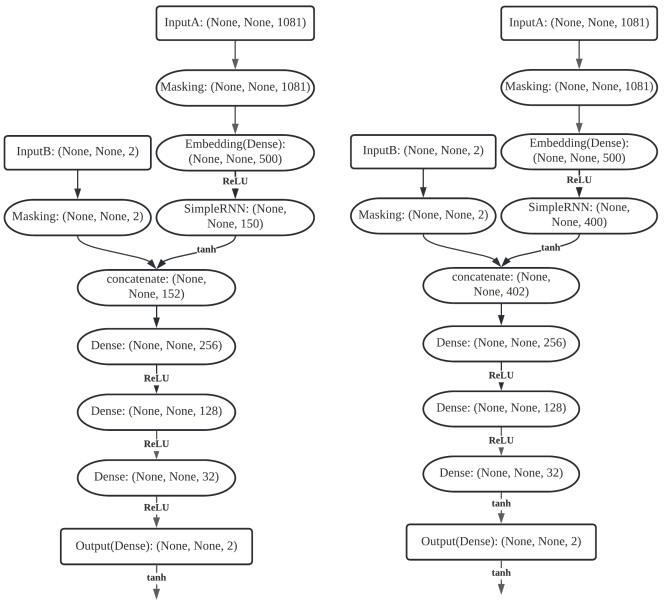
Second, in each of the NNs shown in Fig. 5 we added an **embedding layer**. This structure is borrowed from natural language processing (NLP) [17]. The embedding layer can reduce the dimensionality of the input data and can add a weight to the data. This is useful because not every column in the LiDAR data is equally important when making decisions. The weights will be learnt from the data during training. We test different sizes of embedding layers, with the following trade-offs. If the size is too large, the amount of data is insufficient to train the embedding layer well. If the size is too small, some important information may be lost after the embedding layer.

Third, we add two pieces of information as **new input**: the current speed and steering angle of the car. Both are easily obtainable from sensors in a real car as well, and therefore valid choices as input data. When the car is driving forward on a very straight racetrack, there is very little difference between two LiDAR data points at different timestamps. Thus, little variation in the input data to make a decision on.

The structure of the new models is shown in detail in Fig. 5. InputA is the same as before, which is LiDAR data, InputB is the new data, a pair of current speed and steering angle. Fig. 5 also gives different hyper-parameter settings in detail that impact over-taking performance. The main differences between them are the size of the RNNs layer and the size, number, and activation function of dense layers, shown as purple in Fig. 5. When choosing the size of each layer, we consider that the size represents the capacity of processing data and it should not change suddenly, which may cause loss of information. For the activation function, we chose the ReLU function and hyperbolic tangent function, due to the ReLU function solving the gradient vanishing problem and the hyperbolic tangent function being able to map data into a central symmetry of 0. Moreover, before feeding the input data to the models, we need to apply a padding function to the input data and add a masking layer after the input layer, so that the length of input data will be same.

We apply several versions of this NN-based AI, SimpleRNN, on the Australian track as dataset, and we find the performance of the new structure much better than previous structures. We discuss details in the following section. In total, we created 20 models with different hyper-parameters (activation function, size of RNNs layer, and number and size of dense layers) by using cross-validation.

After having used four testing datasets (Australian, Chinese, Bahrain, Malaysian) and two shuffled datasets (combining Australian+Malaysian and all four racetracks), we got five models that can over-take successfully in most cases on those four racetracks. *These models are RNN_10_3, RNN_10_4, RNN_12_3, RNN_14_3, and RNN_14_5.* The reason we used two shuffled datasets is to balance the bias of the steering angle, in each mixed dataset, since the proportion of full steering angle data on Chinese and Malaysian racetrack is greater than the proportion on Australian and Bahrain. The naming convention encodes information about the model as follows: the first part characterises the kind of NN; the second part specifies the number for a setting of the hyper-parameters; the last part is an index for the race-track (Australian, Chinese, Bahrain, Malaysia, Australian+Malaysian, all four tracks). Note that for the latter, all tracks up to this number are used as training sets. For example, RNN_12_3 and RNN_14_3 used



(a) RNN_12 (b) RNN_14

Fig. 5: Structure and hyper-parameters of models RNN_12 and RNN_14

different hyper-parameters but both models have been trained on Australian, Chinese, Bahrain, and Malaysian datasets.

A concern with more sophisticated NN designs is the increase in training time. Tab. I shows how long it takes to train a model. Compared with the corresponding stage of RNN_10 series, the LSTM_10 series has significantly longer training time of an epoch, which is 9, 8.7, 4.5, and 8.3 times longer, respectively. There are two reasons for this result, one is that the number of parameters in the LSTM layer is greater compared to the RNN layer; another reason is that training an LSTM layer requires much more memory than an RNN layer, causing memory swapping when exceeding the main memory.

TABLE I: Training time of models RNN_10 and LSTM_10

Model Name	Number of Epochs	Training Time of an Epoch (Seconds)
RNN_10	10	146
RNN_10_1	10	132
RNN_10_2	10	220
RNN_10_3	10	149
RNN_10_4	5	2042
LSTM_10	10	1314
LSTM_10_1	5	1147
LSTM_10_2	5	983
LSTM_10_3	5	1236

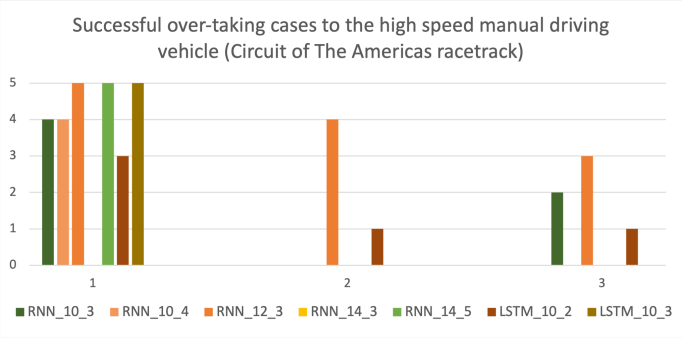
V. Results

In this section we evaluate whether our models meet the main requirement: "*the algorithm can overtake an opponent with different speed, i.e., the opponent may drive at a very low speed or a very high speed*". We use two unseen racetracks and set up various experiments to illustrate the result. Both race-tracks are from Formula One: United States and Spanish. Note that the racing direction of Circuit of The Americas is anti-clockwise where all racetracks in the training dataset are clockwise. We set several beginning positions, and set up two cars in a beginning position, one leading vehicle (blue car) and one ego vehicle (red car). The leading vehicle starts ahead of the ego vehicle. The leading vehicle is manually controlled with three tiers of maximum speed, low: 4.5m/s, moderate: 5.5m/s, and high: 6.5m/s. This also determines the difficulty of overtaking. The ego vehicle is controlled by one of our over-taking models with 6.8m/s max speed. Due to the different maximum speed of the leading vehicle, the ego vehicle will overtake at different places. For each model and track, we test three beginning positions and three different maximum speeds of the leading vehicle, repeating the same experiment 5 times, amounting to 45 experiments per model and track.

The data in Fig. 6 shows the number of successful overtaking experiments *out of five* experiments in different beginning positions and racetracks. The x axis is the beginning position (encoded as 1, 2, and 3). The y axis is the number of successful over-takes (out of 5; thus a value of 5 means perfect performance) for each of the models (colour coded). In the upper figure, we can see that most models failed at the second and third beginning position. The possible reason is that the direction of racing on The Americas track is **anti-clockwise** and there is a sharp curve where the car needs to **turn left sharply, but all sharp curves in the training datasets are right-turns**. Therefore, most models cannot follow the curve correctly and collide with the racetrack. For the third beginning position, when the maximum speed of the blue car is high, the overtaking will happen at the second curve after the third beginning position, which is another sharp left-turn curve, and most models fail again. But if the maximum speed of the blue (leading) car is low or moderate, the red (following) car will finish overtaking before the sharp curve, and the successful cases are much larger. Obviously, the limitation of imitation learning is that the training dataset needs to contain all over-taking scenarios, otherwise the model is likely to fail in difficult unseen cases. In direct comparison, the worst model is RNN_14_3 (yellow), with only 8 out of 45 success cases in over-taking, compared to the best model, RNN_12_3 (orange), with 39 out of 45 success cases in over-taking on The Americas racetrack. For the LSTM models, only LSTM_10_2 can pass at least one test per beginning position. We believe that the distribution of the training data is too biased and the LSTM layers are over-fitted to the training data.

On the lower figure, for The Spanish track, all models can perform over-taking successfully, except model RNN_14_3. Although the models never trained on this racetrack, the training datasets include all kinds of curves on this racetrack. This is the main reason why most models have a very high success rate. The result is the same as on The Americas racetrack, the best model is RNN_12_3, which achieves a perfect result of 45 out of 45 successes in over-taking. The worst model is RNN_14_3, which has only 13 out of 45 success cases. Summarising the results on both racetracks, **model RNN_12_3 has the highest number of successful**

over-taking cases, 84 out of 90 cases. We believe that the structure and size of simple RNNs (as shown in Fig. 5) and the training level of RNN_12_3 is most suitable for this task.



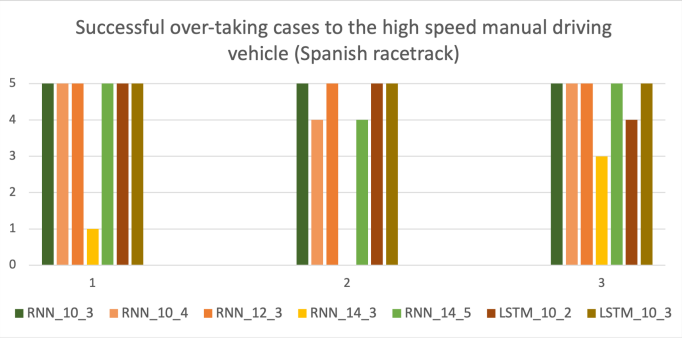


Fig. 6: Successful overtakes of each model on Circuit of The Americas and Spanish racetrack with high speed tier of the leading vehicle (manual driving).

VI. CONCLUSIONS

We have tried dense neural networks, long short-term memory (LSTM), and simple RNNs on the problem of an over-taking algorithm for cars in a simulator. We found the SimpleRNN model is the most suitable structure. **The additional input (real speed and steering angle)** proved to be crucial and fair information, since it is available to the physical car as well. Compared with a fully dense model, the SimpleRNN model has capacity to process sequential data, in the form of the LiDAR data at different timestamps being connected rather than independent. Compared with an LSTM model, **a SimpleRNN model is faster to train and less prone to overfitting**. Due to the number of parameters in LSTM being four times that of SimpleRNN, it will take longer to update all parameters and it will be more likely to tightly fit the dataset than SimpleRNN. In the SimpleRNN models, the average training time is 5 minutes for each epoch. In comparison, the LSTM models take three and half hours for each epoch. But the distribution of predicted values in our dataset is biased, thus LSTM models suffer from overfitting. Notably, the **embedding layer** helped with reducing the dimensionality of LiDAR data, which also reduced the total number of parameters in the following RNN layer and training time.

From these results we see that the hyper-parameters have a significant impact on the performance of over-taking. It is difficult to predict the quality, in over-taking behaviour, of a certain hyper-parameter setting of a model. Compared with the modular pipeline approach in Fig. 1, improving models is more like data engineering rather than control engineering, which limits its usage in many fields. From our study we can summarise that, although NN models are not necessarily the best solution for autonomous driving, they can handle specific tasks like controlling a car in a simulated environment to over-take another car.

Additionally to the above results, we also used data from a past F1TENTH head-to-head virtual competition (IROS 2020). We used the same setting as in the competition: one car is controlled by a combination of MPC and the over-taking algorithm, whilst another car is controlled manually and measuring lap time. The fastest experiment ranks in the top 2, compared with the data on the competition website. This gives an indication that, despite not competing ourselves, our simple RNN based AI should perform well compared to other control algorithms for this simulated car.

REFERENCES

[1] U. of Pennsylvania, "F1tenth simulator," 2020. [Online]. Available: https://github.com/f1tenth/f1tenth_simulator

[2] F. Chollet *et al.*, "Keras," 2015. [Online]. Available: https://github.com/keras-team/keras

[3] TensorFlow Developers, "Tensorflow," Feb. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7641790

[4] R. McAllister *et al.*, "Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning." International Joint Conferences on Artificial Intelligence, Inc., 2017.

[5] M. Bojarski *et al.*, "End to end learning for self-driving cars," *arXiv:1604.07316*, 2016.

[6] Y. Pan *et al.*, "Agile autonomous driving using end-to-end deep imitation learning," *arXiv preprint arXiv:1709.07174*, 2017.

[7] Y. Song *et al.*, "Autonomous overtaking in gran turismo sport using curriculum reinforcement learning," in *IEEE Intl Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 9403–9409.

[8] G. Williams *et al.*, "Information theoretic mpc for model-based reinforcement learning," in *IEEE Intl Conf on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1714–1721.

[9] P. R. Wurman *et al.*, "Outracing champion gran turismo drivers with deep reinforcement learning," *Nature*, vol. 602, no. 7896, pp. 223–228, 2022.

[10] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE access*, vol. 8, pp. 58 443–58 469, 2020.

[11] V. Sezer and M. Gokasan, "A novel obstacle avoidance algorithm:"follow the gap method"," *Robotics and Autonomous Systems*, vol. 60, no. 9, pp. 1123–1134, 2012.

[12] S. Chen, Y. Leng, and S. Labi, "A deep learning algorithm for simulating autonomous driving considering prior knowledge and temporal information," *Computer-Aided Civil and Infrastructure Engineering*, vol. 35, no. 4, pp. 305–321, 2020.

[13] T. Weiss and M. Behl, "Deepracing: A framework for autonomous racing," in *2020 Design, automation & test in Europe conference & exhibition (DATE)*. IEEE, 2020, pp. 1163–1168.

[14] M. Boulet, O. Guldner, M. Lin, and S. Karaman, "MIT racecar simulator," 2019. [Online]. Available: https://github.com/mit-racecar/racecar_simulator

[15] ——. (2019) MIT racecar. [Online]. Available: https://racecar.mit.edu/

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[17] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," *Advances in neural information processing systems*, vol. 13, 2000.