# Learn-to-Race: A Multimodal Control Environment for Autonomous Racing

**Supplementary Material**

## A. Racing Simulator Details

### A.1. Client and Server Information Exchange

The agents and the racing simulator act together as a client-server system. The racing simulator includes both a physics and graphics engine and provides numerous communications mechanisms for a variety of use cases. Figure 1 summarises the simulator system architecture.

#### A.1.1 Simulator State

Management of the simulator's state is done through a websocket interface, allowing for two-way communication and for clients to update the state of the simulator including the ability to:

- Change the map
- Change the type of vehicle
- Change the pose of the vehicle
- Change the input mode
- Turn on/off debugging routines
- Turn on/off sensors
- Modify sensor parameters
- Modify vehicle parameters

#### A.1.2 Simulator-to-Agent Communication

The simulator communicates to agents primarily with sensory information including:

- LiDAR data from 4 independent sensors
- RADAR data from the vehicle's radar sensor
- Images from the front-facing camera
- Pose information from the inertial measurement unit on the vehicle
- Additional data about the state of the vehicle such as brake pressure and tire speed *per wheel*
- Ground-truth information about other vehicles
- Ground-truth information about virtual objects

The camera publishes images using Transmission Control Protocol (TCP) while the others publish sensory data using User Data Protocol (UDP) or over a Controller Area Network (CAN). While the `Learn-to-Race` framework exclusively supports software-in-the-loop simulation, and therefore, only virtual CAN buses, the racing simulator also supports hardware-in-the-loop simulation and physical CAN buses.

#### A.1.3 Agent-to-Simulator Communication

Agents can communicate racing actions to the simulator in a variety of ways:

- Via a keyboard or joystick for human drivers
- UDP packets with steering, acceleration, and gear requests
- Through a safety layer with longitudinal acceleration and curvature requests
- Via various API modes which allow for more granular control of the vehicle including individual motor torques and brake pressure requests

Consistent with the simulator-to-agent above, agent-to-simulator communication can be done over virtual or physical CAN buses.

### A.2. Additional Visualisation

The racing simulator features multiple real world racetracks each with unique features that challenge human and autonomous agents alike (see Figure 2).

## B. Dataset Details

We generate a rich, multimodal dataset of expert demonstrations from the training racetracks (`Track01:Thruxton` and `Track02:Anglesey`), in order to facilitate pre-training of agents via, e.g., imitation

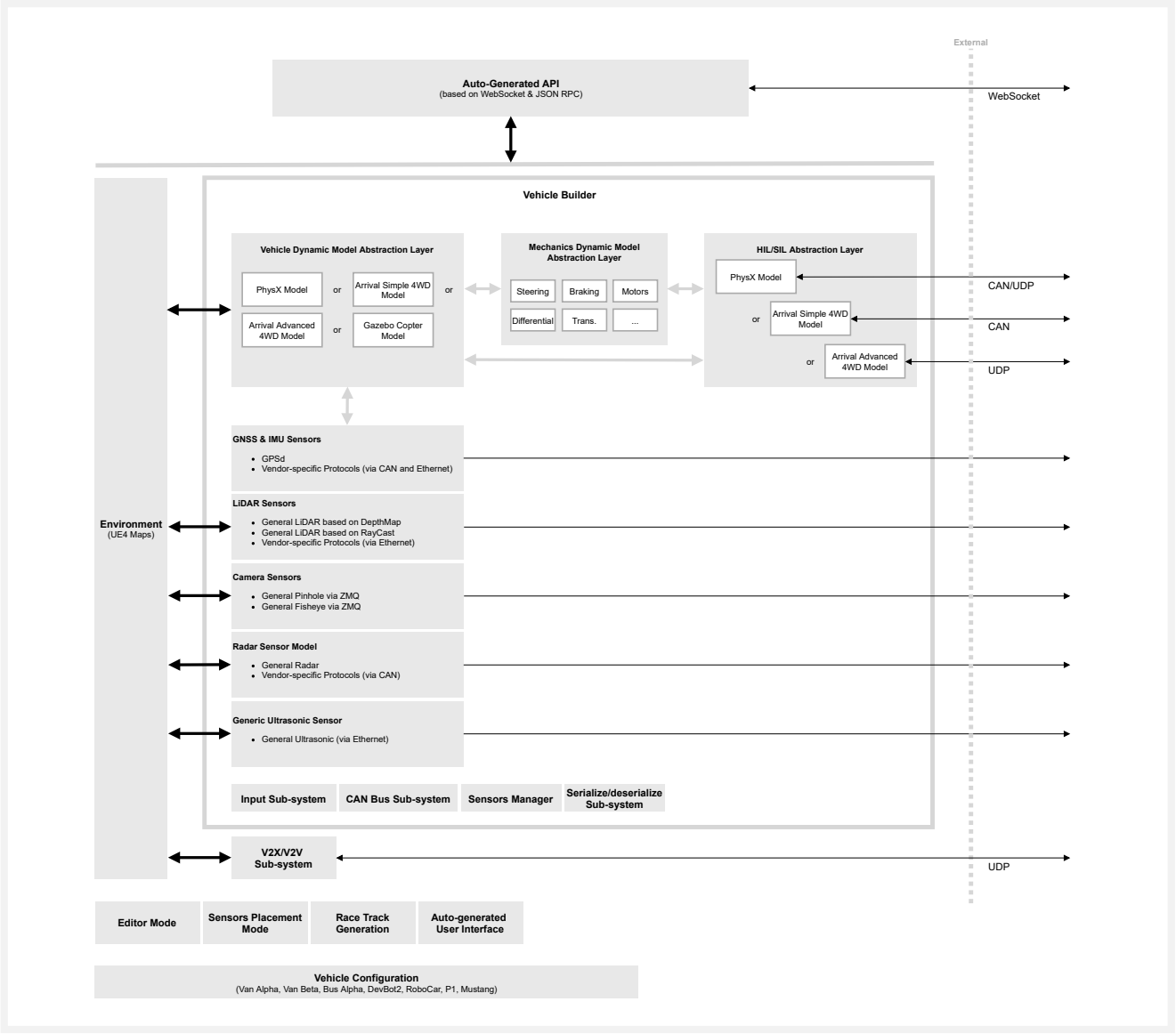


Figure 1: Overview of the racing simulator.

learning (IL). The L2R dataset contains multi-sensory input at a 100-millisecond resolution, in both the observation and action spaces. See Table 1 in the main paper for a complete list of available modalities. The expert demonstrations were collected using a model predictive controller (MPC) that tracks the centerline of the race track at a pre-specified reference speed. Important parameters for this centerline MPC expert included acceleration range of [-1, 1], steering range of [-1, 1], and image H×W dimensions of $384 \times 512$. This training dataset contains 10,600 samples of each sensory and action dimension, in this first version, which includes 9 complete laps around the track. Demonstrations were saved as invidual step-wise transitions, using `numpy.savez_compressed`[1], with the following as `dict` fields in the data: (i) `img` with shape $(384, 512, 3)$; (ii) `multimodal_data` with shape $(30, )$; (iii) and `action` with shape $(2, )$. The fields in `multimodal_data` correspond to the vector dimension mappings, indicated in Table 1.

Future version releases of L2R will include access to new simulated tracks (also modelled after real tracks, from around the world) as well as expert traces generated from

[1] `https://numpy.org/doc/stable/reference/generated/numpy.savez_compressed.html`

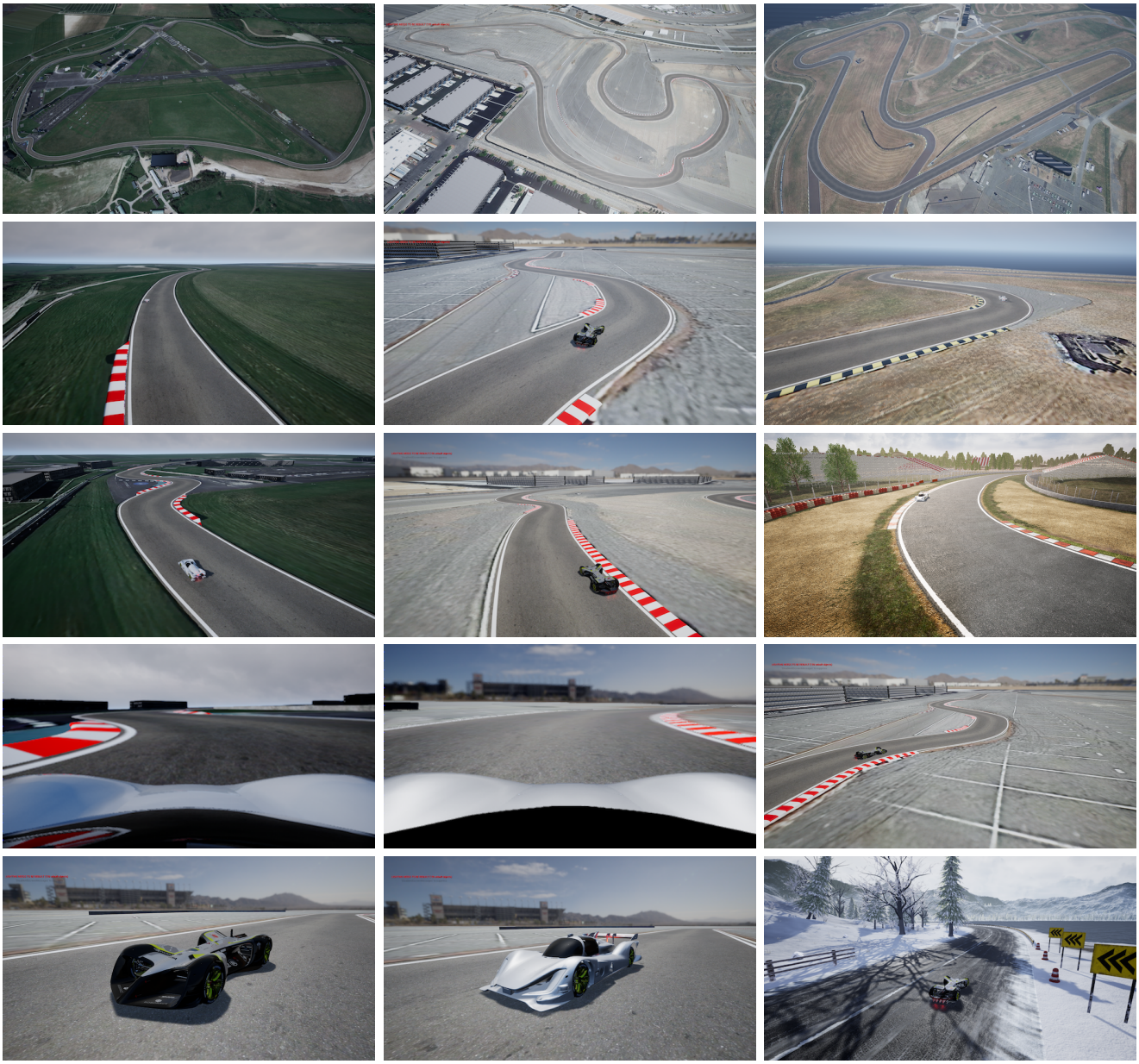

Figure 2: *First column, top four rows*: the Thruxton Circuit race track, United Kingdom, is infamous for its long straightaways, high speeds, and a difficult speed-trap near the finish line. *Second column, top four rows*: the North Road race track at Las Vegas Motor Speedway, United States, includes the sharp turns and merciless speed traps and adds a vision-processing challenge for learning agents, due to the lower contrast between the track and its surroundings. *Third column, top four rows*: the Anglesey Circuit race track, United Kingdom, features two prominent straights and several harrowing turns. *Last row*: the racing simulator features multiple car models, sensor placements, weather conditions, and additional tracks.

these additional tracks—across various weather scenarios, in challenging multi-agent settings, and within dangerous obstacle-avoidance conditions.

## C. Additional Agent Details

### C.1. RL-SAC Model Details

The `RL-SAC` agent learns from image embeddings rather than raw pixels. The encoder used is a convolutional variational autoencoder (VAE) which was trained prior to, and

Table 1: Vector dimension mappings, to which the data fields in `multimodal_data` $(30,)$ correspond.

| Array indices | Description |
|---|---|
| 0 | steering request |
| 1 | gear request |
| 2 | mode |
| 3, 4, 5 | directional velocity in $m/s$ |
| 6, 7, 8 | directional acceleration in $m/s^2$ |
| 9, 10, 11 | directional angular velocity |
| 12, 13, 14 | vehicle yaw, pitch, and roll, respectively |
| 15, 16, 17 | center of vehicle coordinates in $(y, x, z)$ |
| 18, 19, 20, 21 | wheel revolutions per minute (per wheel) |
| 22, 23, 24, 25 | wheel braking (per wheel) |
| 26, 27, 28, 29 | wheel torque (per wheel) |

Table 2: `RL-SAC` model hyperparameters

| Hyperparameter | Value |
|---|---|
| Buffer size | 100,000 |
| Gamma | 0.99 |
| Polyak | 0.995 |
| Learning rate | 0.001 |
| Alpha | 0.2 |
| Batch size | 256 |
| Start steps | 1000 |
| Learning steps | 5 |

frozen during, the `RL-SAC` agent learning. The VAE was trained to encode RGB images of with a width and height of 144 pixels each into a latent space of size 32. The encoder architecture consisted of 4 convolutional layers, each followed by a ReLu activation, with a kernel size and stride of 4 and 2, respectively. The result of the convolutions was passed through a single fully connected layer to the compressed representation. Binary cross entropy loss and an Adam optimiser were used for training.

The `RL-SAC` agent was trained for 1,000 episodes which was approximately 1 million steps in the environment. We trained this agent in vision-only mode, so it only had access to the camera's images. The agent passed the encoded images through two fully connected layers with 64 units each and a final layer with an output shape of 2, matching the environment's action space. Gradient updates were taken at the conclusion of episodes, and the training hyperparameters are listed in Table 2.

### C.2. MPC Agent Details

The MPC problem is summarised by Equation 1. The objective (Equation 1a) is to minimise the tracking error with respect to a reference trajectory, in this case the centerline of the race track at a pre-specified reference speed, with regularisation on actuations, over a planning horizon of T time steps. $\mathbf{Q}$ and $\mathbf{R}$ are both diagonal matrices corresponding to cost weights for tracking reference states and regularising actions. At the same time, the MPC respects the system dynamics of the vehicle (Equation 1b), and allowable action range (Equation 1c).

$$\min_{\mathbf{a}_{1:T}} \quad \sum_{t=1}^{T} \Big[ (\mathbf{s}_t - \mathbf{s}_{ref,i})^T \mathbf{Q} (\mathbf{s}_i - \mathbf{s}_{ref,i}) + \mathbf{a}_i^T \mathbf{R} \mathbf{a}_i \Big] \tag{1a}$$

$$\text{s.t.} \quad \mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t), \quad \forall t = 1, \ldots, T \tag{1b}$$

$$\underline{\mathbf{a}} \leq \mathbf{a}_t \leq \bar{\mathbf{a}} \tag{1c}$$

Specifically, we characterise the vehicle with the kinematic bike model[2] [2] given in Equation 2, where the state is $\mathbf{s} = [x, y, v, \phi]$, and the action is $\mathbf{a} = [a, \delta]$. $x, y$ are the vehicle location in local east, north, up (ENU) coordinates, $v$ is the vehicle speed, and $\phi$ is the yaw angle (measured anti-clockwise from the local east-axis). $a$ is the acceleration, and $\delta$ is the steering angle at the front axle.

$$\dot{x} = v \cos(\phi) \tag{2a}$$

$$\dot{y} = v \sin(\phi) \tag{2b}$$

$$\dot{v} = a \tag{2c}$$

$$\dot{\phi} = v \tan \delta / L \tag{2d}$$

A key challenge is that the ground truth vehicle parameters were not known to us. Aside from $L$ defined as the distance between the front and rear axle, the kinematic bike model expects actions, i.e. acceleration and steering, in physical units, while the environment expects commands in $[-1, 1]$. The mapping is unknown to us, and non-linear based on our observations. For instance, acceleration command = 1 results in smaller acceleration at higher speed. In the current implementation, we make a simplifying assumption that $a = k_1 \times$ acceleration command, and $\delta = k_2 \times$ steering command.

We use the iterative linear quadratic regulator (iLQR) proposed in [3], which iteratively linearizes the non-linear dynamics (Equation 2) along the current estimate of trajectory, solves a linear quadratic regulator problem based on the linearized dynamics, and repeats the process until convergence. Specifically, we used the implementation for iLQR from [1]. The parameters used by the MPC are summarised in Table 3.

## D. Metric Equations

We quantify the parametric curvature of a trajectory in Eqn. 3, with $x'_t$ denoting $\frac{dx}{dt}$ at time $t$, and we summarise the curvature of the entire path as $\kappa_{rms}$ in Eqn. 4:

[2]This set of equations is defined with respect to the back axle of the vehicle and is used for generating expert demonstrations. The kinematic bike model defined with respect to the centre of the vehicle is also included in our code base.

Table 3: MPC parameters

| Parameter | Value |
|---|---|
| $\mathbf{Q}$ | diag($[1, 1, 1, 16]$) |
| $\mathbf{R}$ | diag($[0.1, 1]$) |
| $v_{ref}$ | 12.5 m/s |
| $\bar{\mathbf{a}}$ | $[1, 0.2]$ |
| $\underline{\mathbf{a}}$ | $[-1, -0.2]$ |
| $L$ | 2.7 m |
| $k_1$ | 10 |
| $k_2$ | 6 |
| $T$ | 6 |

$$\kappa_t = \frac{x'_t y''_t - y'_t x''_t}{\left((x'_t)^2 + (y'_t)^2\right)^{\frac{3}{2}}} \tag{3}$$

$$\kappa_{rms} = \sqrt{\frac{1}{T}\left(\sum_{t=0}^{T} \kappa_t^2\right)} \tag{4}$$

We measure *Trajectory efficiency* as $\rho$ in Eqn. 5 based on the curvature, $\kappa_{rms}$, of the race track and the racing agent's trajectory.

$$\rho = \frac{\kappa_{rms,\, racetrack}}{\kappa_{rms,\, trajectory}} \tag{5}$$

## References


[1] Brandon Amos, Ivan Dario Jimenez Rodriguez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable mpc for end-to-end planning and control. *arXiv preprint arXiv:1810.13400*, 2018.

[2] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1099. IEEE, 2015.

[3] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, pages 222–229. Citeseer, 2004.