Personalization of Assitive Driving Tasks in CARLA

Prajankya Sonar Sanjeev Kannan Sapan Agrawal Zhuoyun Zhong

***Abstract*— Our goal is to develop a driver-in-the-loop framework to enable studying and testing of the personalized assisted driving systems. The project involves development of integrated hardware and software to enable the interactive user study. Here, we focus on personalization primarily for three tasks : Adaptive Cruise Control, Lane Keeping and Lane Change. We base our work on the latest state-of-the-art methods.**

***Index Terms*—ADAS, personalization**

I. Introduction

Advanced driver-assistance systems (ADAS) are being deployed in modern vehicles to automate, adapt and enhance vehicle systems for a safer and better driving experience. Safety features are designed to avoid collisions and accidents either by alerting the driver (Passive ADAS) or by taking the control over the vehicle (Active ADAS). These adaptive features may provide adaptive cruise control (ACC), lane departure warning, forward collision avoidance, Lane keeping assist and other forms of driving assistance. However, the generic ADAS system might not be suitable for all drivers. Hence, personalization of ADAS is being adopted to not only improve safety but also to adapt the technology to the user's driving style. For example, some drivers value efficiency and time, and feel comfortable being closer to other vehicles. On the other hand, some prefer defensive driving style where they value safety and maintain larger distances from the leading vehicle (as depicted in Fig. 1). In this work, we personalize an assistive driving system that can learn and adapt to the driver's style and meet his/her needs.

We have looked at research on personalizing certain driving tasks and have implemented a slightly modified version of the algorithms on the CARLA simulator and our hardware setup. We have modified the existing algorithms so that we can add a layer of personalization. Through user study, we have invited multiple users to perform different driving tasks. From their driving data, our model is successfully able to learn different parameters and personalize a driving system to perform tasks in their respective driving styles.

All authors are from Worcester Polytechnic Institute, (prajankya, skannan, ssagrawal, zzhong3)@wpi.edu

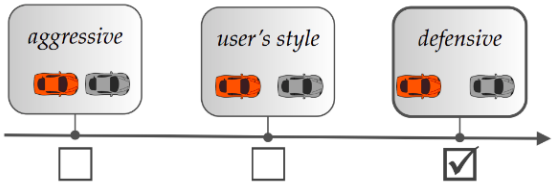

Fig. 1: Different driving behaviors in distance keeping [1]

As our contribution, we developed a driver-in-the-loop framework for studying and improving driver assisted systems. The project involved development of software (assisted driving and personalization) to simulate the driving tasks in CARLA simulator and the hardware to enable the interactive user study. To validate our framework, we performed user study experiments for different assisted driving tasks (ACC, lane keeping and lane changing) while learning and modifying the ADAS system to mimic the user driving behaviour. Following are the different use cases of our framework:

- Students/researchers can implement and test advance motion planning or control algorithms independently as we provided the basic functionalities in each module.
- The driver-in-loop framework can be used to perform user studies not only limited to personalization but also for testing shared autonomy, learning for data tasks, human sensing and shared perception-control [2].

The report is organised as follows: The next section, Sec II outlines the related work in the field of personalization. In Sec. III, we explicitly define our project goals and proposed method. Later in Sec IV, we discuss the implementation details of the project at levels of Planning, Controls, Personalization and Software-Hardware

Integration. In Section V, we discuss the user study and experimentation and results of personalization.

II. Related Work

Personalization in assisted driving systems is a research area that started in the late 90's and has gained traction in recent years due to progress in vehicle autonomy. Personalization methods can be broadly be categorized into two groups : explicit and implicit. The explicit methods involve presenting a fixed number of setting options for the vehicle owner/driver to choose. On the other hand, implicit methods try to observe the driver patterns over a period of time and aim to match that driving style. Explicit methods offer more direct control, but restrict the driver to a small number of settings. Implicit methods allow precise and fine tuned control. Our work focuses on coming up and testing implicit personalization methods.

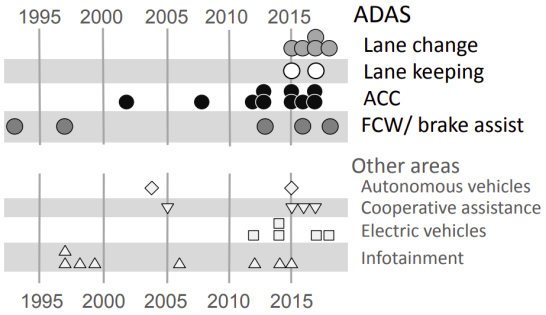

Fig. 2: Timeline : personalization research in the automobile sector [3]

Figure 2 shows a timeline for research done on assisted driving tasks. Our project focuses on personalization for three tasks - Adaptive Cruise Control(ACC), Lane Following and Lane Change. As can be seen in the figure, ACC has been researched since the early 2000s, with lane change being relatively new research topic.

Approaches for ACC are broadly broken down into group based and individual based. In the former, drivers are assigned to one of few representative driving styles for which an ACC control strategy is implemented. In the latter, the ACC strategy tries to best reproduce the driving style of an individual driver. Rosenfeld et al. [3], [4], present a group-based approach to the predict the driver's preferred ACC gap setting and when they tend to engage and disengage ACC. A notable individual driver approach includes the work done by Bifulco et al. [5], where ACC is adapted in real-time to individual drivers based on observation of their driving style. The controller framework is based on a linear model using a recursive least squares filter to reproduce the time gaps. This approach contains two modes: a "learning mode", in which the current driving style is observed and the corresponding parameters of the car following model are learned, and a "running mode" where the newly learned car following model is deployed. A similar approach used by Ramyar et al. [6] extended the modes to path following mode, car following mode, and lane change mode.

Our method of personalizing vehicle following is based on a method described by Wang et al. [7]. The method defined time headway THW and time to collide TTC as the parameters to personalize. The collected data showed that during a vehicle following task, the inverse of TTC, TTCi, is distributed around 0, leaving THW the only key factor of a driver's behavior. The prediction of a driver's preffered THW, however, was simpler computed by using mean. In our user study, a better modeling method was used and described in later section.

To decide whether the driver is going to perform a lane change, [8] develop a system that automatically detects the possibility of a lane change, and implements the maneuver without requiring driver's explicit initiation using the turn signal lever. The lane change decision is made by a support vector machine (SVM) based classifier. However, the decision making for a lane change is not the consideration of this project. Our foucs is on the personalized lane change trajectroy, hence we assume that the driver is driving on a mandatory lane change (MLC) condition.

For automating a mandatory lane change, five factors must be looked at - ego vehicle's velocity, lateral distance of lane change, time to finish a lane change, distance of leading vehicle in destination lane and distance of trailing vehicle in destination lane [9]. In their work, the collected data was separated into training set and test set, and a high-dimensional Gaussian Mixture Model (GMM) method is used to train a driver's model. This showed a low mean difference when predicting with test set. However, the model only focused on the lateral movement of the vehicle without personalizing the longitudinal performance of the driver. In our implementation, longitudinal performance was also considered.

III. Proposed Method

The development of the project is divided into three parts - an Autonomous driving system(software), a driver personalization model and hardware to interact with the driver.

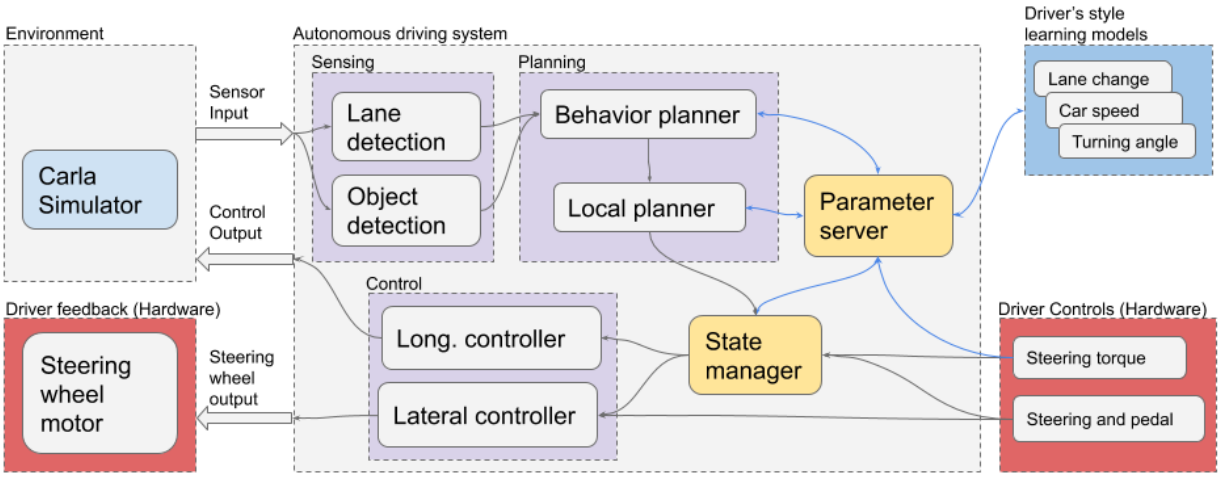

Fig. 3: Proposed architecture

The Autonomous driving system architecture is very similar to most commonly used architectures in autonomous vehicles, consisting of three parts- sensing, planning, control. As shown in Fig. 3, sensing module comprises lane and object detection. Planning consists of Behaviour and Local planning. These planners will generate waypoints which would be used by Lateral and Longitudinal controllers to control the vehicle's pedal and steering. The pedal and steering output is then used by the simulator to control the vehicle in simulation. The steering output from lateral controller is also used to give torque feedback on the steering to the driver, which can be used as assist driving or as autonomous driving.

Two communication modules namely Parameter server and State manager manages all the communication and states between all these modules, and also gives the parameters to Driver's style learning models. These models try to understand the driver's behaviour and try to update driving parameter to better adapt to the drivers' styles.

There is a torsional resistive sensor on steering wheel to detect when the user is controlling the steering wheel. This can be used as a switch to give back the control to user, or detect the amount of resistance by the driver to control how much assistance torque the system will give on the wheel.

Our end goal is to use all the above modules in a personalization module. For this, we will be inviting different users to perform three tasks on the hardware setup - lane following, vehicle following and lane lane change. After multiple iterations of each task by each user, key parameters are identified that can be used to simulate the user's driving style and personalize an assisted driving system. Our model will be evaluated by the accuracy in identifying key parameters like preferred target velocity, time taken for a lane change to ultimately derive a model that operates similar to the driver's characteristic driving style.

IV. Implementation

As shown in Fig. 4, Carla simulator is connected to Autonomous driving system using Sensor input and controller output. The developed Autonomous driving system (ADS) consists of three parts, sensing, planning and control.

Sensing is done with 2 sensors, Camera and Radar. These inputs are given to Finite state machine in planning. This is used for local planning using waypoints. Control section contains a Longitudinal controller and MPC, Stanley based lateral controllers. All this information is used for learning the Driver styles by various sub models.

Hardware is used in learning mode where it gives input to carla. In performing mode, controllers control the ego vehicle in simulation. The Steering wheel is attached to a motor which is used to turn the wheel. A feedback in the form of rotation angle is published out which is used by the developed motor driver. This driver communicates with the motor in SimpleMotion Protocol and is connected to run in near real-time loop. The loop is implemented in a multi-threaded fashion and hence uses atomic locks and mutexes extensively to share memory between the threads. A front API for the motor driver is developed in python to easily communicate with

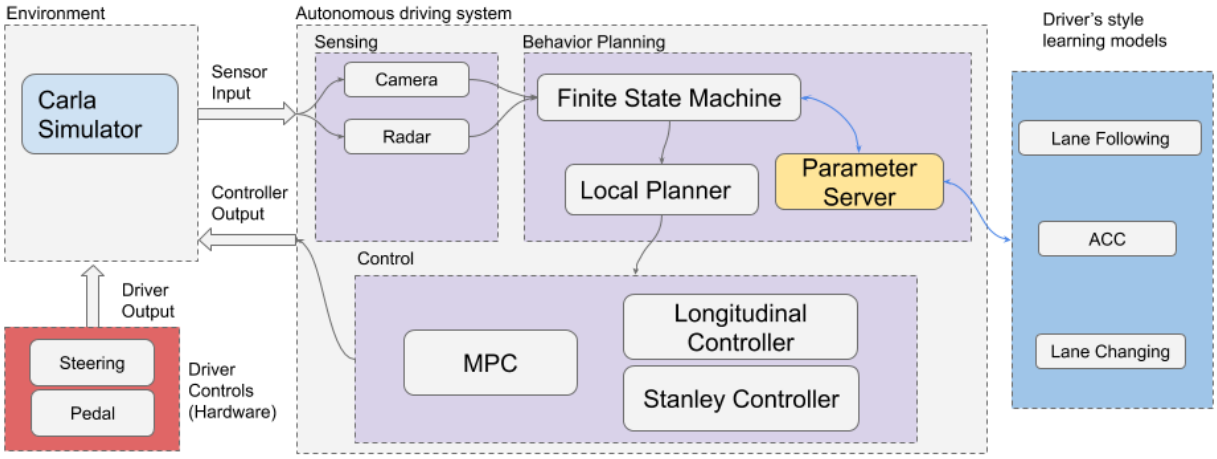

Fig. 4: Implemented architecture

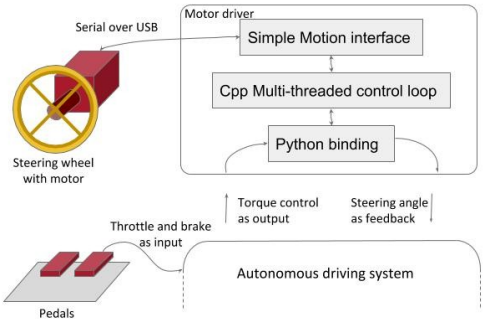

Fig. 5: Hardware implementation

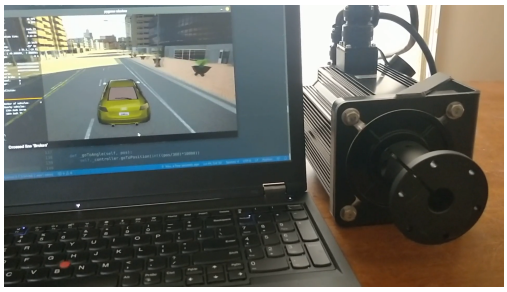

Fig. 6: Hardware testing

CARLA. Hence an internal python to C++ binding is used to communicate.

This python binding will emit callbacks on event for encoder feedback and other events. This is used by the Autonomous driving system(ADS) as steering angle input. Output from the Autonomous driving system as Torque control is given to the python binding to run on motor.

Throttle and brake pedals are externally connected as joystick interface to the Autonomous driving system.

Fig. 6 shows the hardware developed for the project. The motor has a flange which is used to connect a steering wheel.

A. *Planning*

The Planning stack for the simulator is broadly divided into two modules- global planning and local planning modules. The global planner searches for a an optimal path in a graph that represents the environment. The output of the global planner is a list of waypoints, and this list is passed to the local planner. The local planner further discretizes this path and identifies an optimal path between the waypoints present in the path. The local planner takes care of operations like lane change and obstacle avoidance. The output of the local planner is also a list of waypoints, but the size of this list is much bigger than the output of the global planner. The output from the local planner is then passed to the controller. The process is explained in greater detail below :

Global Planner

Environment Map : Our environment is chosen from one of the available town maps provided by the CARLA simulator. Each town is represented in the form of directed graph to accurately model one-way roads. An example of a town map is shown in Fig 7. This directed graph is modeled using the Python *netwrokx* library. Another parameter we can control is the hop resolution. The Hop resolution allows us to customize spacing

between nodes and hence dictate number of nodes in the graph. Since precise control is desired, we set the hop resolution at 0.1 metres.

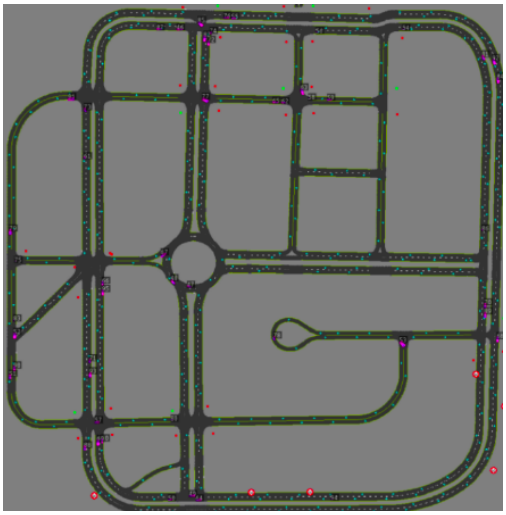

Fig. 7: Town Map in the CARLA simulator

Path Search : Once we have a graph, we can specify start and destination locations. Exact coordinates of the locations are not required since CARLA simulator has inbuilt functions that can identify nearest valid nodes that lie on the road. Once the nodes nearest to the start and destination locations are identified, the optimal path between the two graph nodes is identified using the A* search algorithm. The nodes' coordinates(waypoints) are stored in a list and passed to the local planner.

Local Planner

Once we have the path from the global planner, the local planner further discretizes it. The local planner is responsible for fine control and generates waypoints between nodes given by the global planner. The discretization is based on a parameter called sampling radius. The sampling radius is defined as the distance travelled by the car travelling at a set target velocity in one time step. The time step used here is 0.05 seconds and the target velocity typically lies between 20 to 60 km/hr.

Waypoints are generated based on the sampling radius are then stored in a double ended queue - Waypoint Queue. The size of the Waypoint Queue is typically very large(20,000 in our case) so that it can store several waypoints. At each time step, its' size is checked. If the size is less than 10,000, then it is further populated with a few hundred waypoints (100-200 in our case).

Waypoints are transfered from the *Waypoint Queue* to another double ended queue - *Waypoint Buffer*. The size of the waypoint buffer is usually very small (14 in our case). At each time step, we delete the waypoints the car has crossed and check the Waypoint Buffer size. If the Waypoint Buffer Size is not equal to its capacity, waypoints are added from the Waypoint Queue till the Waypoint Buffer is full. The transferred waypoints are then deleted from the Waypoint Queue. The small size of the Waypoint Buffer allows an iteration across its length at every time step. The waypoints in the Waypoint Buffer are then passed to the Controller at every time step. The entire flowchart of the planning module is shown in Fig 8.

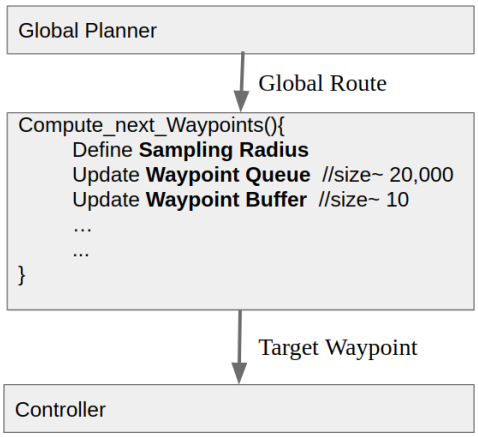

Fig. 8: Flowchart showing inputs and outputs of the planning module

B. Controllers

To navigate through the waypoints generated by the local planner, we tested three different controllers in CARLA. The vehicle can be controlled using the steering, throttle and brake inputs to the CARLA. The details of each controller has been mentioned in this subsection.

1) PID Controller: We use two separate PID controllers for minimizing the errors in the each direction of motion viz. Lateral and Longitudinal. The Longitudinal PID controller maintains the reference velocity of the car. The longitudinal controller can be defined as,

$$\begin{aligned} e_{long}(t) &= v - v_{ref} \\ throttle(t) &= K_p e_{long}(t) + K_d \dot{e}_{long}(t) \\ &\quad + K_i \int e_{long}(t) dt \end{aligned} \tag{1}$$

A simple lateral control strategy is to minimize the heading error. As shown in Fig. 10, $d\theta$ represents the

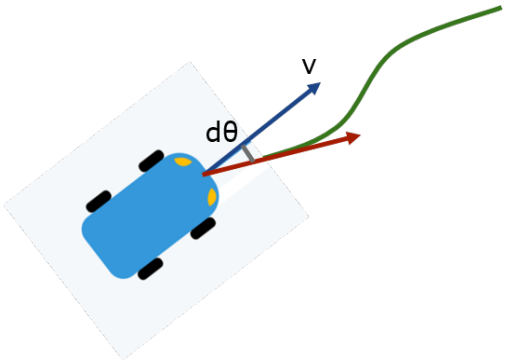

Fig. 9: PID Controller

heading error as the difference between the heading angle of the car and the tangent to the waypoints trajectory. Thus, the lateral controller ca nbe formulated as,

$$\begin{aligned} e_{lat}(t) &= \theta - \theta_{des} \\ steering(t) &= K_p e_{lat}(t) + K_d \dot{e}_{lat}(t) \\ &+ K_i \int e_{lat}(t) dt \end{aligned} \tag{2}$$

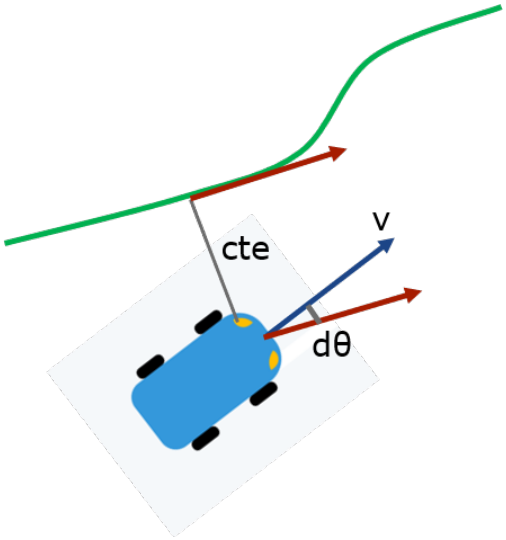

Fig. 10: Stanley Controller

2) Stanley Controller: For given set of kinematic equations, steering controller is designed such that the resulting differential equation has a globally asymptotically stable equilibrium at zero crosstrack error [10]. Beside the heading error, this controller also minimizes the cross track error (cte) defined as the distance from the vehicle normal to the trajectory along the lateral direction. The steering controller can be defined as,

$$steering(t) = d\theta + \arctan(\frac{K_{cte} * cte}{v}) \tag{3}$$

The longitudinal controller remains the same as shown in the Equation (1).

C. Model Predictive Controller(MPC)

MPC is a more advance and widely used control strategy for autonomous vehicles. It is a model-based receding horizon control which minimizes the cost satisfying a set of constraints. Additionally, due to its predictive nature (outputs state and control trajectory for N time-steps in future), it can take into account the latency or the time lag between the control command generation and control execution (practical value of 100 ms).

At each period, we read from the sensors to determine the current state of the vehicle including:

$$X = [x, y, \psi, v, \delta, a] \tag{4}$$

where, (x, y) location of the vehicle (x along the longitudinal direction and y in lateral direction), v is speed, ψ is heading angle, δ is the steering angle and a is the the acceleration.

For our purpose, we use a simple kinematic bicycle model to describe the vehicle's motion as shown in the 11.

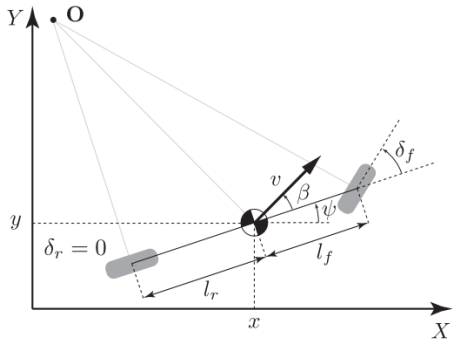

Fig. 11: Bicycle Model

where, l_f is the distance from the center of the mass to the front axle. β is the angle of v with respect to the car axis. In our example, we assume this is zero. (i.e. the car is not sliding). Thus, the kinematic model is formulated as,

$$
\begin{aligned}
x_{t+1} &= x_t + v_t cos(\psi_t) * dt \\
y_{t+1} &= y_t + v_t sin(\psi_t) * dt \\
\psi_{t+1} &= \psi_t + \frac{v_t}{L_f}\delta_t * dt \\
v_{t+1} &= v_t + a * dt \\
cte_{t+1} &= f(x_t) - y_t + v_t sin(e\psi) * dt \\
e\psi_{t+1} &= \psi_t - \psi_t^{des} + \frac{v_t}{L_f}\delta_t * dt
\end{aligned} \tag{5}
$$

where, $f(x)$ is the cubic fitted trajectory for the waypoints and ψ_{des} being defined as,

$$
\begin{aligned}
f(x) &= a_3x^3 + a_2x^2 + a_1x + a_0 \\
\psi_{des} &= \arctan\left(\frac{df(x)}{dx}\right)
\end{aligned} \tag{6}
$$

In MPC, we define a cost function to optimize our path with the trajectory. For speed, we penalize the model if the car cannot maintain at a target speed. We want no acceleration and zero steering if possible. But since it is unavoidable, we want the rate change to be as low as possible if it happens.

$$
\begin{aligned}
J = \sum_{t=1}^{N} w_{cte}||cte||^2 + w_{e\psi}||e\psi||^2 + w_v||v - v_{ref}||^2 \\
+ \sum_{t=1}^{N-1} w_\delta||\delta||^2 + w_a||a||^2 \\
+ \sum_{t=2}^{N} w_{rate_\delta}||\delta_t - \delta_{t-1}||^2 + w_{rate_a}||a_t - a_{t-1}||^2
\end{aligned} \tag{7}
$$

Beside the system model as one of the constraints, we define additional constraints on the control limits as,

$$
\begin{aligned}
\delta &\in [-25^o, 25^o] \\
a &\in [-1, 1]
\end{aligned} \tag{8}
$$

Finally, we run the optimization using Equations (4-8) using the CasADi [11]. Following is the result of the test run for the race track environment in CARLA.

V. Personalization

The personalization module collects driver's key behaviors in the following three scenes: lane following, vehicle following and lane changing. Different parameters are used in these three scenes to personalize the model.

Gaussian Mixture Model (GMM) is a very helpful tool to learn from the driver's data and is used in all of the three personalization tasks. GMM is defined as,

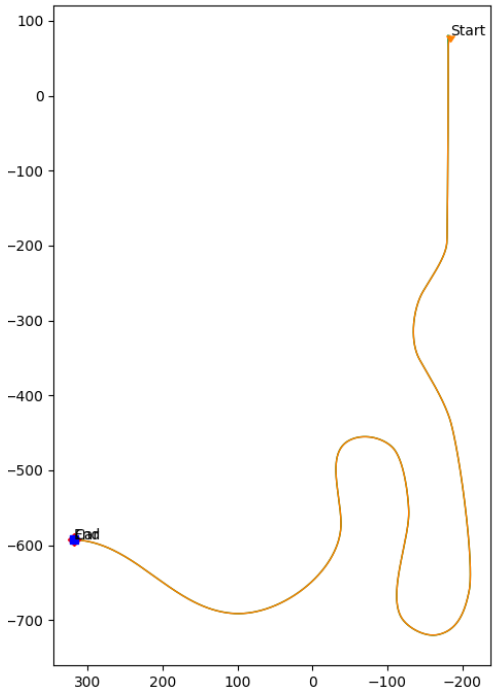

Fig. 12: MPC, the orange trajectory followed by the vehicle overlaps perfectly with the green waypoints trajectory.

$$
p(x) = \sum_{i=1}^{K} \phi_i N(\mu_i, \Sigma_i) \tag{9}
$$

where the i^{th} component is a Gaussian distribution with weights ϕ_i, means μ_i and covariances Σ_i. One example of a GMM with 5 components fitting a set of 2D data is shown in Fig. 13. Besides fitting and clustering data, the GMM can also perform regression to predict the missing values. The red points in the figure are the most likely y values given a set of x values.

The experiment is separated into two phases, the learning phase and the performing phase. In learning phase, the driver is driving, and the model will collect the data and learned from them. In performing phase, the autopilot will take the control of the vehicle.

A. Lane Following

Our first task is to personalize the lane following, which is defined as driving on a certain road with a speed limit and without any vehicle in front as shown in Fig. 14.

We followed some principles while collecting data for lane following task. We assumed the driver had a desired

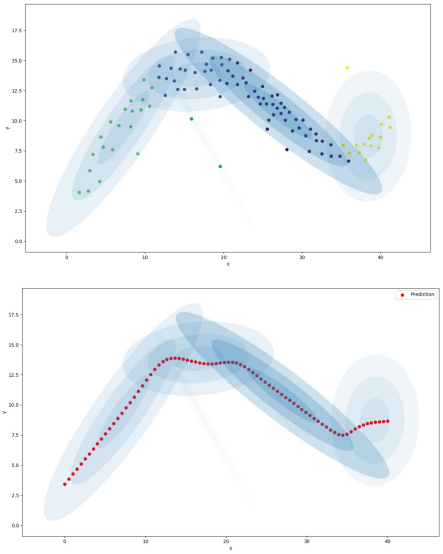

Fig. 13: Gaussian Mixture Model fitting and predicting

Fig. 14: Lane Following Scene

speed range. The data used for training was the one with stable driving speed. The vehicle would be running on only one test road and 10 sets of data would collected for each driver.

The personalization parameters of this task is the target speed. As shown in Fig. 15, the first set of data of driver 1 was plotted. The first half of the image is the histogram of stable speed. The y axis is the density of probability and the x axis is the collected stable speed. GMM with 3 components were used to fit the data. In this case, the whole GMM was not used to predict the mean of stable speed, but was used to cluster the data. The second half of the figure gives a plot of velocity over time. The straight lines give the estimated main of each Gaussian component. The red Gaussian has the highest weight, so we believe that is a good fitting of the driver's target speed.

All 10 sets of data of 3 drivers are plotted in Fig. 16.

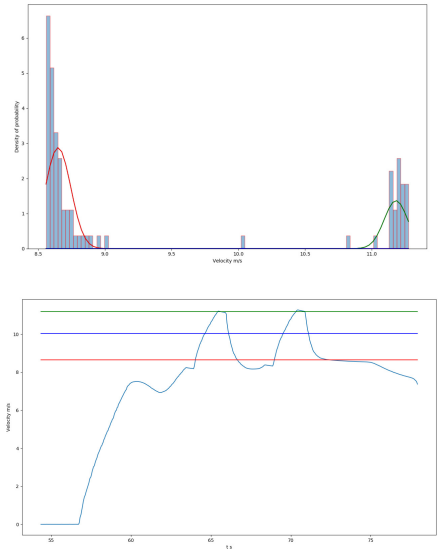

Fig. 15: Driver 1 Set 1 Target Speed Training Data

Driver	Learned Result	Desired Speed Range
Driver1	40.02 km/h	35-40 km/h
Driver2	24.91 km/h	22-25 km/h
Driver2	58.96 km/h	About 55 km/h

TABLE I: Target Speed Result

One could clearly observe that driver 3 is more defensive, the driver 2 is more aggressive and the driver 1 is more an average style. After the experiment, we asked the drivers for their desired speed ranges while performing the driving task, and we computed the result velocity of the training set. The comparison among them is shown in TABLE I. The result shows that the predicted target speed are within the desired range speed, the model could therefore predict the target speed in this case.

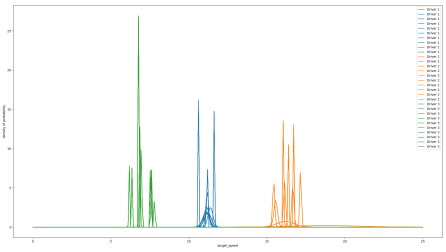

Fig. 16: Target Speed Training Data of All 3 Drivers

B. Vehicle Following

The second task is to personalize the vehicle following performance, a typical scene of Adaptive Cruise Control (ACC), which is defined as driving on a certain road with vehicles in front as shown in Fig. 17.

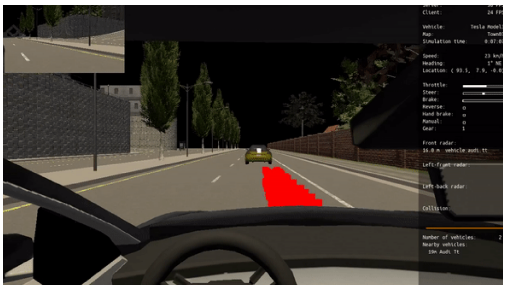

Fig. 17: Vehicle Following Scene

The principles we follow during collecting data in this case was that we only collect data when the relative speed to the leading vehicle is close to 0 [7]. The idea of doing this will be explained later. Same as before, the vehicle would be running on only one test road and 10 sets of data would collected for each driver.

The personalization parameter used in this case is called time headway, which is defined as

$$THW = D/v \tag{10}$$

where D is the distance to the leading vehicle and v is self velocity. Another very important parameter is the inverse time to collide (TTC) and its inverse TTC_i, which are defiend as

$$\begin{aligned} TTC &= D/v_r \\ TTC_i &= v_r/D \end{aligned} \tag{11}$$

With the desired TTC_i to be zero, the personalization parameter of this task is THW. As shown in Fig. 18, the right upper plot is TTC_i histogram. The left upper plot is TTW histogram, which is our desired parameters. The lower side of the figure shows the data during vehicle following process. The missing part is considered to be the vehicle approaching state, which is therefore removed from the training.

We inherited the idea of the related work that TTC_i would be distributed around 0 and the fact was shown in our data as well. In the task of computing the driver's desired THW, however, we believed calculating the time using simple mean would be slightly larger than the driver's intention, as demonstrated by one of our data set. Therefore, we, same as the last task, used GMM

Driver	Learned Result of mean	Learned Result of GMM
Driver1	3.64 s	3.20 s
Driver2	3.12 s	2.88 s
Driver2	3.23 s	2.94 s

TABLE II: THW Result

to fit the data. In this case, we could either assume the Gaussian component with highest weight, the red one, predicts the best parameter, or the general GMM predicts the best.

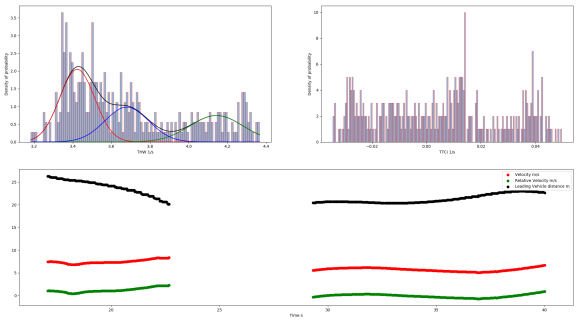

Fig. 18: Driver 1 Set 1 Safe Distance Training Data

All 10 sets of data of 3 drivers are plotted in Fig. 16. In this task, the performance of all the 3 drivers are very similar. The computed result of THW is shown in TABLE II. The results using the idea of the original work, were slightly larger.

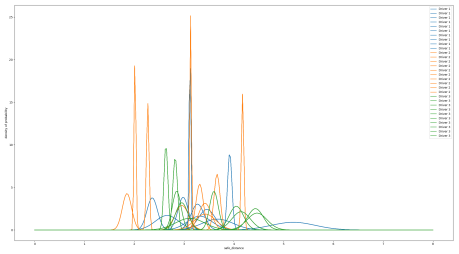

Fig. 19: Safe Distance Training Data of All 3 Drivers

C. Lane Changing

The final task is to personalize the lane changing behavior. The lane changing task is much more complicated than the previous two, as a lot more traffic information is considered during lane changing. In this task, we tried to personalize the behavior of lane changing when merging between two vehicles. The test scene used in

this case included four other vehicles, two on the current lane and two on the desired lane as shown in Fig. 20.

Fig. 20: Scene setup for lane changing

We only collected data when the vehicle left the current lane and ended up in the desired lane. The vehicle would be running on only one test road with 4 vehicles surrounding. 15 sets of data would be collected for each driver.

The lane change model used in this case is a two-layer model, consisting GMM and sinusoidal trajectory module [9]. The model flow is shown in Fig. 21. The GMM model predicts a time t_{lat} to complete a lane changing task based on four critical parameters, V, the self vehicle velocity, H, the lateral distance, D_{XL_D}, the longitudinal distance from the leading vehicle on desired lane and D_{XF_D}, the longitudinal distance from the lagging vehicle on desired lane. A sinusoidal model can then take the predicted t_{lat} and generate a trajectory fot the lane changing task.

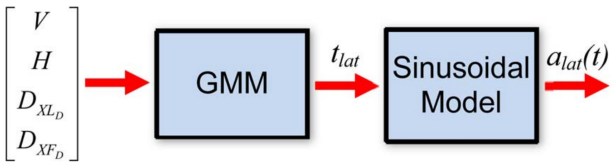

Fig. 21: Lane Change Model

This model, as described before, lacks the personalization of the driver's performance in longitudinal direction. Additionally, we collected driver's longitudinal speed and performed a Gaussian distribution fit to estimate the driver's preferred lane change speed, assuming that in longitudinal direction is constant.

In order to use GMM to predict t_{lat} and v_{lon}, a large amount of samples will be required to train and test the GMM as the it is working in a 5-dimensional space. The visualization of these data is not trivial either when the number of samples are not big enough. In our case, in the test scene we set up, only 15 samples were collected for each driver due to time limit. All the data was used to train the GMM and no test set was set up to validate this model. Nevertheless, the small set of samples still showed how the model was working with different data.

To compare the trained GMM model and the predicted trajectory, we assume the inputs are the same for each driver, which is [10m/s, -3.5m, 15m, -12m]. The output lateral motion over time for three drivers is shown in Fig. 22. In this specific case, one could observe that the driver 3 had a more 'defensive' style while driver 2 had more 'aggressive' behavior since the lane changing was done in a relative short amount of time, 3.5 seconds.

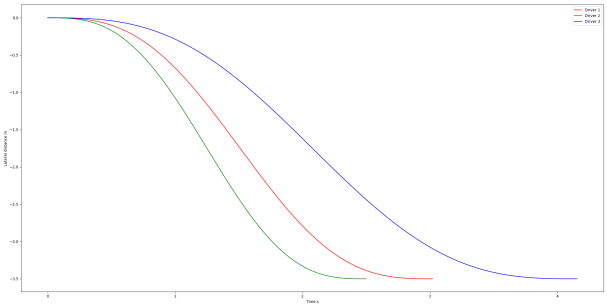

Fig. 22: Lateral Motion Given [10m/s, -3.5m, 15m, -12m]

Noted that lane changing is a very complicated task, the time to finish a lane changing could be affected by many factors. A shorter amount of time or a higher longitudinal speed could be 'defensive' as well if, for example, the driver consider lane changing under a certain situation tends to be dangerous and hence it should be finished as soon as possible. A high-dimensional GMM model is a very powerful tool to handle this kind of complicated situation, but at the same time, it will require more amount of driver's data.

REFERENCES

[1] C. Basu, Q. Yang, D. Hungerman, M. Sinahal, and A. D. Draqan, "Do you want your autonomous car to drive like you?" in *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI*. IEEE, 2017, pp. 417–425.

[2] L. Fridman, "Human-centered autonomous vehicle systems: Principles of effective shared autonomy," *arXiv preprint arXiv:1810.01835*, 2018.

[3] M. Hasenjager, M. Heckmann, and H. Wersing, "A survey of personalization for advanced driver assistance systems," *IEEE Transactions on Intelligent Vehicles*, 2019.

[4] A. Rosenfeld, Z. Bareket, C. V. Goldman, S. Kraus, D. J. LeBlanc, and O. Tsimhoni, "Towards adapting cars to their drivers," *AI Magazine*, vol. 33, no. 4, pp. 46–46, 2012.

[5] G. N. Bifulco, F. Simonelli, and R. Di Pace, "Experiments toward an human-like adaptive cruise control," in *2008 IEEE Intelligent Vehicles Symposium*. IEEE, 2008, pp. 919–924.

[6] S. Ramyar, A. Homaifar, S. M. Salaken, S. Nahavandi, and A. Kurt, "A personalized highway driving assistance system," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 1596–1601.

[7] J. Wang, L. Zhang, D. Zhang, and K. Li, "An adaptive longitudinal driving assistance system based on driver characteristics," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 1, pp. 1–12, March 2013.

[8] C. Vallon, Z. Ercan, A. Carvalho, and F. Borrelli, "A machine learning approach for personalized autonomous lane change initiation and control," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 1590–1595.

[9] V. A. Butakov and P. Ioannou, "Personalized driver/vehicle lane change models for adas," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 10, pp. 4422–4431, 2014.

[10] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," in *2007 American Control Conference*. IEEE, 2007, pp. 2296–2301.

[11] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "Casadi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.