A Learning-Based Autonomous Driver: Emulate Human Driver's Intelligence in Low-speed Car Following

Junqing Wei[a] and John M. Dolan[a,b] and Bakhtiar Litkouhi[c]

[a]Department of Electrical and Computer Engineering,
Carnegie Mellon University, Pittsburgh, PA 15213 USA;
[b]The Robotics Institute,
Carnegie Mellon University, Pittsburgh, PA 15213 USA;
[c]GM-CMU Autonomous Driving Collaborative Research Lab,
GM R&D Center, Warren, MI 48090 USA

ABSTRACT

In this paper, an offline learning mechanism based on the genetic algorithm is proposed for autonomous vehicles to emulate human driver behaviors. The autonomous driving ability is implemented based on a Prediction- and Cost function-Based algorithm (PCB). PCB is designed to emulate a human driver's decision process, which is modeled as traffic scenario prediction and evaluation. This paper focuses on using a learning algorithm to optimize PCB with very limited training data, so that PCB can have the ability to predict and evaluate traffic scenarios similarly to human drivers. 80 seconds of human driving data was collected in low-speed ($< 30 miles/h$) car-following scenarios. In the low-speed car-following tests, PCB was able to perform more human-like car-following after learning. A more general 120 kilometer-long simulation showed that PCB performs robustly even in scenarios that are not part of the training set.

Keywords: autonomous driving, traffic scenario evaluation, human behavior emulation, genetic algorithm

1. INTRODUCTION

Since the 1980s, autonomous driving has gradually become a fast-developing and promising area.[1] The abilities of autonomous vehicles have been extended from simple cruise control to adaptive cruise control, lane-keeping, intelligent route planning, off-road navigation and interacting with human-driven urban traffic.[2–4] Autonomous driving technology has the ability to provide driver convenience and enhance safety by avoiding some accidents due to driver error. While building autonomous driving algorithms, one significant issue is to make the autonomous vehicle be able to emulate human drivers' intelligence and driving styles. This will allow the passengers of the autonomous vehicle to feel more comfortable and convinced in the car's ability to drive itself. Also, drivers of surrounding vehicles will be able to better predict and understand the autonomous vehicle's behavior and more naturally interact with it. Therefore, it is desirable to train the autonomous driver with human driver's behaviors. As it is impossible to have all possible traffic scenarios as training data, a learning-based autonomous driving algorithm should be robust enough to perform in untrained situations. In this paper, we develop a learning-based PCB autonomous driver model and perform tests on the distance keeping module.

2. RELATED WORKS

In 2007, the DARPA Urban Challenge provided researchers a platform to test the latest sensors, computer technologies and artificial intelligence algorithms in an urban setting environment.[5] Basic interactions between autonomous vehicles and human-driven vehicles were proven in low-density, low-velocity traffic. However, to finish the race, most teams used rather conservative algorithms: the vehicles preferred avoiding difficult maneuvers in dense traffic by stopping and waiting for a clear opening instead of interacting with it and operating the vehicle similar to human drivers. Additionally, though those vehicles were fully functional in distance keeping,

Further author information: (Send correspondence to Junqing Wei)
Junqing Wei: E-mail: junqingw@cmu.edu, Telephone: 1 412 268 6228

Unattended Ground, Sea, and Air Sensor Technologies and Applications XII, edited by Edward M. Carapezza,
Proc. of SPIE Vol. 7693, 76930L · © 2010 SPIE · CCC code: 0277-786X/10/$18 · doi: 10.1117/12.852413

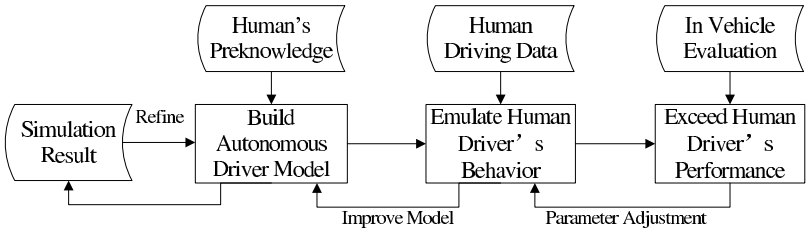

Figure 1. Autonomous Driver Model Development Approach

lane changing, parking, etc., they were designed to run without passengers. Therefore, no measures of comfort for potential passengers and drivers of surrounding vehicles were considered: the autonomous vehicle usually accelerated and decelerated as fast as possible to finish the race in the shortest time.

Adaptive cruise control (ACC) is one of the most widely used advanced driving assist systems in modern passenger cars.[6] A few car manufactures even equip their vehicles with ACC having 'stop and go' ability.[7,8] Most adaptive control systems use classical control theory-based models. Control theory has been proven to be very powerful only if accurate vehicle and interaction models are built. However, it is complicated to build an accurate deterministic model of human traffic. What is more, the control theory based ACC is built based on human's pre-knowledge and understanding of driving. The real human driving data is usually not used in constructing the controller.

One of the most famous and widely used car-following models was developed in 1961, which can emulate human drivers acceleration output using as few as three parameters.[9] This model, and some further enhancement of it, worked fairly well in microscopic traffic simulations, which focus on simulating an individual vehicle's behavior.[10] Though this kind of model is able to reproduce a driver's response to surrounding traffic, it only tries to emulate the human driver's output, instead of understanding the reason for making these decisions. Therefore, it has difficulty in dealing with more complicated scenarios.

Learning algorithms such as Artificial Neural Networks (ANN) have also been implemented in lane keeping and adaptive cruise control. NAVLAB at CMU implemented an ANN-based algorithm to perform steering control on freeways.[3] The ANN is trained by the front-view images of the vehicle and the human driver's steering wheel operation. After a few minutes of training, the vehicle is able to perform lane-centering by itself using the trained model. The major constraint of learning algorithms is that they depend too much on training. Therefore, no safety criteria can be verified in untrained scenarios . After learning, the ANN-based autonomous driving, at most, performs the same as its training data. Therefore, the autonomous vehicle's potential ability to exceed a human driver's performance is restricted.

In conclusion, a good autonomous driving algorithms should have the features that: enable them to use the human driver's prior-knowledge to better understand traffic scenarios, perform robustly in most traffic scenarios and be able to learn from human drivers to improve performance. Therefore, we propose a four step framework for developing better autonomous driver intelligence, which is shown in Figure 1. In the first step, the control model is built based on a human's prior-knowledge and could be verified in simulation to ensure the functionalities of driving. Then the learning algorithm is implemented to optimize the pre-knowledge-based control model. This enables the performance of the autonomous pilot to be improved through learning while avoiding the problems of using pure training based control models. Through this development process, our autonomous driver will be able to better interact with human traffic by emulating human driving behavior. It also retains the autonomous vehicle's potential to exceed human driver performance in some scenarios.

3. PREDICTION- AND COST-FUNCTION BASED ALGORITHM

In the first step of the development process, a Prediction- and Cost function-Based algorithm (PCB) for autonomous freeway driving was created.[11] The autonomous driver model was designed to model a human driver's decision process, including prediction and evaluation of traffic scenarios. By using prediction, we reduced the difficulty of building long-term heuristic cost functions. The evaluation was based on some human-understandable

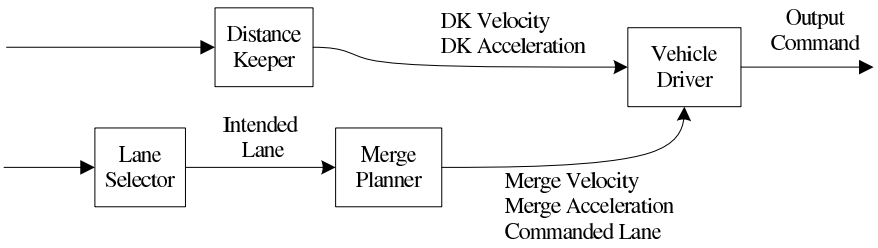

Figure 2. Freeway Driving Modules

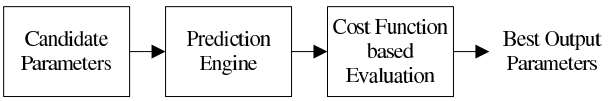

Figure 2. . Prediction- and Cost Function-Based Algorithm

cost functions, which were easy to extend. The algorithm is also highly reconfigurable so that it can be adjusted to perform different driving styles from conservative to aggressive.

3.1 Autonomous Freeway Driving Control Modules

In PCB, we separate the freeway driving ability into three modules.[12] These are the distance keeper, lane selector and merge planner. The data flow is depicted in Figure 2. The role of the distance keeper is to keep a reasonable distance from the leading vehicle. The lane selector outputs the intended lane, i.e., the lane the autonomous vehicle wants to merge into. Whenever the intended lane is different from the current lane, the merge planner will be triggered. It adjusts the vehicle's position and speed to find a best merging opportunity. In this paper, the learning algorithm will be implemented and evaluated mainly in the distance keeper. The distance keeper will be trained to emulate a human driver's behavior, while the other two modules, the lane selector and merge planner, will still be functional to perform lane changes in the integration tests.

3.2 Algorithm Framework

A diagram of PCB is shown in Figure 3. There are three primary steps: candidate strategy generation, prediction, and cost function-based evaluation. In the candidate parameter generation step, a set of candidate strategies is generated. In the distance keeper module, the generator produces 20 different acceleration values that range from $-3.0m/s$ to $1.8m/s$. Then the strategy set and the map of the current moving vehicles are sent to the prediction engine, which generates a series of simulated scenarios of the following t seconds. The cost function-based evaluator then begins to compute the cost of each strategy for each scenario.

By using this mechanism, we successfully separate the complicated behavior strategy generation process into two relatively independent parts. The prediction engine only considers how to accurately generate simulated scenarios, while the cost function block implements and imitates a human driver's evaluation of a given scenario.

3.3 Designed for Learning

PCB is built based on a development process shown in Figure 1. It is therefore a model that can be efficiently improved through learning. In the learning process, the prediction and evaluation parameters are optimized to fit a human's predicting and evaluating process. Compared with other learning-based or data-based models, PCB is designed to learn not only human drivers' control output (desired acceleration, desired velocity, etc.) but the way human drivers evaluate traffic scenarios while making the decision. Therefore, after learning, PCB will be able to deal with scenarios not restricted to the training data, as it better understands how humans make decisions in driving.

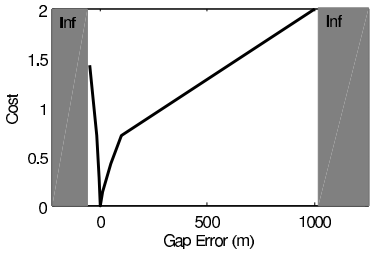

Figure 3. Distance keeper progress cost: $C_{DKpro}(\Delta d_{gap})$, with vertices: (-25,-1.5), (-15,-0.9), (-5,-0.14), (0,0), (10,0.14), (50,0.43), (100,0.7), (1000,2)

3.4 Parameters and Cost functions

There are a few key parameters in this emulated human's car-following decision-making process.

- Prediction length (t_{pre})

  The length of prediction is a key parameter in PCB. The prediction engine predicts the future scenarios based on some basic assumptions about the microscopic traffic, such as slowing down when there is a slower leading vehicle, keeping constant velocity otherwise, etc. Using prediction, PCB can behave in advance to achieve control goals including smoother accelerations and shorter response time. However, if the length of prediction is too long, the vehicle's decision might become too sensitive to small changes in the surrounding vehicles' states and the decision will oscillate.

- Desired gap ($d_{desired}$)

  Distance keeping aims to maintain a certain distance between the host vehicle and the leading vehicle. It should be a linear function of the vehicle's velocity, which is verified by some exisitng autonomous or human driver models.[13] In PCB, we are using a linear model to represent this, as shown in Equation 1. In the learning process, both D_0 and k_{vgain} will be adjusted to best fit human driver's desired distance in car following.

$$d_{desired} = D_0 + k_{vgain} v \tag{1}$$

- Gap error cost

  The default gap error cost is shown in Figure 4. The gap error is the difference between the actual gap to the leading vehicle (computed by Equation 1) and the desired gap . As shown in the figure, the cost function is represented by some vertices. When the gap error is less than 0, which means we are too close to the leading vehicle, the cost increases significantly. The cost is lower when the gap error is between 0 meters and 10 meters, which is the preferred following distance error. The default gap error cost is built based on human domain knowledge and some basic simulations. The limits of the horizontal axis represent the minimum and maximum gap errors the evaluator can accept. The cost is infinite outside this range.

- Acceleration cost

  While driving a car, experienced human drivers will try to avoid large accelerations for greater comfort. Therefore, a comfort cost $C_{comfort}$ is built to represent this logic, as shown in Figure 5.

- Safety cost

  The safety costs are built to penalize strategies that lead the vehicle into unsafe situations. Normally the safety cost is very low. But in situations in which the vehicle is too close to the leading vehicle or is getting close too fast, the safety cost will dominate and force PCB to perform safer maneuvers.

- Cost weights

  Once the scenarios corresponding to different potential strategies are generated, PCB begins to compute the cost for each strategy. The weights of those costs show the PCB's preference in evaluating strategies.

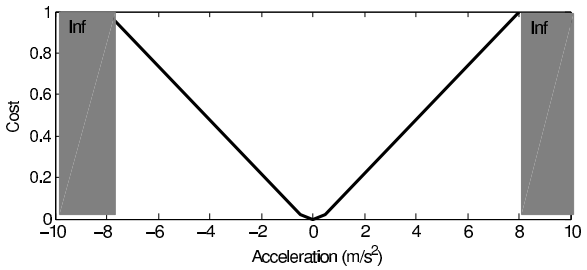

Figure 4. Comfort cost: $C_{comfort}(acc)$, with vertices: (-8,1), (-0.5,0.02), (0,0), (0.5,0.02), (8,1)

Table 1. Default Parameter Selection

Prediction Length (t_{pre})	0.6
Desired Gap ($d_{desired}$)	10 + 0.88v
Gap error cost	(as shown in Figre 4)
Acceleration cost	(as shown in Figure 5
Gap error cost weight	50
Acceleration cost weight	10
Safety cost weight	10

For instance, if the weight of the gap error cost is large while the acceleration cost is small, PCB will prefer to strictly follow the leading vehicle with certain clear distance with relatively large acceleration amplitude. If the weights are reversed, it will follow the leading vehicle more flexibly with smoother velocity adjustments.

The default parameters in the implementation of PCB before learning are shown in Table 1. The parameters are selected based on an estimation using human domain knowledge and were preliminary adjusted and verified in some simple simulations.[11]

4. IMPLEMENT LEARNING ALGORITHM

4.1 Integrating learning into the Autonomous Driving System

Our autonomous vehicle, Boss, uses a real-time control platform called TROCS (Tartan Racing Operator Control System). Most of the artificial intelligence used in the competition was developed and tested using the simulation mode of TROCS, so its performance and accuracy have been well proven. Another benefit of using this platform is that after safety and performance verification in the simulator, the algorithm can be directly ported into the vehicle and tested on-road.

The PCB algorithm is ported from TROCS to MATLAB, which can optimize the parameters of PCB without the real-time restrictions of TROCS. MATLAB's optimization toolbox also simplifies the implementation of optimization algorithms. We tested the implementation of PCB in TROCS and in MATLAB, as shown in Figure 6. Compared to the MATLAB implementation, there are small noises in the TROCS simulation, which may be caused by its inconsistent real-time simulation refresh rate. In Figure 6, both implementations output almost the same strategy. Therefore, after optimizing the algorithm in MATLAB, we only need to adjust PCB's parameters in TROCS to the optimized values to make it ready for verification either in simulation or on the road.

4.2 Data Collection

The data were collected from five manual driving runs of Boss in a low-speed ($40.23km/h$ or $< 25miles/h$) car-following scenario. Each test run is about 80 seconds on a road in the experiment field which is 500 meters long. The leading vehicle changed velocity to generate stop-and-go scenarios, as in traffic jams. The host vehicle collected its own velocity and the distance and velocity of the leading vehicle. In the data collection runs, the driver of the host vehicle was asked to follow the leading vehicle in his normal driving style.

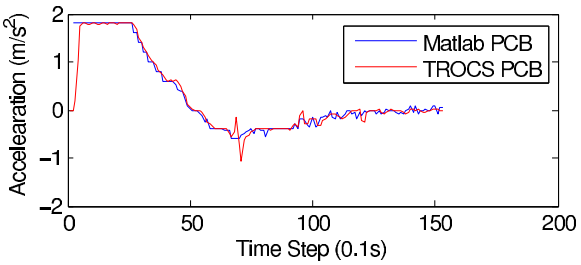

Figure 5. Verification of implementation of PCB in Matlab

Obviously, the described scenario is only a very limited part of all the possible traffic scenarios we could encounter while driving a car. Therefore, the algorithm and the learning mechanism should be able to optimize themselves with limited training and use the knowledge learned to deal with more complicated scenarios. To test this, we selected one data set to train PCB, which is around 80 seconds of driving. The other four data sets were used for testing.

4.3 Learning Algorithm

In this paper, the autonomous driving algorithm, PCB, is optimized off-line. Therefore, we only need to run the optimization once to find the best parameter set. In this case, the genetic algorithm will be a reasonalbe choice, which works robustly in finding the global optimum in even non-convex or high-dimensional search spaces.

The input data are collected with the data format: $Data_i = \{V_{lead}, D_{lead}, V_{host}, a_{host}\}$, in which a_{host} is the test driver's response to the traffic scenario. The goal of learning is to make the PCB distance-keeping model output an acceleration command similar to that of human driver. Therefore, a fitness function is constructed for the optimization algorithm to be minimized, as shown in Equation 2.

$$Fit = \frac{1}{n}\sum_{i=0}^{n} |acc_{training} - acc_{current}| \tag{2}$$

4.4 Refine Training Data

Based on previous research on human driver models, there is a relatively constant delay γ between a human's making a decision and the vehicle's performing the actual acceleration, as shown in Equation 3, in which S_t is the microscopic traffic scenario at time t.

$$acc_t = f(S_{t-\gamma}) \tag{3}$$

This is caused by the human's mental and physical response times and the vehicle's mechanical and dynamic delay. To avoid PCB's learning this extra delay, we manually reduced the delay by offsetting the timestamp of the acceleration output in the training by $t_{offset} = 0.97s$. t_{offset} is approximately computed by analyzing the cross-correlation of the leading vehicle's speed and the host vehicle's speed. To accelerate the process of the genetic algorithm-based optimization, 80 training data were uniformly sampled from the 80 seconds of the collected data, which includes a total of 800 data points.

4.5 Learning PCB Parameters

The learning algorithm's performance is evaluated using two metrics. Firstly, some parameters of the optimization process are extracted, such as the best fitness values of 0, 50, 125 and 250 generations. Then the cost function shapes in the evaluation step are reconstructed. As the number and coverage of training data are limited, we will evaluate whether the optimized parameters lead to any over-fitting or unreasonable results.

Table 2. Genetic Algorithm Optimization Results, N_{par} is the number of parameters being optimized, $Fit_{0,50,125,250}$ are the values fitness functions in the $0, 50, 125, 250$ generations, r_{err} is the error percentage which is computed by Equation 4

	Description	N_{par}	Fit_0	Fit_{50}	Fit_{125}	Fit_{250}	r_{err}	t_{pre}	$d_{desired}$
0	Default parameter	N/A	(1.17)	(1.17)	(1.17)	(1.17)	60.43%	6.0	$15 + 1.12V$
1	Prediction+desiredGap	3	0.53	0.50	0.47	0.46	23.76%	6.3	$-6 + 1.56V$
2	1 plus Weights	7	0.60	0.47	0.46	0.45	23.24%	4.82	$-0.15 + 1.87V$
3	2 plus Cost Function Shape	20	0.55	0.35	0.33	0.32	16.53%	2.72	$1.98 + 0.32V$
4	3 with slope description	18	0.55	0.35	0.28	0.27	13.95%	3.38	$1.18 + 0.57V$
5	4 with constraints	18	0.43	0.29	0.28	0.27	13.95%	1.98	$1.45 + 0.75V$
6	5 with x adjusting	29	0.84	0.35	0.33	0.33	17.04%	1.49	$1.72 - 0.60V$
7	Parabola	7	0.70	0.35	0.35	0.33	17.04%	4.24	$-1.99 + 2.29V$
8	Parabola and linear	8	0.57	0.34	0.34	0.31	16.01%	4.26	$-2 + 2.19V$
9	LCF	1	0.78	0.40	0.40	0.40	20.66%	N/A	N/A
10	NCF	3	0.88	0.37	0.36	0.36	18.59%	N/A	N/A

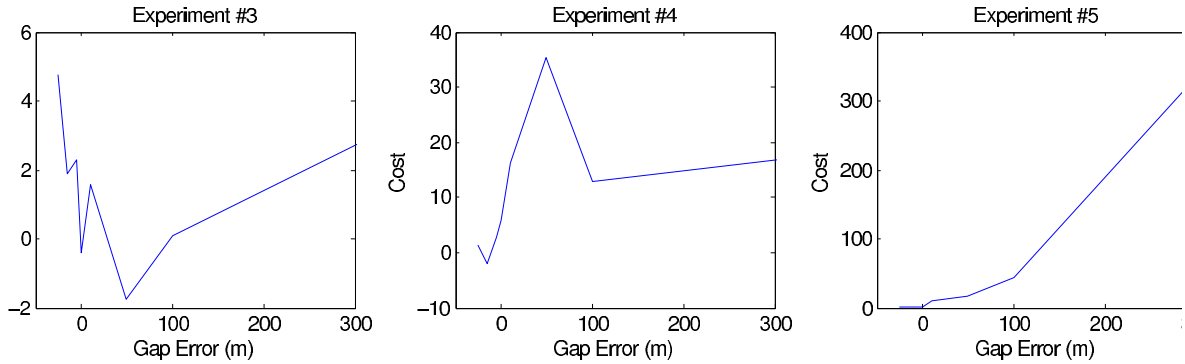

Figure 6. Learned Cost Function Shapes for Experiments #3,4,5

4.5.1 Prediction Length and Desired Gap

The prediction length and the desired gap are two of the most important factors for PCB to make driving decisions. Therefore, in the first set of experiments, we only optimize these parameters. The result in Table 2 shows that by only adjusting these parameters, the average acceleration command error can be as low as $0.46m/s^2$. The error ratio r_{err}, which is computed in Equation 4, is 23.67%, meaning that the average error amplitude is 23.67% of the max acceleration in the training set.

$$r_{err} = Fit_{final}/max(acc_{train}) \tag{4}$$

4.5.2 Cost Function Weights

The weights of the gap error cost, acceleration cost and safety cost imply the driving style of the autonomous vehicle. In this experiment, the weights of the three cost functions are adjusted to emulate the test driver's driving preference. Compared to only adjusting the prediction and desired gap parameters, the average acceleration error decreases to $0.34m/s^2$. According to these two tests, we found that without the consideration of optimization complexity, the more flexible the model is, the better the fitting results.

4.5.3 Cost Function Shapes

To pursue better emulation results, in Experiment #3, the y values of the cost function vertices are also optimized. As shown in Table 2, the emulation result is $0.32m/s^2$, $r_{err} = 16.53\%$ Another cost function representation method based on the slope of each section between vertices is tested in Experiment #4. Because it is the slope, not the actual value of those vertices that influence the evaluation of different strategies, using slope leads to a better result: $0.27m/s^2$, $r_{err} = 13.95\%$.

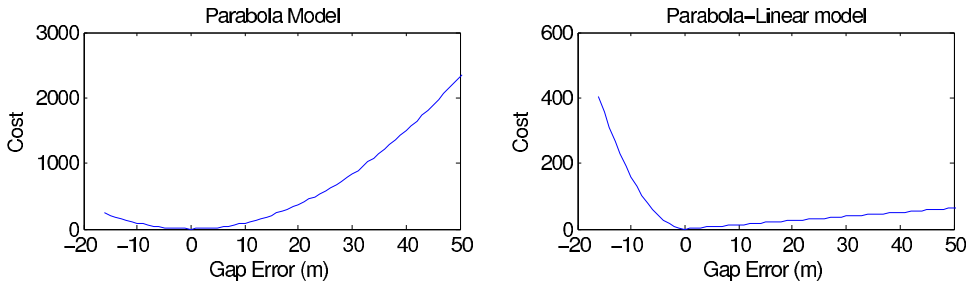

Figure 7. Learned Cost Function Shapes for Experiments #7,8

The learned cost function shapes from Experiments #3 and #4 are shown in Figure 7. It is obvious that though the fitness function is optimized to a lower value, the learned shapes of cost function are not reasonable. Since it does not agree with the logic that the closer the input to the desired gap, the lower the cost should be. It is possible to solve this problem by train the model using more data. However, as it is impossible to train PCB in all possible scenarios, the shape of the cost function should also be restricted for more efficient learning.

Such constraints are implemented in the training process in Experiment #5. We required the shape of the vertices-based cost function to have restricted slope. When the error is smaller than 0, the slope is negative ; in contrast, when the error is larger than 0, the slope is positive. The constrained learned cost function from Experiment #5 is also shown in Figure 7.

To achieve potentially better emulation results, both the horizontal and vertical coordinates of the vertices representing the cost function are optimized with the same restriction as in Experiment #5. After 200 generations of optimization, the fitness function is trapped in a local minimum point. As shown is Table 2, at this local minimum point, the desired gap is computed by $1.72 - 0.60V$, meaning that the faster the vehicle goes, the smaller distance it will keep to the leader, which is very undesirable. This is because there is a larger number of parameters to optimize, and the optimization space of this problem is not smooth enough. This vertices-based cost function may exponentially increases the search space each vertice will insert two parameters for learning. It makes the genetic algorithm much more inefficient in finding the global optimum results.

Table 2 also shows that the flexibility of cost function shape does not provide big benefits but instead causes a serious over-fitting problem. For the acceleration and desired gap error, the cost should represent the logic that the larger the error is the larger the cost it will be. It is also obvious that the cost function curve should be smooth as in the evaluation, which means similar scenario should correspond to similar cost value. Therefore, a straight-forward form of the smooth cost function, the parabola function, is used as the assumption for training. As shown in Experiment #7, the performance is not as good as that obtained using complicated cost functions, but the number of parameters is only 6 compared to 28. The optimum parameter set convergence process is much faster and the over-fitting problem is also avoided effectively.

The learned parabola cost functions are shown in Figure 8. One problem of this cost function restriction is that when the gap error is larger than zero, the cost increases proportionally to the square of the gap error. This may cause the host vehicle to accelerate at its maximum acceleration when the gap error is large. However, the real situation is that the human will control the velocity so that the host vehicle gets close to the leading vehicle gradually. Therefore, when the gap error is larger than zero, a linear model is used instead of the parabola model shown in Figure 8. When the gap error is smaller than 0, we are still using the parabola model. The result of experiment #8 shows that this modification helps improve the evaluation of traffic scenarios and leads to a better fitness function value.

4.6 Linear/Nonlinear car following models

The linear/nonlinear model as shown in Equation 5 is a widely-used model for car following in traffic simulations.[9,10]

$$acc = \mu_1 * \frac{(V_{host})^{\mu_2}}{(D_{leading})^{\mu_3}} * (V_{host} - V_{lead}); \tag{5}$$

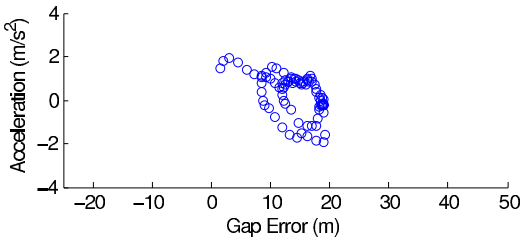

Figure 8. Training Data Coverage Map

The non-linear car following (NCF) model has three parameters: μ_1, μ_2 and μ_3. The linear model (LCF) is the same model with fixed parameters $\mu_2 = 0$ and $\mu_3 = 1$. The NCF and LCF models are also trained here with the same training data using a genetic algorithm for performance comparison. As shown in Table 2, the LCF and NCF models fit a human's output well with only 1 or 3 parameters. However, the NCF and LCF models are simple input-output fitting functions. The algorithm does not select the output strategy based on the evaluation of current traffic scenario and possible future scenarios if perfoming certain control. Though they have the benefit of use fewer parameters and easier to be tuned, given only low speed car following data, they are not able to perform full speed spectrum distance keeping. In scenarios haven't encountered in the training set, their robustness and car following performance are not as good as the PCB model, which will be shown in the integration test section.

4.7 Model selection

As shown in Table 2, some good (smaller) fitness function values correspond to the over-fitted cost function shapes. One reason for this problem is that the number and coverage of training data are not enough. As shown in Figure 9, the training data only cover a small amount of all the acceptable input and output for PCB. However, it is true that there will always be traffic scenarios that have not been trained. Therefore, the learning mechanism's adaptablity given limited data is of great importance. The number of parameters is another key factor influencing the learning performance. Though we are using the genetic algorithm, which is of proven performance in finding global optimum, as the search space exponentially increases, it is difficult and time-consuming for it to converge to the best result.

Based on these analysis, the parabola-linear model is selected as our final PCB model for further tests. It is a good trade-off between the dimension of the optimization space and the final performance. Compared with the original vertices-based cost function representation, the number of parameters in PCB decreases from 27 to 7. Over-fitting is also efficiently avoided by using a parabola-linear cost function shape.

5. PERFORMANCE EVALUATION

To analyze the performance of PCB after learning, two tests were implemented in the simulation mode of the real-time autonomous vehicle control platform TROCS. The first one was a low-speed car-following test, in which the autonomous vehicle was controlled by the emulated PCB to perform distance keeping. We replayed the leading vehicle's velocity and position recorded in the data collection step for the host vehicle to follow. The second test used the learned PCB to perform autonomous driving in a more general 3-lane freeway scenario. In this case, the speed was not limited to 40.23 km/h (25 miles/h) and lane changeing ability is activated as well. In the integration test, the statistical data were extracted and compared between the learned PCB and the original PCB.

5.1 Low-speed car-following

Figure 10 shows the PCB's low-speed car-following performance in the testing set we recorded. Compared to the following performance before learning, the learned PCB can better emulate a human driver's car-following behavior. While performing car following, the acceleration, velocity and following distance error between the human and autonomous driver are decreased significantly. The acceleration, velocity and following distance errors' amplitudes and error percentages are shown in Table 3. According to these tests, it can be concluded

Table 3. Low-speed car-following performance

	$\delta a_{ave}(m/s^2)$	$\delta v_{ave}(m/s)$	$\delta d_{ave}(m)$	$\frac{\delta a_{ave}}{max(a_{test})}$	$\frac{\delta v_{ave}}{max(v_{test})}$	$\frac{\delta d_{ave}}{max(d_{test})}$
Leanred PCB	0.33	0.71	2.68	15.97%	5.46%	7.51%
Original PCB	0.76	2.40	9.11	37.00%	18.47%	25.53%

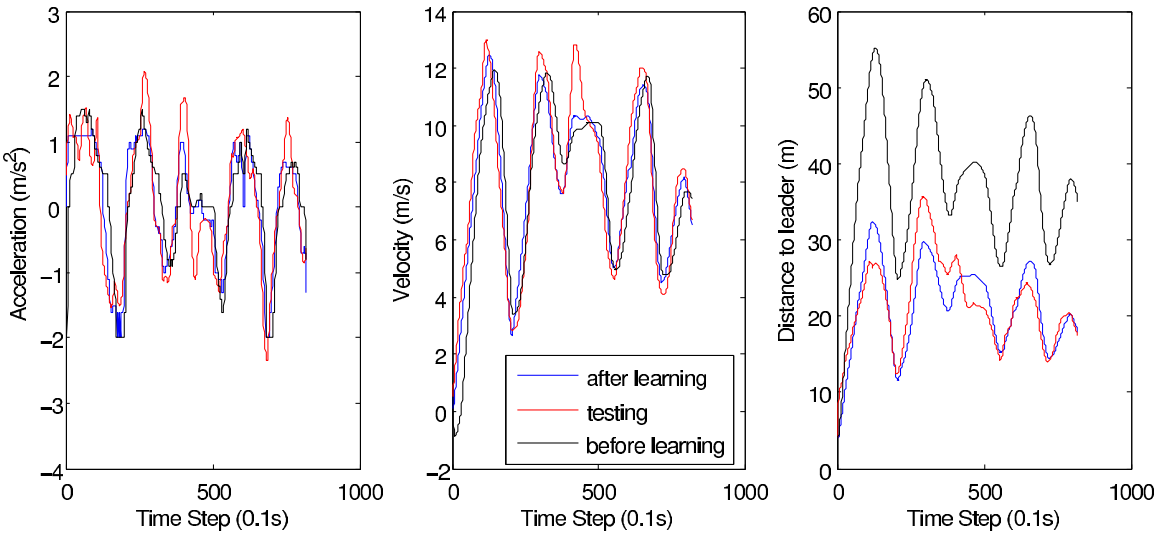

Figure 9. Low-Speed following simulation

Table 4. Comparison of freeway driving performances

	NCF	Learned PCB	Original PCB
$N_{LaneChange}$	20	30.0 ± 13.9	28.5 ± 9.2
$acc_{ave}(m/s^2)$	0.14 ± 0.57	0.22 ± 0.41	0.28 ± 0.50
$t_{arrival}(s)$	2249	2039.3 ± 180.7	1966.3 ± 221.9
$t_{D<10m}(s)$	0.00	0.00 ± 0.00	0.00 ± 0.00
$t_{brakeD<10m}(s)$	0.00	105.6 ± 146.1	59.3 ± 74.3
$t_{brakeD<0m}(s)$	0.00	0.0 ± 0.0	0.0 ± 0.0

that through learning, the PCB algorithm, in general, learned how to evaluate a traffic scenario and generate commands in a way more similar to that of human drivers. Compared to the PCB model before learning, the number of parameters of the parabola-linear cost function-based PCB model is reduced from 29 to 7, and the emulation performance is improved.

5.2 Integration test

To verify learning-based PCB's adaptability and performance in more complex traffic scenarios, a 40 kilometer-long road was built in TROCS's simulation mode. Simulated human traffic capable of simple distance keeping and lane changes was also added to the simulator. Both velocity and distance between simulated vehicles had Gaussian distributions with parameters: $\mu_v = 20m/s, \sigma_v = 6m/s, \mu_d = 40m, \sigma_d = 12m$. To get the statistical results, each experiment was run for three times with different randomly generated traffic having the same distribution.

Table 4 shows that after learning from only low speed car following data, PCB also has the ability to travel at high speed on freeways. Though the training data did not cover any high-speed driving, the performance of PCB is not sacrificed by adjusting the prediction and evaluation parameters. This is because there is a safety cost when PCB is evaluating traffic scenarios. Though the weight of the cost is adjusted while learning, in a challenging scenario, the safety cost increases so significantly that PCB chooses safer strategies. Compared with the original PCB algorithm, the learning-based PCB's average acceleration amplitude is smaller. In Table 4, the total time when braking distance is smaller than 10 meters ($t_{brakeD<10m}$) is larger in the learned PCB. That is because the learned desired distance is smaller than that computed using the default parameters. However,

the safety criterion is still strictly followed by the learned PCB, and it has the similar statistical performance to that before learning.

In contrast, with limited training data, the NCF model failed to finish two of the three test runs. In the two failed test runs, the vehicle controlled by the NCF non-linear model did not slow down early enough so the error recovery mode of the vehicle was triggered to emergency brake. The performance of the only finished run is also not as good as that of PCB, with an average speed decrease of around 13%. Compared with the PCB model, which can work in most traffic scenarios, the NCF model is only functional in car-following scenario within the velocity range covered by the training data. Therefore, in scenarios that the vehicle is starting up, at high speed or with no leading vehicles, PCB has much higher performance. Though it is possible to improve NCF model by using all speed range (e.g. $0 - 120km/h$) training data or using different parameter sets for different speed range, PCB is proved to have better utilization of human domain knowledge to achieve faster and more robust learning. In a word, the integration tests show that the learning-based PCB's prediction- and evaluation-based decision mechanism is functional and promising for autonomous vehicle control.

6. CONCLUSION

In this paper, genetic algorithm-based learning is proposed to improve autonomous driving by emulating human driver behavior. Using this mechanism, the learning-based prediction- and cost function-based algorithm well emulates a human driver's traffic scenario evaluation ability. In a low-speed vehicle-following test, the learned PCB model controls the vehicle very similarly to human drivers with only about 5% velocity difference. In a 120-kilometer long integration test, the learning-based PCB's adaptability was demonstrated. Though the training data were limited, the learned PCB could still deal with many untrained scenarios, including high-speed driving. In the future, a more realistic vehicle dynamic model will be used for better control of the vehicle's acceleration. On-road experiments will be implemented to determine what human passengers perceive as a natural and comfortable driving style. More data will also be collected on human driving in a wider range of scenarios, including high-speed and high-density driving. The algorithm can be further refined based on these experiments and potentilly achieve higher performance.

In conclusion, the PCB algorithm and the learning mechanism provide autonomous pilot the ability to learn from human driver's behaviors and driving styles. This approach has great potential to make autonomous vehicles behave more naturally while sharing the road with human traffic.

7. ACKNOWLEDGMENTS

This work was supported by General Motors through the GM-Carnegie Mellon Autonomous Driving Collaborative Research Laboratory. The authors also would like to thank the Tartan Racing Team in the DARPA Urban Challenge, who built the vehicle control platform and the simulator.

REFERENCES

[1] Dickmanns, E. D., "Vehicles capable of dynamic vision: a new breed of technical beings?," *Artificial Intelligence* **103**, 49–76 (Aug. 1998).

[2] Gregor, R. et al., "Ems-vision: a perceptual system for autonomous vehicles," *IEEE Journal of ITS* **3**, 48–59 (March 2002).

[3] Pomerleau, D., "Alvinn: An autonomous land vehicle in a neural network," in [*Advances in Neural Information Processing Systems 1*], Touretzky, D., ed., Morgan Kaufmann (1989).

[4] Thorpe, C., Herbert, M., Kanade, T., and Shafer, S., "Toward autonomous driving: the cmu navlab. i. perception," *IEEE Expert* **6**, 31–42 (Aug. 1991).

[5] Urmson, C. et al., "Autonomous driving in urban environments: Boss and the Urban Challenge," *Journal of Field Robotics* **25**(8), 425–466 (2008).

[6] Ardalan Vahidi, A. E., "Research advances in intelligent collision avoidance and adaptive cruise control," *IEEE Transactions on Intelligent Transportation Systems* **4**(3), 143–153 (2003).

[7] Venhovens, P., K.Nabb, and Adiprasito, B., "Stop and go ruise control," *I.J. of Automotive Technology* **1**(2), 61–69 (2000).

[8] Delorme, D. and Song, B., "Human driver model for smartahs," *UC Berkeley: California Partners for Advanced Transit and Highways (PATH)* (2001).

[9] Gazis, D. C., Herman, R., and Rothery, R. W., "Nonlinear follow-the-leader models of traffic flow," *OPERATIONS RESEARCH* **9**(4), 545–567 (1961).

[10] Ahmed, K. I., *Modeling Drivers' Acceleration and Lane Changing Behavior*, PhD thesis, Department of Civil and Environmental Engineering, MIT, Cambridge, MA (1996).

[11] Wei, J. and Dolan, J. M., "A robust autonomous freeway driving algorithm," *2009 IEEE Intelligent Vehicle Symposium (IV2009)* (2009).

[12] Baker, C. R. and Dolan, J. M., "A case study in behavioral subsystem engineering for the Uuban Challenge," *IEEE RAM Special Issue on Software Engineering in Robotics* (2008).

[13] Xu, Q., Hedrick, K., Sengupta, R., and Vanderwerf, J., "Effects of vehicle-vehicle/roadside-vehicle communication on adaptive cruise controlled highway systems," *Vehicular Technology Conference* **2**, 1249–1253 vol.2 (2002).