Learning a Racing Strategy from the Geometric Features of a Racetrack

Catherine Weaver
catherine22@berkeley.edu

Sara Shonkwiler
sara_shonkwiler@berkeley.edu

Jose Zavala
jose_zavala@berkeley.edu

Team SCJ

December 2, 2020

Abstract

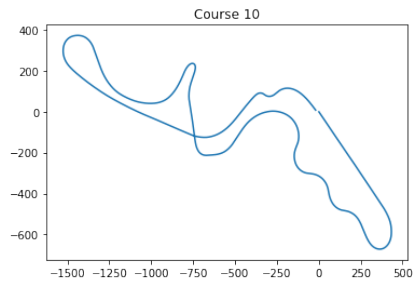
Finding a trajectory that minimizes the lap time of a car around a racetrack is a key step in optimal control of race-cars. Formulating the minimum time problem can be difficult to solve because of the nonlinear vehicle dynamics; however, researchers have simplified the problem into two steps: path planning and velocity maximization. Since a nearly optimal path can be found using minimum curvature path planning which depends only track geometry, we propose a method to calculate the maximum velocity a vehicle can achieve given its path. We introduce a featurization of the way points along a path that incorporates the path geometry before and after a given point, and analyze the effect of the featurization hyperparameters. We determine the features of the path that most influence the velocity function through sparsity-encouraging LASSO regression, and then compare the performance of ordinary least squares, ridge regression, and polynomial features of varying degrees on the important track features. Github Link: `https://github.com/cwj22/CS289Project`

1 Introduction

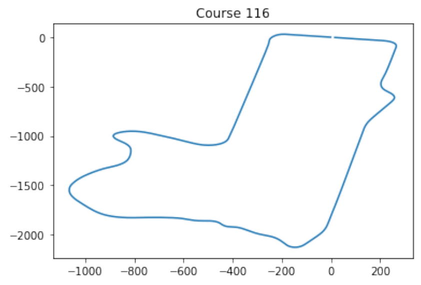
A key aspect of control of racing vehicles is determining a minimum time trajectory for navigating the track. The method to find a minimum lap time was initially an interest of professional racing teams, and early approaches found the optimal path and velocity or optimal control inputs by formulating it as control problem with nonlinear vehicle dynamic models [1], [2]. However, solving these nonlinear optimization problems is non-trivial, and computation of an optimal trajectory cannot be performed in real time. Online computation of an optimal racing trajectory would allow for a new optimal trajectory to be generated whenever the vehicle significantly deviates from any offline path, as well as modify the optimal trajectory when encountering track changes or obstacles such as other vehicles.

Minimum lap time formulations plan both the optimal path and velocity simultaneously; however, researchers have split the problem into a two-step process of first planning a minimum curvature lap time and then finding the maximum velocity achievable by the vehicle for the fixed path [3]. The selection of a minimum curvature path has shown only marginally worse performance than a minimum time formulation, with significantly less computation required [4]. These two step methods first plan a path which minimizes the curvature around the track, and therefore only requires track geometry for planning. Heilmeier et al. [4] was able to formulate the minimum curvature problem as a quadratic program.

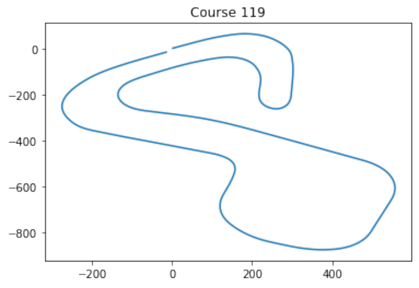
After the path is planned, the maximum velocity the vehicle can achieve around the minimum curvature path must be determined. The vehicle's velocity is constrained by the curvature of the track, the engine acceleration capacity, the braking capacity, and the friction between the tires and road. A common approach for determining an achievable velocity is a "three pass" approach described in [5] based on the work of [6]. The approach breaks the velocity planning problem down into additional steps. Since the velocity around a curve is constrained by the curvature, the points of minimum velocity is determined so that the vehicle is able to achieve the required lateral acceleration around the curve. The velocity planner then employs acceleration and deceleration limits, which must be identified for each vehicle, to determined how fast the car can accelerated out of a curve and how soon the car must begin decelerating to reach the minimum velocity required by the curve. Finally, friction limits are enforced by using nonlinear models to calculate the forces experienced by the tires.



(a) Course 10.



(b) Course 116.



(c) Course 119.

Figure 1: Examples of Sony's Gran Turismo Sport (GTS) Racing Game Tracks.

Whether the velocity planner is able to achieve a maximum velocity depends heavily on the accuracy of the models, and a system identification must be performed to determine the appropriate parameters for each vehicle and track surface. Many approaches assume a constant acceleration and braking limit [3], [4]; however, engine acceleration changes at different speeds and engine RPMs, and brake capacity is influenced by the temperature of the tires and road surface. The friction limits, which are most commonly constrained using a friction circle, depend on the lateral, longitudinal and normal forces on each tire. Models that capture the highly nonlinear behavior of the tires (for example the well-known Magic tire model [7]) are quite difficult to completely isolate the parameters for system identification and the complexity of the tire models increases the computational complexity of enforcing the friction constraints. Additional effects, such as aerodynamic down force on a spoiler, can increase the normal force experienced by the tires; a car with a spoiler could therefore experience higher lateral and longitudinal forces, allowing in higher cornering velocities.

In this work, instead of identifying vehicle parameters to input into a model for velocity planning, we attempt to learn the maximal velocity function by determining the relationship from the path geometry to the maximum achievable velocity for a given vehicle. Our work is inspired by Kapania and Gerdes, who attempted a learning-based approach to sample experimental runs around a racetrack using a velocity planned with different values of road surface friction [8]. In this work, the researchers again use the "three pass" approach to determine plan a velocity for a given minimum curvature path. They then vary the planned path by changing the road surface coefficient, μ, resulting in multiple velocity profiles around a racetrack. They experiment by controlling an actual racecar to follow the different trajectories, and determine the best friction coefficient to use for planning at section of the racetrack via a greedy search algorithm. Since they employ the use of a real racecar, they are limited by the amount of different trajectories and tracks they can include in the search to determine the best value of μ at different parts of the track.

In this paper, we attempt to learn the velocity function for a given path without the need of the "three pass" method which requires development of an accurate vehicle model and system identification. Instead, we learn the velocity function for a vehicle directly from the path geometry using human racing results from Sony's Gran Turismo Sport (GTS) racing game, which contains a highly accurate dynamic vehicle model. Given the path the racer took around a track, we extract features from the track and then aim to fit a function to describe the velocity for each waypoint along the track. The methods we use are ordinary least squares (OLS) regression, ridge regression, polynomially augmented features, Lasso regression, and simple neural network. We validate by testing performance on a set of tracks that are previously unseen by the model (test tracks)

This paper is organized as follows. In Section 2 we describe the methods of collecting and pre-processing the data, training the velocity function using regression models, and testing and validating the velocity function using different tracks. In Section 3 we discuss the results of the various models.

2 Method

2.1 Data Collection and Pre-Processing

We collect data from the racing simulation game *Gran Turismo Sport* (GTS), which is available on *Playstation* and is regarded as a highly realistic driving simulation [9]. GTS models many physical phenomena experienced by race cars, and the optimal velocity for a vehicle depends on the physical characteristics and states of the car

Car state	Units	Description
X	m	Global X position at time t
Y	m	Global Y position at time t
Z	m	Global Z position at time t
ΔX	m	Difference in X position at time t and $t+1$
ΔY	m	Difference in Y position at time t and $t+1$
ΔZ	m	Difference in Z position at time t and $t+1$
Θ_X	[rad]	Rotation of the car about the X axis
Θ_Y	[rad]	Rotation of the car about the Y axis
Θ_Z	[rad]	Rotation of the car about the Z axis
w	m	width of the track at the current position
κ	1/m	magnitude of curvature of path at the current position
V	m/s	Vehicle speed at the current position

Table 1: Table of car states recorded at each time step.

during the simulated game. Therefore the problem is very similar to a real-world racing scenario; however the GTS system provides significantly more data for analysis.

We collect records from all human drivers with a high rating that performed in a "Time attack" for the from March 2019 to February 2020 using the vehicle Alfa Romeo 4C Gr.4. In total there are 624 records on 33 different tracks. Each record contains information about the state of the car throughout the time trial, including global position of the car, rotational position of the car, curvature of the path, and width of the track. The state of the simulated car is sampled with a frequency of 60 Hz. Since we are interested in predicting the best velocity profile around the track (i.e. the one that minimized lap time), for each of the 33 tracks we only analyze the record that achieved the minimum lap time. We are only analyzing one car, since each car will have a different velocity function depending on car characteristics.

Each record of the minimum lap time for a given track contains the car states at each time step sampled at a frequency of 60 Hz. The states included in the data at each time step are listed in Table 1.

Based on intuition from vehicle model based approaches [3], [4], the velocity at a given position of the path is dependent on the path before and after the current position. Therefore, we wish to construct a feature matrix that includes information about the path prior to the current position, the current position, and the path that the driver will take in the future. We then wish to use this path information to create a model to describe the velocity, V. However, by the nature of how the car states are sampled, the velocity information is contained in the $\Delta X, \Delta Y, \Delta Z$ features, since these states are sampled at a fixed time step.

To remove the velocity information from how the track features were sampled, we re-sample the state of the car with respect to the distance along the path. Denoting the distance between two consecutive waypoints as Δs

$$\Delta s(t) = \sqrt{\Delta X(t)^2 + \Delta Y(t)^2 + \Delta Z(t)^2}, \tag{1}$$

we find the distance traveled along the path as the cumulative sum of Δs along the path

$$s(t) = \sum_{\tau \in (0,t]} \Delta s(\tau). \tag{2}$$

We now construct the full feature matrix $\Phi^{n \times d}$. Each row in Φ is comprised of the vector $\phi(s) \in \mathbb{R}^d$ for a given sample at s. Denote $x(s) \in \mathbb{R}^k$ as the states of the car (from those listed in Table 1) at the distance s along the path. We assume that the car states are identical to the path of the car at the position s, which is valid as long as the car does not lose contact with the ground, which is highly irregular. We additionally assume the sampling frequency of the states is sufficiently high so that we can use a cubic interpolation with respect to the distance s along the path to find a given state from the original data.

Since the current velocity of the car, $V(s)$, depends on the path ahead and behind the current position, we introduce three hyperparameters to build the feature matrix from the path $(x(s))$. Let L_a and L_b denote the look-ahead and look-behind steps, respectively, and let Δs denote the distance between each way point. We add L_b points along the path behind the current position s, each at a distance Δs from the proceeding point. Likewise, L_a points along the path in front of the position s are added, each at a distance Δs from the previous point.

Therefore the feature vector, $\phi(s)$ is built as

$$\phi(s) = \begin{bmatrix} x(s - L_b\Delta s) \\ x(s - (L_b - 1)\Delta s) \\ \vdots \\ x(s - \Delta s) \\ x(s) \\ x(s + \Delta s) \\ \vdots \\ x(s + L_a\Delta s) \end{bmatrix}, \tag{3}$$

and the feature matrix is constructed with

$$\Phi = \begin{bmatrix} \phi(s_1)^T \\ \vdots \\ \phi(s_n)^T, \end{bmatrix} \tag{4}$$

where the distance between each current position and the next is given by the same hyperparameter $\delta s = s_{i+1} - s_i$.

Our goal is to model the speed $V = [V(s_1), \dots, V(s_n)]^T \in \mathbb{R}^n$ as a function of the features in Eq. 4.

2.2 Regression Models

The goal is to predict a continuous quantity, speed; hence regression models, which are designed to predict continuous quantities rather than classify objects, are implemented. First, two base models are constructed to serve as a preliminary comparison to the regression models. The first base model is a zero model that assumes the speed of the car is zero at all points. This model is highly unrealistic as the car must travel at non-zero speed in order to complete laps around the track; however, this model gives an upper bound on the total error. The zero model predicts V as continuously zero speed:

$$[V(s_1), \dots, V(s_n)]^T = [0, \dots, 0]. \tag{5}$$

The second base model constructed is an average model. This model uses knowledge of the true speed of the car along the track, which is not used when constructing regression models later, and assumes the car drives the average speed at all times. This second base model, the average model, has a piece of information that the predictor models constructed don't have: the speed of the car along the courses. This model can be viewed as a more realistic baseline that we would like to surpass with the regression models. The average model is given by

$$[V(s_1), \dots, V(s_n)]^T = [\frac{1}{n}\sum_{i=1}^{n} V(s_i), \dots, \frac{1}{n}\sum_{i=1}^{n} V(s_i)]. \tag{6}$$

The data consists of trials around 33 different tracks. We have split the data into a training and testing set. Cross validation is used to tune hyperparameters for some of the regression techniques. The first 28 tracks were used as the training data and last five tracks were used as the testing data. Separating the training data and testing data means that the model performance is evaluated using unseen data which mimics the real life condition of seeing a new track and not having access the true speed data.

The raw feature matrix is comprised of variables that include information about path geometry prior to the current position and after the current position. The input feature variables used are track width, w, and track curvature, κ. All models are evaluated using mean squared error (MSE) which the is average of the square differences between the predicted speed and actual speed. MSE:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(V(s_i)_{pred} - V(s_i)_{actual})^2 \tag{7}$$

The first regression model constructed is an ordinary least square (OLS) linear regression model. This is a linear model that minimizes the sum of squared error between the predicted speed and actual speed. OLS model:

$$[V(s_1), \dots, V(s_n)]^T = \begin{bmatrix} \phi(s_1)^T \\ \vdots \\ \phi(s_n)^T, \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_f, \end{bmatrix} \tag{8}$$

To solve for the weights, w, in the OLS model:

$$\hat{w} = \arg\min_{w} \frac{1}{n}\sum_{i=1}^{n}(V(s_i)_{pred} - V(s_i)_{actual})^2 \tag{9}$$

The next step taken was to regularize the OLS model using ridge regression. The ridge regression model penalizes high weight values on the assumption that we know, a priori, that the model weights won't be extraordinarily large. Ridge regression uses the same equation to solve for the predicted speeds, but determines weights using:

$$\hat{w} = \arg\min_{w} \|\Phi w - V(s)_{actual}\|^2 + \lambda\|w\|^2 \tag{10}$$

The hyperparameter, λ, is set by sweeping over a range of values (i.e. tuning the hyperparameter) to determine an optimal value that minimizes MSE. Similar to ridge regression, LASSO regression has a penalization term which encourages sparse weights. LASSO weights are determined as follows:

$$\hat{w} = \arg\min_{w} \|\Phi w - V(s)_{actual}\|^2 + \lambda|w| \tag{11}$$

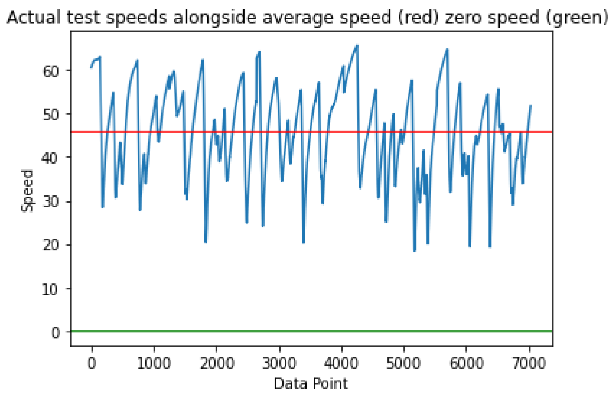
Next, the input data matrix, phi, is augmented using polynomial features of order two, three, and four. Using polynomial features makes it possible for the model to describe more complicated relationships in the data. The augmented matrix is then used as the feature matrix for ordinary least squares linear regression, ridge regression, and LASSO regression.

3 Results

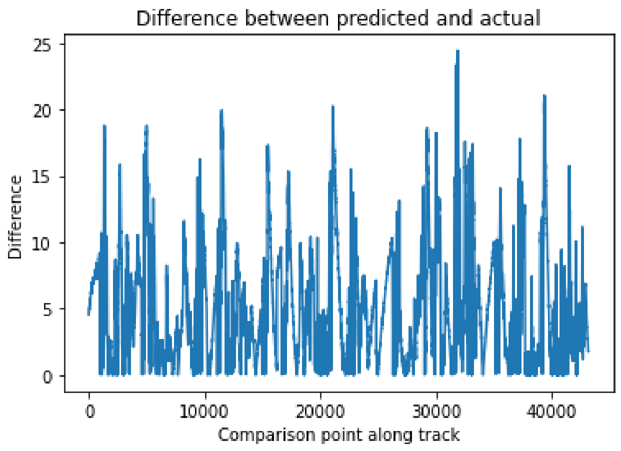
3.1 Regression

Baseline models:

The zeros model has an MSE over 2200 and the average model has an MSE of 96.5. The zeros model is unrealistic and hence 2200 represents an extreme upper bound on the error of modeling this system. In contrast, the average model has information that the true models will not have so it is a more realistic baseline model with a much lower error. The zeros model (green) and average model (red) are graphed against the true speed data (blue) in the figure below left.



(a) Zero Model, Average Model, and True Speed Data.



(b) Ordinary Least Squares Difference between predicted and actual speed.

Figure 2

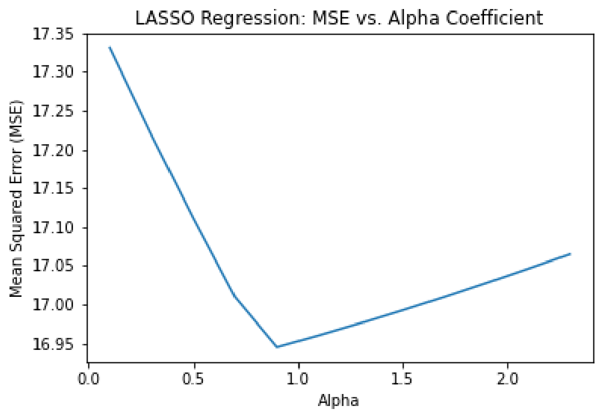


Figure 3: LASSO Hyperparameter search

LASSO Regression:

We do not know if all of the features have a significant influence on the velocity function, so we therefore perform LASSO regression using all possible features:

$$x^T = [w, \kappa, \Theta_X, \Theta_Y, \Theta_Z].$$

The remaining variables in Table 1 are not relavant after the transformation to s coordinates given in Eq. 1 and 2. We perform LASSO, which encourages sparsity in the weight vector, in order to determine the most important features in the regression model. We search through various values of the hyperparameter λ in Eq. 11 of LASSO regression. The results are shown in Fig. 3. Increasing the value of λ encourages more sparsity, as more weight is added to the component of the cost function that is influenced by the magnitude of the weight vector in Eq. 11. The minimum MSE is found at $\lambda = 0.9$, and we performed LASSO regression at this level and found that the significant parameters are only given by

$$x^T = [w, \kappa],$$

and therefore the following sections include only the width of the track and curvature in construction of the feature vectors.

Ordinary least squares:

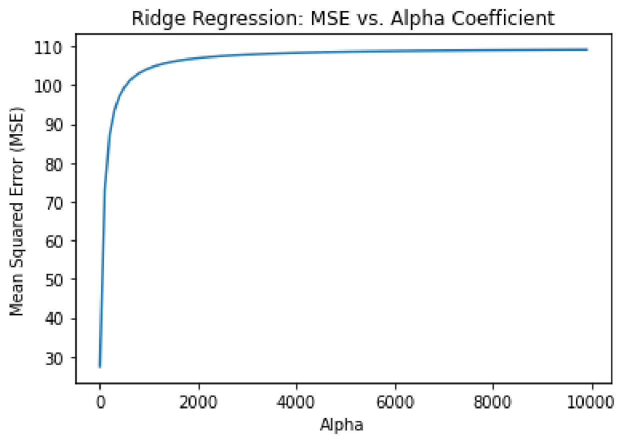
The ordinary least squares (OLS) model has an MSE of 27.5 (for hyperparameters $L_a = 15$, $L_b = 10$, $\Delta s = 3$) which is less than a third the MSE of the average model. Regularizing via ridge regression does not improve performance. Regularizing via LASSO regression also does not improve performance.

Polynomial Features:

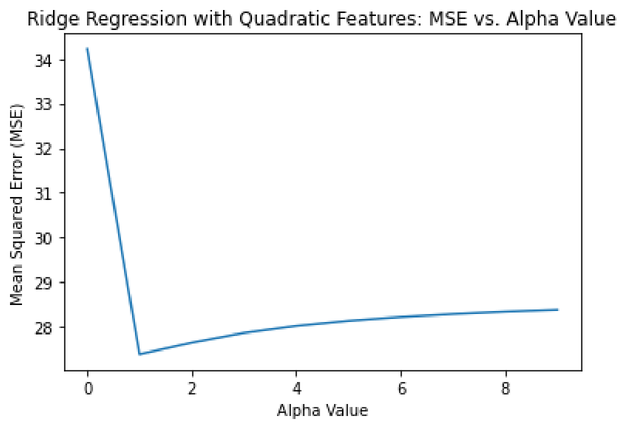
Polynomial degree two features using ordinary least squares regression has an MSE of 34. Due to the computational complexity of running models with polynomial degree features the hyperparameters were changed to minimize the size of the initial feature matrix. This was achieved by setting La and Lb equal to five and Δs=10. In this model, in the figure below right, it can be seen that higher order polynomials perform better.

3.2 Feature matrix hyperparameter search

We first determine the best values for the hyperparameters L_a, L_b, and Δs as shown in Eq. 3. We first vary the values of L_a, L_b, and Δs in Fig. 6 and compare the mean square error (MSE) for regression. For comparison of the hyperparameters, we perform regression using ordniary least squares with second order polynomial features. As shown in Fig. 6, increasing L_a and L_b decreases the MSE, which is expected since more information about the track is given for higher values of L_a and L_b. However, the computational complexity also increases with L_a and L_b, limiting these values. As shown in Fig. 6c, the MSE reaches a minimum at $\Delta s = 17$ m.

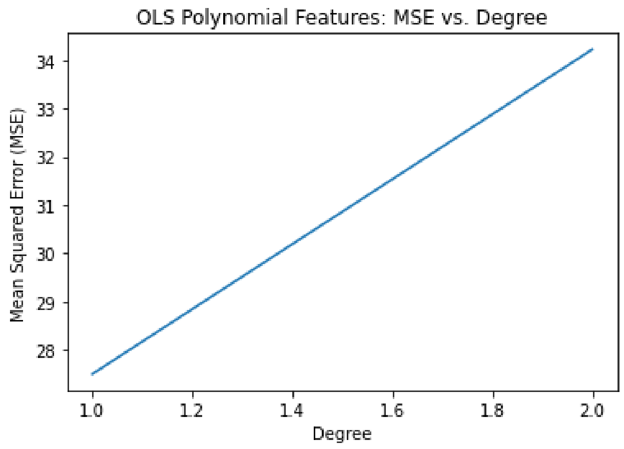


(a) Ridge Regression Model using Raw Feature Data as Input (Degree One).

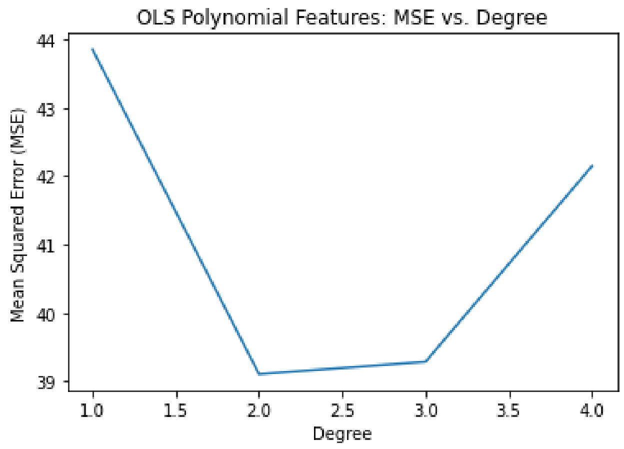


(b) Ridge Regression Model using Polynomial Degree 2 Feature Data as Input.

Figure 4

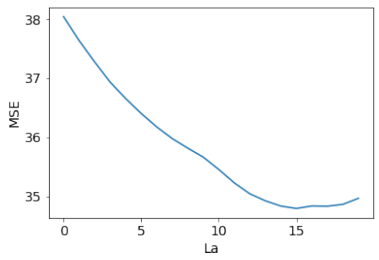


(a) Performance on True Features Degrees One and Two Polynomials.

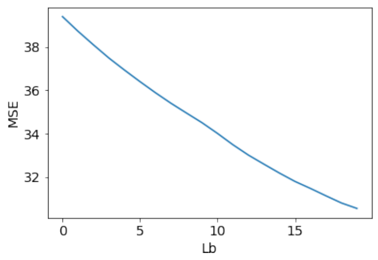


(b) Performance on Simple Features Degrees One, Two, Three and Four Polynomials.

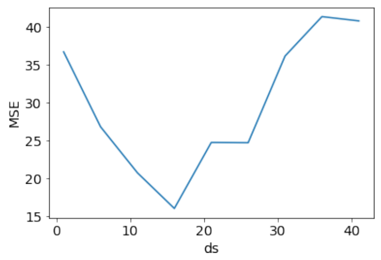
Figure 5



(a) Varied L_a with $L_b = 5$ and $\Delta s = 2$



(b) Varied L_b with $L_a = 5$ and $\Delta s = 2$



(c) Varied Δs with $L_a = L_b = 10$

Figure 6: Effect of varying look-ahead distance, look-behind distance, and distance between way points in feature matrix with polynomial features of degree 2.

To determine the best value for Δs, we assume fixed values of $L_a = L_b = 10$ to balance error and computational complexity. We perform a hyperparameter search by isolating one track of data as a test set, and training the model for varying values of Δs on the remain tracks which serve as the training sets. For each training set, we find the value for Δs which has the least MSE on the test set. The average value across the minimum values for each set is given by $\Delta s = 18.74$.

4 Conclusion

In this work we presented a method to use regression techniques to model the achievable velocity of a race car given the path taken by the car. By building a feature vector which contains path information ahead and behind of the current position of the car, we can predict the current speed using regression. The contribution of this work is an alternate method to determine the velocity of the car without the need for development and identification of physics-based vehicle models. We determine the best hyperparameters for building the feature matrix by performing a folding validation approach. We additionally examine various methods of performing regression in addition to ordinary least squares, including varying the polynomial degree of the features, examining ridge regression, and LASSO regression. We showed that only some of the features were influential on modeling accuracy with LASSO regression, and then determined second order polynomial features were the best at describing the velocity function. Future work could include kernalization to reduce the computational complexity of polynomial features, which would allow for potential use of higher order polynomials or larger values of the look ahead and look behind hyperparameters.

Link to Github

`https://github.com/cwj22/CS289Project`

References

[1] D. Casanova. "On minimum time vehicle manoeuvring: the theoretical optimal lap". en. In: (Nov. 2000). Accepted: 2006-05-25T11:45:53Z Publisher: Cranfield University. URL: `http://dspace.lib.cranfield.ac.uk/handle/1826/1091` (visited on 12/01/2020).

[2] Daniel Patrick Kelly. "Lap time simulation with transient vehicle and tyre dynamics". en. In: (May 2008). Accepted: 2011-02-02T11:17:21Z Publisher: Cranfield University. URL: `http://dspace.lib.cranfield.ac.uk/handle/1826/4791` (visited on 12/01/2020).

[3] Nitin R. Kapania, John Subosits, and J. Christian Gerdes. "A Sequential Two-Step Algorithm for Fast Generation of Vehicle Racing Trajectories". en. In: *Journal of Dynamic Systems, Measurement, and Control* 138.9 (Sept. 2016). ISSN: 0022-0434. DOI: `10.1115/1.4033311`. URL: `https://asmedigitalcollection.asme.org/dynamicsystems/article/138/9/091005/384344/A-Sequential-Two-Step-Algorithm-for-Fast` (visited on 01/31/2020).

[4] Alexander Heilmeier et al. "Minimum curvature trajectory planning and control for an autonomous race car". en. In: *Vehicle System Dynamics* (June 2019). ISSN: 10.1080/00423114.2019.1631455. URL: `https://www.tandfonline.com/doi/abs/10.1080/00423114.2019.1631455` (visited on 10/14/2019).

[5] J. Subosits and J. C. Gerdes. "Autonomous vehicle control for emergency maneuvers: The effect of topography". In: *2015 American Control Conference (ACC)*. ISSN: 2378-5861. July 2015, pp. 1405–1410. DOI: `10.1109/ACC.2015.7170930`.

[6] E. Velenis and P. Tsiotras. "Minimum-Time Travel for a Vehicle with Acceleration Limits: Theoretical Analysis and Receding-Horizon Implementation". en. In: *Journal of Optimization Theory and Applications* 138.2 (Aug. 2008), pp. 275–296. ISSN: 1573-2878. DOI: `10.1007/s10957-008-9381-7`. URL: `https://doi.org/10.1007/s10957-008-9381-7` (visited on 12/01/2020).

[7] Hans B. Pacejka. "Chapter 13 - Outlines of Three Advanced Dynamic Tire Models". en. In: *Tire and Vehicle Dynamics (Third Edition)*. Ed. by Hans B. Pacejka. Oxford: Butterworth-Heinemann, Jan. 2012, pp. 577–591. ISBN: 978-0-08-097016-5. DOI: `10.1016/B978-0-08-097016-5.00013-9`. URL: `http://www.sciencedirect.com/science/article/pii/B9780080970165000139` (visited on 12/01/2020).

[8] Nitin R. Kapania and J. Christian Gerdes. “Learning at the Racetrack: Data-Driven Methods to Improve Racing Performance Over Multiple Laps”. In: *IEEE Transactions on Vehicular Technology* 69.8 (Aug. 2020). Conference Name: IEEE Transactions on Vehicular Technology, pp. 8232–8242. ISSN: 1939-9359. DOI: `10.1109/TVT.2020.2998065`.

[9] Anthony Ingram. *Gran Turismo Sport - exploring its impact on real-world racing with Kazunori Yamauchi*. en. URL: `https://www.evo.co.uk/news/22774/gran-turismo-sport-exploring-its-impact-on-real-world-racing-with-kazunori-yamauchi` (visited on 12/01/2020).