Real-time Model Predictive Control of Autonomous Racecars with Accurate Dynamics

1st Jay Fleischer
University of Pennsylvania
Philadelphia, USA
jayf@seas.upenn.edu

I. Abstract

Autonomous racecars are becoming increasingly common and this paper describes the implementation of a model predictive control algorithm to help the racecar follow the optimal trajectory. This should allow it to reach higher speeds without exceeding the traction limit and reduce its lap time. The algorithm mostly works, but due to time constraints it is still unable to perform as well as hoped.

II. Background

Penn Electric Racing is a club that designs, builds and races custom electric vehicles. Each year we build a new car and REV4 is our latest vehicle, featuring four wheel drive and custom motor controllers for the first time. It doesn't go as fast as many racecars, with a top speed of only 90mph or so, but it makes up for it with its 0-60 time of 2.7s and a large wing that provides enough downforce to go around sharp corners at relatively high speeds.

Fig. 1. REV4

A couple of years ago, a new division of the competition was created for driverless vehicles and Penn Electric is considering competing in future years. An important part of this is creating a control system that allows the vehicle to make it around the track as quickly as possible and the goal of this project is to develop this controller. Beyond its uses for autonomous, the controller will have other applications, such as designing a torque vectoring system that better distributes torque between the four wheels with a person driving, estimating the optimal lap time on a course, and evaluating which potential design changes improve performance enough to be worth implementing.

III. Simulation

A. Framework Overview

1) C++: The goal of this project is to create an algorithm that can adapted to work on an actual racecar in the future and therefore I decided to develop everything in C++, because its fast and suitable for real time applications. Its also a language Penn Electric Racing uses frequently, making it easier to use this project with the rest of our code in the future.

2) Graphics: The simulation includes graphics rendered in real time with OpenGL. The vehicle model is a simplified version of the actual REV4 CAD that still includes over a million triangles. A cone model was created to match the actual cones that are used in the competition and a custom shader was used to overlay the cone textures and handle lighting. A couple other shaders were also used for generating the trajectories and other debug information. Note that time was mainly spent on the graphics because realistic images of cones with motion blur were needed for a machine learning project for another class.[1] Nevertheless, this system proved to be quite convenient for debugging the dynamics and controller.

3) Integration: There are many different integration methods and initially I hoped Euler integration would be sufficient for the project. However, with some quick experiments it became clear that this might not be accurate enough and after some research I concluded that RK4 would be a better option. It requires evaluating the dynamics 4 times per iteration, so it takes 4 times as

long as Euler, but it is a fourth-order integration method, which means it is much more accurate.[2] It's not quite as accurate as the fifth order RK45 commonly used in MATLAB, but it's almost as good and roughly twice as fast. To compare the two, I used a simple model that only had a constant lateral acceleration and Euler integration was off by more than 5m, whereas the RK4 error is barely noticeable.

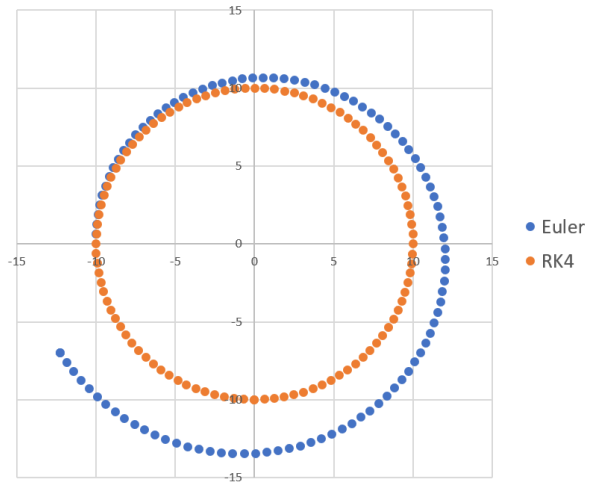

Fig. 2. Integration Method Comparison

B. Dynamics

In order to make the dynamics model accurate, I focused on creating an accurate tire model. The tires respond much more quickly than the rest of the system, because they are powered by a motor controller that runs a control loop at between 32kHz and 64kHz and because the motors have so much torque that they can change speed by a significant amount in milliseconds. A further complication is that the competition prohibits racecars from using more than 80kW, which means that the racecar can accelerate faster at lower speeds. There's also a significant amount of friction in the drivetrain that reduces the actual torque output. In addition, the motor controllers can regenerate energy when to slow the car, but the front motors have less torque, so hydraulic braking is required to get the maximum deceleration. To simplify things a bit, each wheel was assigned its own torque value from -1 to 1 and it was assumed that these were always feasible, leaving power checking for the control algorithm. It was also assumed that the steering wheel could change angles instantaneously to make it easier for the controller.

The wheel angular velocity is not directly simulated, because the motor controllers are so much faster than any higher level controller that simulating it wouldn't be that

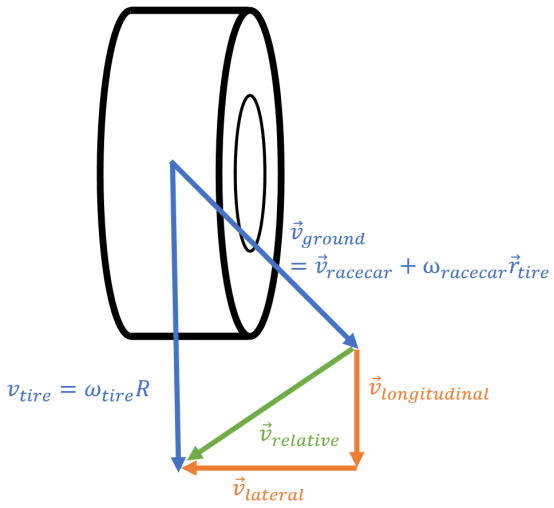

Fig. 3. Tire Velocity Vectors

useful, but would be very computationally expensive. Instead, the tire angular velocity is determined based on the torque request and there's one scaling parameter that was tuned to make it a bit more realistic, because the car would tend to spin out a bit too much when it was one. The maximum force on each tire is calculated based on weight transfer, the aerodynamic downforce and coefficient of friction. The velocity of the ground relative to the wheel is determined based on the racecar's velocity, angular velocity and the tire location on the car. These values are then used to determine the tire force vector as a weighted average of the lateral and longitudinal directions depending on how much torque was requested and the ratio between the lateral and longitudinal velocities.

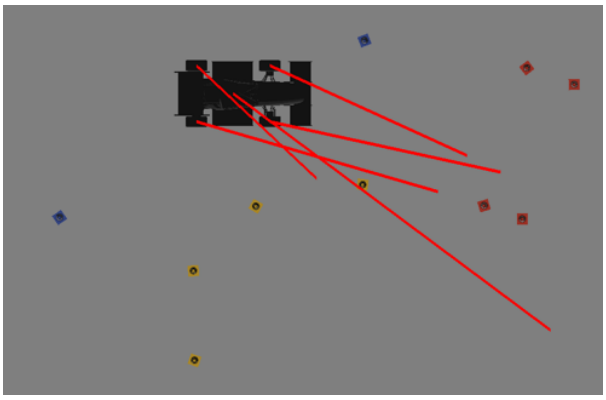

Fig. 4. Tire Forces when Accelerating out of a Turn

The simulation then runs this tire model once for each

wheel and adds together all the forces and moments to determine the acceleration and angular acceleration of the vehicle at each point in time. Keyboard and mouse controls were added to test the code and ensure the behavior seemed reasonable. When going around a circular track, lap times were about 5-10% slower than the actual car, demonstrating that the dynamics are accurate.

C. Paths

The competition specifies that the cones be located 3m apart on all the courses, which means that there's little room for the car to move around in, because it is nearly 2m wide and the cone position estimates will have some some error. Therefore, the project assumes that a course has already been provided. However, it is designed so that the course can be built up as the car maps the environment and everything will work without major modifications. The system supports lines, arcs and nth order splines, where everything is parameterized with a variable that goes from 0 at the start to 1 at the end, because this makes them easier to work with and the parameter can always be scaled such that this is the case. The position coordinates, derivative, second derivative, signed curvature and distance along the path can be calculated for each primitive type in constant time. It also supports finding the closest point on the course to a given point in $O(\log n)$ time using a binary search given a known previous point.

$$k = \frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{(\dot{x}^2 + \dot{y}^2)^{\frac{3}{2}}}$$

Fig. 5. Signed Spline Curvature Equation[3]

One of the challenges with finding a suitable path is that the curvature must be continuous or there will be a point where the racecar must have an infinite angular acceleration and linear acceleration. Therefore paths had to be carefully chosen to ensure this constraint is met. For example, an oval track that was used for much of the project had straight lines on each side and a spline on each end that had polynomials chosen to ensure curvature was zero at both ends. Another common curve used to help with this problem is the Euler spiral, which has linearly increasing curvature. These weren't implemented for this project though because splines were sufficient and it's not easy to find the Euler spirals that end up at a given location given a starting position and direction.[5]

$$x(t) = width \times t^2(3 - 2t)$$

$$y(t) = width \times scale \times t(1 - t)$$

Fig. 6. Spline with 180° turn and Zero Curvature at the End Points

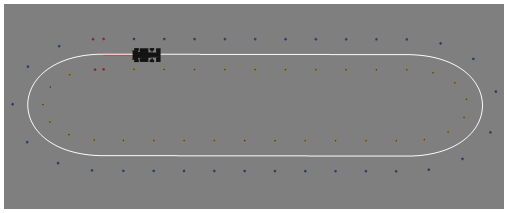

Fig. 7. Circuit with Continuous Curvature

D. Velocity Planning

Rather than defining the target velocity at each point as the derivative of the curve, the velocities are calculated separately to ensure the racecar can go as fast as possible at each point while staying within the traction limit. The algorithm works with a certain dt and begins with a forward pass that assigns a velocity to the position after each time step by assuming the racecar has a certain net acceleration, uses whatever lateral acceleration is necessary and then accelerates forwards as much as possible. There's a safety factor in the maximum acceleration to ensure the racecar has a little margin for error. If it ever hits a point where its going too fast, it instantaneously slows down and continues. This is followed by a backward pass that finds any of these discontinuities and back calculates the speeds that the car must have been going to slow down in time, using the same calculations.

The code also handles loops by going around an extra time and then merging the results between half a lap and one and a half laps in to ensure the velocities match at the point where the ends meet. The results of this algorithm are used to create a lookup table that can be used to find the velocity at any given point in $O(\log n)$ time.

I later found a paper where a similar approach where a forward and backward pass was used, but this was after I had implemented everything.[4]

IV. Control

A. Timing

REV4 has an inertial measurement unit that updates at 400Hz, which means this is the maximum useful controller frequency, even though the motor controllers can actually control current significantly more quickly.

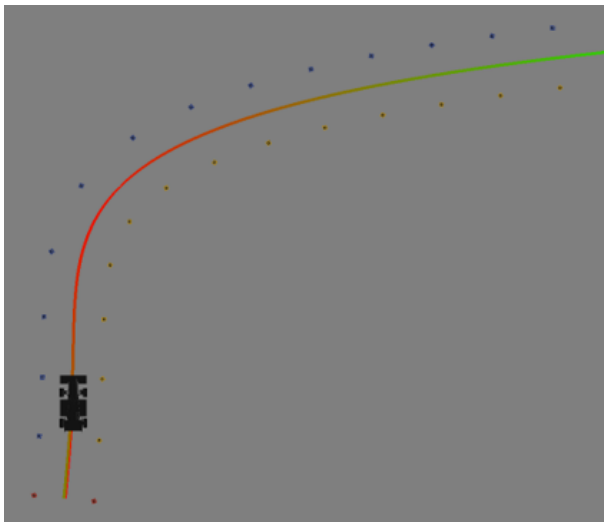

Fig. 8. Maximum Velocities along Path (6m/s - 25m/s)

This was the frequency used for most of the experiments, but in some cases the controller frequency was reduced so it could look farther ahead while still running fast enough. To ensure the dynamics were accurate and be more similar to the real world where you can't perfectly predict the future car positions, the dynamics were simulation at 6kHz.

B. Nominal Controller

Model predictive control needs the linearized dynamics about some nominal trajectory and to get the best results, it should be done with some other controller that works well by itself. This ensures that the linearization is actually a good approximation of the system, because it will be computed near the points of the actual trajectory.

$$\vec{x} = \begin{bmatrix} x \\ z \\ v_x \\ v_z \\ \theta \\ \omega \end{bmatrix}, \dot{\vec{x}} = \begin{bmatrix} v_x \\ v_z \\ a_x \\ a_z \\ \omega \\ \alpha \end{bmatrix}, \vec{u} = \begin{bmatrix} \theta_{steer} \\ \tau_{fl} \\ \tau_{fr} \\ \tau_{rl} \\ \tau_{rr} \end{bmatrix}$$

Fig. 9. State Vector, its Derivative and Control Vector

The nominal controller finds the point on the target trajectory closest to the current location and finds a combination of wheel torques and steering angles that produce a force that will accelerate the vehicle in the direction necessary to reach the desired velocity at the next point along the path. I tried making it also match the target angular velocity, but for some reason it struggled to do this and it just reduced accuracy. This controller worked quite well, in that it would follow the path accurately, but after a few laps it would start to drift away until it ended up pretty far away, because it does not include positional feedback.

$$\begin{aligned} \underset{\vec{u}}{\text{minimize}} \quad & ||\vec{F}(\vec{u}) - \vec{F}_{des}||^2 \\ \text{subject to} \quad & -\frac{\pi}{4} \leq u_1 \leq \frac{\pi}{4} \\ & -1 \leq u_i \leq 1, i \in 2,3,4,5 \end{aligned}$$

Fig. 10. Nominal Command optimization Problem

This optimization problem was solved using gradient descent by calculating the gradient of the cost function with respect to each control input, stepping, clamping the control outputs within the allowed region and repeating. Computing the full dynamics is computationally expensive, but by only recalculating on the tires where the force was affected, all five gradients are computed in roughly the same time it would take to evaluate the dynamics 2.5 times. This algorithm worked well, and typically converged to inputs that would produce a force within a few Newtons of the desired force. Hover, it tended not to keep heading in a good direction, because there wasn't any heading feedback. That meant that after part of a lap, it would reach a point where it was facing in a direction where matching the desired forces was no longer possible and then fly off the course and into a path around it a few meters larger in circumference. To help prevent this, gradient descent was started from the previous control inputs, but with a steering angle calculated so that the average of the wheel angles between the front and rear of the vehicle would match the curvature of the course. This change made a big difference and allowed the car to complete multiple laps before the car would start hitting cones.

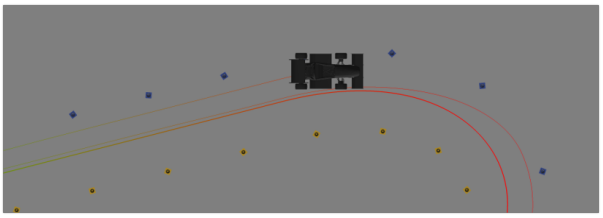

Fig. 11. Nominal Controller vs Target Path (in bold)

C. Dynamics Linearization

Now that we have a controller that mostly follows the path, model predictive control also needs the linearized dynamics at each time step. Again, these derivatives were calculated numerically by determining the new forces if each of the variables were changed. $\frac{\partial \dot{\vec{x}}}{\partial \vec{u}}$ was calculated the same way as before for the force optimizer, but

unfortunately there isn't a more efficient way to do the calculations for $\frac{\partial \vec{x}}{\partial \vec{x}}$ and the dynamics have to be evaluated four times. Another decision was what step size to use when computing the gradient and I still don't have a great answer. If the step is too large or small compared to the controller outputs, the dynamics won't be representative. For example, the opposing force will be large with even a small increase in the lateral velocity, but the opposing force will be the same even with a much bigger velocity change, because friction is nonlinear.

$$\begin{bmatrix} 1 & 0 & dt & 0 & 0 & 0 \\ 0 & 1 & 0 & dt & 0 & 0 \\ 0 & 0 & 1+\frac{da_x}{dv_x}dt & \frac{da_x}{dv_z}dt & \frac{da_x}{d\theta}dt & \frac{da_x}{d\omega}dt \\ 0 & 0 & \frac{da_z}{dv_x}dt & 1+\frac{da_z}{dv_z}dt & \frac{da_z}{d\theta}dt & \frac{da_z}{d\omega}dt \\ 0 & 0 & 0 & 0 & 1 & dt \\ 0 & 0 & \frac{d\alpha}{dv_x}dt & \frac{d\alpha}{dv_z}dt & \frac{d\alpha}{d\theta}dt & 1+\frac{d\alpha}{d\omega}dt \end{bmatrix}$$

Fig. 12. Linearized Dynamics Matrix (A)

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \frac{da_x}{d\theta_{steer}}dt & \frac{da_x}{dt_{fl}}dt & \frac{da_x}{dt_{fr}}dt & \frac{da_x}{dt_{rl}}dt & \frac{da_x}{dt_{rr}}dt \\ \frac{da_z}{d\theta_{steer}}dt & \frac{da_z}{dt_{fl}}dt & \frac{da_z}{dt_{fr}}dt & \frac{da_z}{dt_{rl}}dt & \frac{da_z}{dt_{rr}}dt \\ 0 & 0 & 0 & 0 & 0 \\ \frac{d\alpha}{d\theta_{steer}}dt & \frac{d\alpha}{dt_{fl}}dt & \frac{d\alpha}{dt_{fr}}dt & \frac{d\alpha}{dt_{rl}}dt & \frac{d\alpha}{dt_{rr}}dt \end{bmatrix}$$

Fig. 13. Linearized Control Matrix (B)

D. Model Predictive Control

Model Predictive Control involves optimizing a quadratic program where the error from the nominal trajectory and the control input at each time step are all decision variables and the cost is a function of the error from the desired trajectory.

$$\begin{aligned} \underset{\vec{u_t},\vec{x_t}}{\text{minimize}} \quad & \sum_{t=t_0}^{t_n} (20((x_t + x_{t,err})^2 + (z_t + z_{t,err})^2) \\ & + ((v_{x,t} + v_{x,t,err})^2 + (z_t + z_{t,err})^2) \\ & + 20(\theta_t + \theta_{t,err})^2 + (\omega_t + \omega_{t,err})^2) \\ \text{subject to} \quad & \vec{x}_{t+1} = A_t\vec{x}_t + B_t\vec{u}_t \\ & -\frac{\pi}{4} - u_{t,1,nom} \leq u_{t,1} \leq \frac{\pi}{4} - u_{t,1,nom} \\ & -1 - u_{t,i,nom} \leq u_{t,i} \leq 1 - u_{t,i,nom} \\ & \forall i \in 2,3,4,5 \end{aligned}$$

Fig. 14. Model Predictive Control Optimization Problem

This problem is nearly solved in real-time with an optimization solver called Gurobi, as suggested by Dr. Posa. It was able to solve the quadratic problem over twice and fast and was much easier to work with than the solver I had originally found.

The results of the optimizer are then added to the control inputs generated by the original control inputs used for linearization and are then executed. The final version generates control outputs at 100Hz and looks ahead two seconds, generating a total of 200 control inputs and executing 20 of them. It isn't quite fast enough to run in real time, but with a bit more tuning, making it run fast enough should be feasible.

V. Results

An important part of MPC is predicting the future state of the vehicle and therefore it seemed important to investigate the future predictions. Obviously this works much better in simulation than in the real world, because this doesn't account for model inaccuracies, but I still did an experiment to see how much an effect time delay would have. These were all done with only the nominal controller, which gradually drifts away. The four trials were with the controller running at every time step, calculating outputs for the next 100ms at each timestep, calculating the outputs 100ms in advance at each time step, and calculating 100ms of outputs 100ms ahead. In each delayed case, the future states were predicted using the control inputs that had already been selected. The results show that inaccuracy didn't increase by very much with time delay, which indicates that a 100ms delay of MPC calculating the optimal path in the future should still produce good results. The order isn't quite as you might expect, because the delay with predictions for 100ms did better than the updating predictions trial, but this is likely because of random error from getting lucky and having a better heading at different parts of the course.

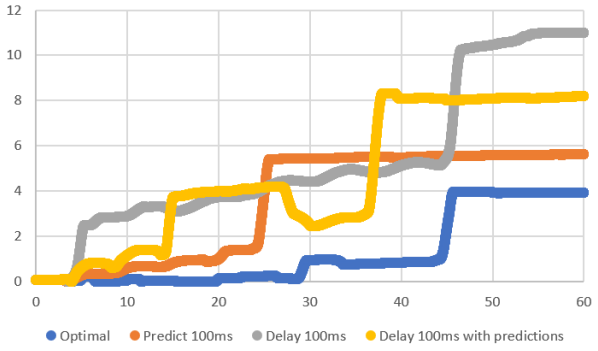

Fig. 15. Time Delay Error Comparison

Model Predictive Control is complex and my implementation of it alone has hundreds of lines of code and

even small mistakes can cause the racecar to perform in very unexpected ways. Unfortunately I ran out of time to fully diagnose all of these, but the controller can still stay on the path, especially at lower speeds. Some of the issues that I was able to fix, include forgetting to divide by mass and multiply by dt in the B matrix, and a misunderstanding about the best way to formulate the MPC problem. With these bugs fixed, the controller works reasonably well When the racecar is started 1m off the track, it successfully drives back onto the course within a few meters. In comparison, the nominal controller would just drive straight, because it doesn't care about positional error and just tries to match the target velocity.

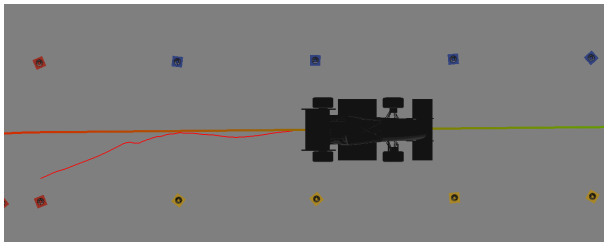

Fig. 16. MPC Correcting Path Error

The controller can still get back on course when started at higher speeds, but it does overshoot a bit. It then proceeds to follow the course well, although a bit slower than desired because velocity isn't weighted as high in the cost function, because otherwise it doesn't stay on course well enough. It overshoots a bit on the first corner, but corrects for this and accurately tracks the path on the opposite side. However, once it has almost completed a lap, the racecar veers to the left and stops before it starts briefly driving backwards. This appears to be an angle wrap around bug, even though the code should account for this, but I haven't had time to finish investigating.

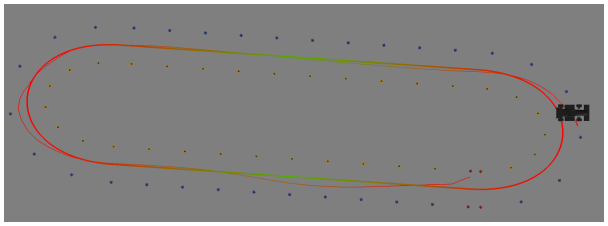

Fig. 17. MPC Stuck Before Completing Lap

VI. Future Work

The Model Predictive Controller I developed works reasonably well, but there is still a lot of room for improvement before we can even consider testing it on the actual racecar. Since the project was so focused on MPC, not enough time was spent on the nominal controller and this should improve the linearized dynamics MPC has to work with, improving performance. In the current implementation there is still a significant safety margin over the actual traction limit to help prevent the racecar from sliding out and even with it closer to the actual limit, MPC doesn't prioritize speed enough and it goes slower than possible. Still, these are relatively minor issues compared to many of the problems that had to be solved to get to this point, leaving me optimistic about the potential of the controller.

There were a few other things that didn't quite fit into the project timeframe, such as optimizing performance more. Also, ideally MPC would run on a separate thread while the rest of the simulation was going to closer match the application in the real world where you can't just slow down time to finish solving the optimization problem. It would also be interesting to experiment with changing the model so the controller can't predict the future behavior as accurately. While there's still much work to be done, the ultimate goal of the project was to create the foundation for a controller that can be used on REV5 in the future and to that end it was a success.

References

[1] J. H. Fleischer and R. V. Hoffelen, "Determining the 3D Locations of Cones in Real Time with Neural Networks for Autonomous Racecars", 19-Dec-2018.

[2] *Runge–Kutta methods*, Wikipedia, 01-Dec-2018. [Online]. Available: https://en.wikipedia.org/wiki/Runge\OT1\textendashKutta_methods. [Accessed: 10-Dec-2018].

[3] *Curvature*, Wikipedia, 25-Nov-2018. [Online]. Available: https://en.wikipedia.org/wiki/Curvature. [Accessed: 10-Dec-2018].

[4] N. R. Kapania, J. Subosits, and J. C. Gerdes, "A Sequential Two-Step Algorithm for Fast Generation of Vehicle Racing Trajectories," *Journal of Dynamic Systems, Measurement, and Control,* vol. 138, no. 9, p. 091005, Feb. 2016.

[5] K. Kritayakirana and J. C. Gerdes, "Autonomous vehicle control at the limits of handling," *International Journal of Vehicle Autonomous Systems,* vol. 10, no. 4, p. 271, 2012.