# *Nonlinear MPC Based Autonomous Racing Vehicle in Real-Time*

*Final project, Autonomous Robots, 2020.*

*Yinon Sinay , 311396576*
*Yarden Ben-Amitai, 312575384*

*abstract:*
This paper reviews the theories and practice of an autonomous racing car implemented with nonlinear model predictive control (MPC) in a real-time environment. We address the basic trade-off between the desired maximum speed and proper road-following meanuevers while presenting the results of our experiment through the optimization process, its challenges and the final result.

## ***Introduction:***

When first getting acquainted with the primary idea of autonomous racing vehicles, there are a variety of different approaches for implementation, such as lane detection and track following through image processing, proportional–integral–derivative (PID) controlled navigation and machine learning based models. Model predictive control (MPC) is an advanced method of process control that is used to control a process while satisfying a set of constraints. It is a multivariable control algorithm that uses: an internal dynamic model of the process, a cost function J over the receding horizon (will be explained later) and finally, an optimization algorithm that minimizes the cost function J using the control input. Nonlinear MPC is characterized by the use of nonlinear system models in the prediction. NMPC is being increasingly applied, with advancements in controller hardware and computational algorithms, to applications with high sampling rates (e.g. the automotive industry)[1], and as an application in aerospace to track optimal terrain-following or avoidance trajectories in real-time[2]. NMPC can accurately predict the driving behavior even in extreme conditions, compared to PID based models, and can produce these results faster than a learning model. We consider these traits advantageous to autonomous racing and thus we constructed our model accordingly.

***Simulation:***

Our project is an extension of Udacity's CarND MPC Project[3] and was tested with Udacity's Self-Driving Car Simulator[4], which is based on the game making engine, Unity, and its purpose is to help simulate autonomous driving for educational purposes, while maintaining an environment as realistic as possible. The simulator's term 2 build allows us to test our autonomous car in a ready made track, specialized for MPC models. What makes this build to be MPC compatible is the data collected by the vehicle's sensors, that constantly update it's x, y coordinates, speed, orientation angle, and path coordinates. All our tests were simulated in the Lake Track premade simulation, because of its similarity to real- world racing track layout.

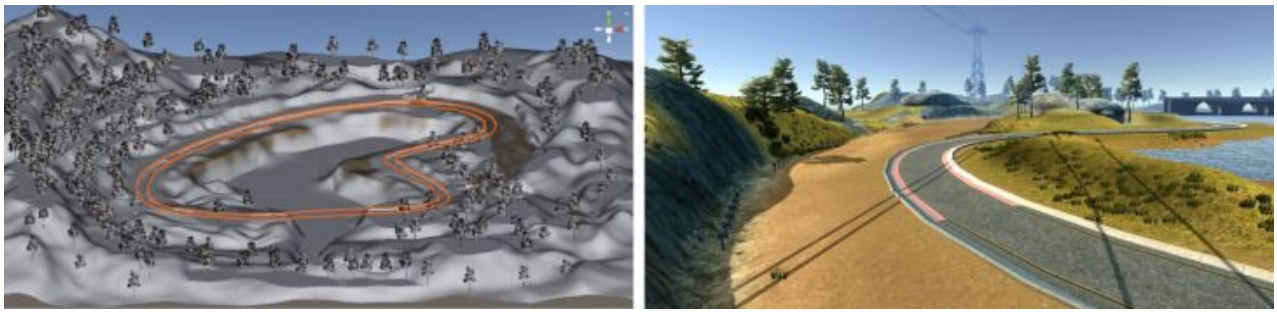

Fig. Udacity's simulation, Lake Track

***Vehicle Model:***

The following paragraphs present the theory and principles of implementing MPC, starting with the kinematic bicycle model, which is a simplified model of a vehicle, where the front and rear wheels are referenced by the point located between them. To analyze the kinematics of the bicycle model, we select a reference point X, Y on the vehicle which is placed at the center of gravity, denoted C. The orientation of the vehicle is described entirely by its yaw angle $(\psi)$. The goal now is to obtain a set of differential equations that describe the motion of the vehicle, meaning, the evolution of the coordinates $x$, $y$ over time $(t)$ and the evolution of the yaw angle $\psi$ over $t$ [5].

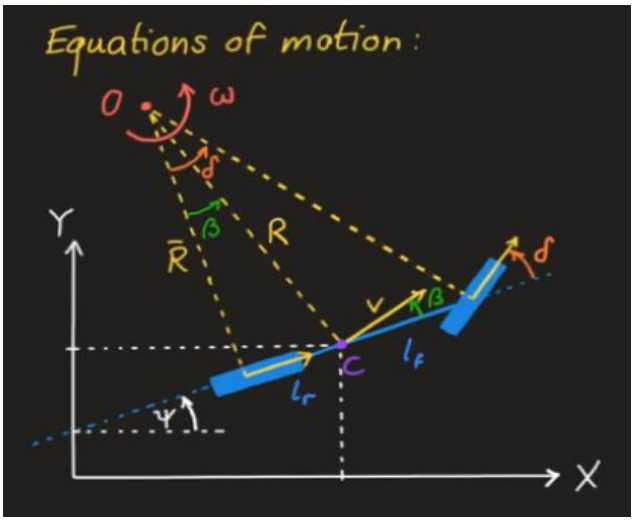


Fig. Kinetic bicycle model

ω – *instantaneous center of rotation*
$C$ – *vehicle reference point*
$l_f$ $l_r$ – *distance from C to front and rear wheels*
$\nu$ – *velocity of vehicle*
$\delta$ – *steering angle of front wheel*
$\beta$ – *sideslip angle of the vehicle*

Fig. Bicycle model parameters

Control variables are the acceleration of the car and the front steering angle, thus the dynamics of the system would be given by:

$$\widehat{X}(t) = \nu \cos(\psi + \beta)$$
$$\widehat{Y}(t) = \nu \sin(\psi + \beta)$$
$$\widehat{\psi}(t) = \frac{\nu}{l_f + l_r} \cos\beta \tan\delta$$
$$\beta = \operatorname{atan}\left(\frac{l_r}{l_f + l_r} \tan\delta\right)$$

***Nonlinear MPC Formulation:***

The trajectory that the car should follow is denoted by a set of waypoints that can be calculated from the data sent over by the vehicle's sensors. We define a number of optimization steps, denoted (N), and the sampling rate or the time in between the timesteps, denoted (dt), which defines how much time elapsed between actuations. The product of these two parameters results in the prediction horizon (T), which is the duration over which future predictions are made. In the case of a self driving vehicle, long horizons increase the computation costs and do not provide more safety because the environment won't stay the same in the near future, and therefore are discouraged. The number of optimization steps determines the size of the control input vector $[\delta_1, \alpha_1, \delta_2, \alpha_2, \ldots \delta_{N-1}, \alpha_{N-1}]$, that once integrated with the vehicle model, results in an optimal trajectory, found by minimizing a cost function. The design of the cost function penalizes the vehicle when deviating off the center of the track ,to some extent, and makes use of the cross track error (CTE) and the orientation error ( $e\psi$ ), specifically, constructing an optimization problem to reduce the cross-track error to a reference path over time $(t \in T)$:

$$
\begin{aligned}
x_{t+1} &= x_t + \nu_t \cos(\psi_t)\, dt \\
y_{t+1} &= y_t + \nu_t \sin(\psi_t)\, dt \\
\psi_{t+1} &= \psi_t + \frac{\nu_t}{l_f}\, \delta_t\, dt \\
\nu_{t+1} &= \nu_t + \alpha_t\, dt \\
cte_{t+1} &= (x_t) - y_t + (\nu_t \sin(e\psi_t)\, dt) \\
e\psi_{t+1} &= \psi_t - \psi_{des,\,t} + (\tfrac{\nu_t}{l_f}\, \delta_t\, dt)
\end{aligned}
$$

Lastly, after applying constraints to $\alpha \in [-1, 1]$, $\delta \in [-25,\ 25]\ rad$, we can define the optimization problem:

$$
J = \sum_{t=1}^{N} cte_t^2 + e\psi_t^2
$$

***Implementation:***

As mentioned above, this project is based on Udacity's CarND MPC Project, and makes use of the initial construct provided in their github. We communicate with the simulator via websocket, forwarding steering and throttle commands, based on data it provides, that consists of telemetry (position, velocity, heading) and a few waypoints for the coming stretch of track.

| Parameter: | Use: | Value: |
|---|---|---|
| *N* | *Number of timestamps* | *8* |
| *dt* | *Sampling rate* | *0.15* |
| *target_v* | *Goal velocity* | *120* |
| *penalize_cte* | *Cost of cte* | *100* |
| *penalize_espi* | *Cost of orientation error* | *200000* |
| *penalize_v_diff_form_t* | *Cost for low velocity* | *3* |

TABLE: Parameters of implementation and their meaning

Once all necessary parameters are obtained, an external library[6] performs the optimization procedure and returns the desired steering angle and throttle, to be passed on to the vehicle. The relevant coordinates of the vehicle movement are saved to 'log.scv' so that they can be analysed at a later time.

***Results:***

The tuning of the parameters was a lengthy process due to the delicate balance between speed, prediction horizon and speedy calculations, in addition to high quality demands, from our part, that must satisfy a smooth drive and turning maneuvers.

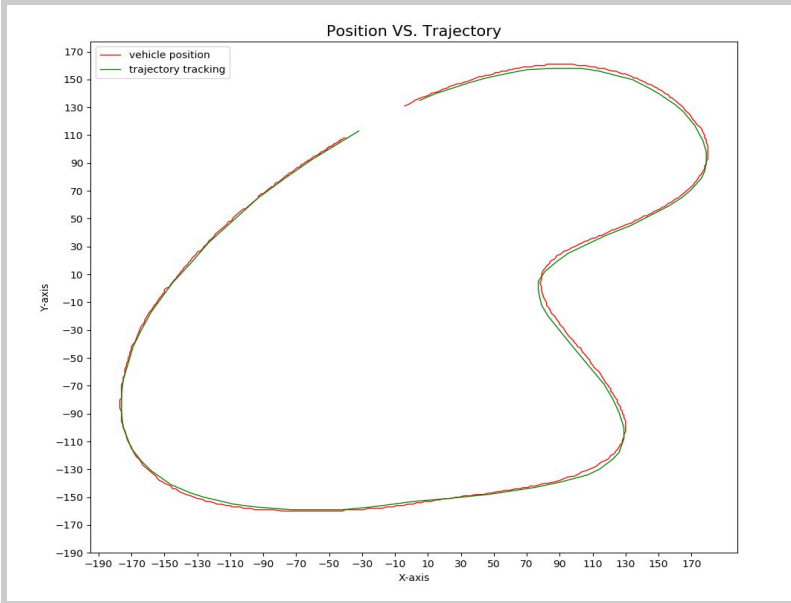


Fig. Position of vehicle vs. actual trajectory

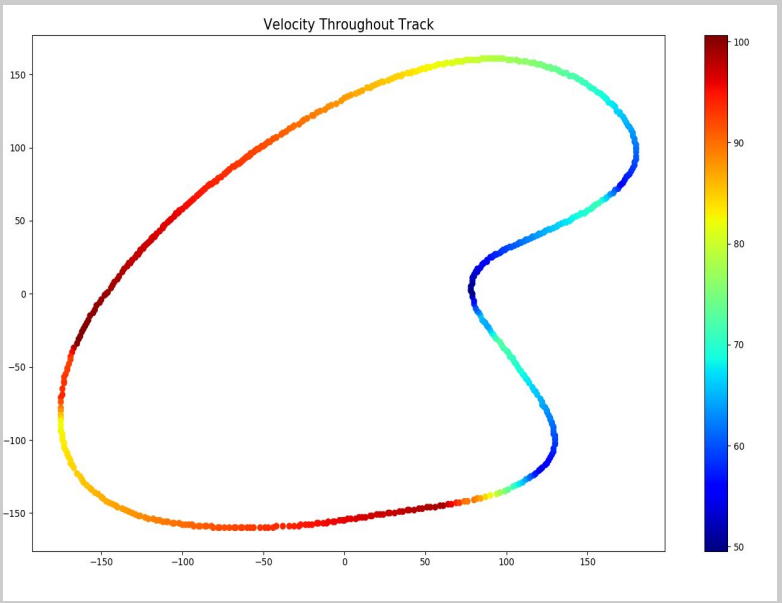


Fig. Velocity of vehicle along the track

The vehicle's path through the track (red) was plotted using x,y coordinates of it's momentarily position, and is compared, in the graph above, to the trajectory (greed) which represents the centerline of the track, presenting us with an informative summary in achieving a decent path following.

Another key component in determining the level of success is the velocity of the vehicle, throughout the track, presented above in a colormap of ranging velocities. It is apparent that the velocity decreases dramatically before making acute turns, while maintaining high speed otherwise. From these results, we propose that a possible improvement to this model would be designing a 'sliding' method, as practiced in real-world races, allowing the vehicle to maintain its high speed, without going off track at acute angles[7]. Although, even without the "sliding" mechanism, the model performs well. The table below contains the data of a 10-rounds length test:

| **Round:** | **Time (s):** | **Best Speed (mph):** | **Worst Speed (mph):** |
|---|---|---|---|

| | | | |
|---|---|---|---|
| *1* | *82.076* | *100.331* | *0* |
| *2* | *32.554* | *100.507* | *51.849* |
| *3* | *32.263* | *100.527* | *52.689* |
| *4* | *32.410* | *100.604* | *49.690* |
| *5* | *32.160* | *100.625* | *52.616* |
| *6* | *32.167* | *100.660* | *53.586* |
| *7* | *64.737* | *100.854* | *52.190* |
| *8* | *32.219* | *100.836* | *51.667* |
| *9* | *32.522* | *100.521* | *49.737* |
| *10* | *32.150* | *100.513* | *51.498* |

TABLE: Test results of 10 rounds, showing the time for each round to be complete, the best and worst velocities reached in each round.

The results shown above indicate that the optimal time and speed remain unchanged, mostly, and while the lowest speed in each round tends to change more frequently, its influence of the resulting time is not as dire as one might think. The results of the first round are brought here for the sake of comparison, and it is clear that when the vehicle starts from velocity of 0, it will perform poorly in the aspect of time. In addition, while inspecting round 7, an abnormality comes to mind, as it posses the highest speed reached in a round, and its lowest speed is far from being the worst, in comparison to the others, yet the time it took round 7 to be complete is the second highest, after round 1. This abnormality brings forth an important example, which is that the consistency of MPC based vehicles is not absolute and may have momentary lapses, which is why it's important to observe the model performing a sufficient amount of time before evaluating its results.

***Conclusion:***

In this project we attempted to construct a nonlinear MPC formulation based model race car, by employing a kinetic bicycle model. Through the process, a great deal of effort was put into searching and testing various simulations, in order to find a relatively simple program that suits autonomous driving and racing criteria. As a result, we have broadened our knowledge in a variety of methods used in experimenting autonomous racing and the many different components that can be added to the code, each one improving little by little the overall results.

***References:***


[1] Stefano Di Cairano (2012). "An Industry Perspective on MPC in Large Volumes Applications: Potential Benefits and Open Challenges", IFAC Proceedings Volumes. 45 (17): 52-59. http://www.sciencedirect.com/science/article/pii/S1474667016314276.

[2] R. Kamyar; E. Taheri (2014). "Aircraft Optimal Terrain/Threat-Based Trajectory Planning and Control". *Journal of Guidance, Control, and Dynamics*. 37 (2): 466–483.

[3] Udacity, CarND-MPC-Project, (2017), GitHub repository, https://github.com/udacity/CarND-MPC-Project

[4] Udacity, Self-Driving Car Simulation, (2017), GitHub repository, https://github.com/udacity/self-driving-car-sim/tree/term2_collection

[5] Prof. Georg Schildbach, University of Luebeck (2020), *Vehicle Dynamics & Control - 05 Kinematic bicycle model.* Available at: https://www.youtube.com/watch?v=HqNdBiej23I

[6] A. Wächter and L. T. Biegler (2006). "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming", *Mathematical Programming* 106(1): 25-57.

[7] You, Changxi & Tsiotras, Panagiotis. (2019). "High-Speed Cornering for Autonomous Off-Road Rally Racing", IEEE Transactions on Control Systems Technology. 10.1109: 1-17.