Faster Than Verstappen: Optimal Racing Using Model Predictive Control

Rijksuniversiteit Groningen – Honours College
Second-Year Research Project

Student: Michal Tešnar, m.tesnar@student.rug.nl

Supervisor: prof. dr. ir. Bart Besselink, b.besselink@rug.nl

June 2023

Abstract

Formula Student is a design competition for university students, which asks them to build a racing car from the ground up and race it on a race track. The competition follows the latest trends in the automotive industry and in recent years a *driverless cup* has been added to the competition, where the vehicles have to race around the track fully autonomously. To obtain optimal performance on the track, robust vehicle control methods are applied. In recent years, the technique called model predictive control (MPC) has been gaining popularity. In this paper, we discuss the formulation of the MPC for use on optimal track drive, as well as implementation choices on a simulated track in MATLAB. We compare three possible approaches to the problem in terms of their computational complexity and performance on a simulated track.

Keywords: model predictive control, Formula Student, continuous constrained optimization, receding horizon control, unicycle model

1 Introduction

Formula Student is a student design competition in which students from a university design and subsequently race a car against students from different universities. This original competition is organized by the *Society of Automotive Engineers* since 1980 and was later adapted in Europe by the *Institution of Mechanical Engineers* [1]. The competition offers a unique opportunity for students to practice engineering skills, the quest is to build a racecar from scratch and race it on the track. This requires students to design their components and manufacture them. The weight and the aerodynamics of the car must be taken into account to achieve the best performance while adhering to strict safety norms. However, as the teams usually rely on sponsors, the competition is not only about the design itself but also about the organization of the working environment. The resulting environment resembles a small company, where strategic choices have to be made to maximize the gain of points in the competition. Apart from the points for the final race car, the students have to

complete several static events, such as design reports and cost and manufacturing reports, forcing students to also professionally document and market their final product.

The competition has started with vehicles with combustion motors, however, throughout the years, the transition towards electric has not left the competition untouched. Currently, there are separate classes for combustion and electric vehicles, however, as sponsors play a big role in the development, there has been a significant move towards electrification. The competition reflects the latest trends in automotive, closely following state-of-the-art inventions. In the season 2017, the *driverless cup* was introduced. From then, competing teams can earn additional points for events their car completes autonomously on the track: acceleration, skidpad event (running in circles to test for the ability to turn), and autocross (running laps on an open track). The last of the three is regarded as the most challenging, as the environment for this event can vary a lot.

This has brought many new engineering challenges into Formula Student, as well as new opportunities for students to learn about technology. The cars are equipped with environment perception technology (cameras, LiDARs), sensors for the perception of the vehicle's state (wheel encoders, ground speed sensors, inertial measurement units) as well as massive processing power to work with the obtained data. Many advanced techniques have to be handled appropriately to steer the car through the track successfully, this includes sensor fusion, vehicle localization, environment mapping, path planning as well as vehicle control. In development, students get to directly work with the theory they get familiar with within the university, giving them a unique opportunity to gain practical experience and become future experts.

As there are limited resources in Formula Student teams, in terms of both time and money, simple and robust techniques are usually preferred to guarantee success. However, the teams are closely connected to research universities which nurtures the ambition to pursue more and more complex approaches. The area of vehicle control is no exception to this trend, with pure-pursuit control [2] being one of the most popular control techniques for path-tracking, the more complex technique called *model predictive control* (MPC) is slowly growing in popularity among the teams [3]. This technique relies on the utilization of a kinematic model to predict the future states of the vehicle, subsequently, a cost function is defined based on how the future states fulfill the goal of the task. The aim is to find the sequence of control commands which minimize the cost function, then we can follow this sequence to reach the optimal state. This technique can be employed in many different contexts [4], and it also applies to optimal racing. In a broad sense, the goal of MPC in optimal racing is to maximize the predicted progress of the vehicle along the path over a certain time horizon.

The MPC method is gaining more and more popularity within the Formula Student community. The best teams use this technique to squeeze the maximum performance out of their machines, for example, the team AMZ Racing from Zurich [5]. The method has also its disadvantages, such as large computational complexity, which depends on the complexity of the model used in the algorithm and the desired prediction horizon. What is more, keeping the car within the track creates a constraint, which depends on a data-intensive representation of the environment. However, the promise of an optimal racing trajectory makes this pursuit worth it for all racing teams.

In this paper, we will explore and explain a simple implementation of MPC in the setting

of Formula Student, using a simulated representation of a track in MATLAB. The motivation is to implement this technique later within the Formula Student team in Groningen, affiliated with Hanze Hogeschool, called Hanze Racing Division[1] (HARD). We will explore the formulation of the MPC problem, discuss implementation choices and compare three possible approaches to the problem in terms of their computational complexity and performance on a simulated track.

2 MPC Formulation

2.1 Overview

The ultimate goal of model predictive control is to decide question of what to do at a certain state of the vehicle to achieve optimal racing performance on the track. The model predictive control technique decides that by evaluating a control command depending on the future states it would generate. Using this approach, the optimal control sequence can be found, such that the model is predicted to advance the furthest along the track. Once this is done, the first part (a_0 and ω_0) of the optimal control command is applied to the current state, from which the next state is obtained. Afterward, the same optimization problem is solved again from the new position, which is an approach called *online receding horizon control*.

In this chapter, we will first define our model and its transition function and constraints. Then we will discuss the track model and its constraints. Lastly, we will formally state the optimization problem.

2.2 Vehicle Model

Let us mathematically describe the vehicle using the unicycle model. We denote the current state (at step k) as

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \\ v_k \end{bmatrix},$$

where x and y denote the position in Cartesian coordinates, θ the orientation and v the velocity of the vehicle in the environment. The model is controlled at each time step k using the control command u_k given as

$$u_k = \begin{bmatrix} a_k \\ \omega_k \end{bmatrix},$$

where a denotes acceleration and ω the change in the steering angle of the model in the given instant. To find the transition to the next state $\mathbf{x}_{k+1}$ of the vehicle, we use a discretized

[1] https://hanzeracingdivision.com/

unicycle model with discretization constant τ. We define the state transition T such that $\mathbf{x}_{k+1} = T(\mathbf{x}_k, u_k)$, which is characterized by the following equations:

$$\begin{aligned} x_{k+1} &= x_k + \tau v_k \cos(\theta_k) \\ y_{k+1} &= y_k + \tau v_k \sin(\theta_k) \\ \theta_{k+1} &= \theta_k + \tau \omega_k \\ v_{k+1} &= v_k + \tau a_k. \end{aligned}$$

Using this transition function T, we can predict the future state of our model given the control command sequence. Now we can use this information to find a trajectory of the model, which maximizes the progress along the track while satisfying the track constraints. This model is visible also in Figure 1.

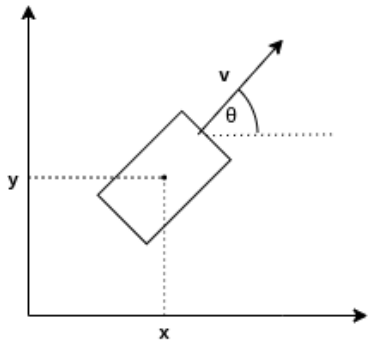


Figure 1: Unicycle model

The model of the car must not violate the laws that the model obeys in reality, which gives rise to various constraints. This means that the a_k and ω_k are constrained at each step. For acceleration, this denotes the maximum possible acceleration and maximum braking, let us call them $a_{\max}$ and $a_{\min}$. Furthermore, the angular velocity must have the absolute value of ω_k smaller than $\omega_{\max}$. Finally, the model must not be faster than some top speed, v_k lower than $v_{\max}$, and the lateral acceleration must not be too high, giving rise to the constraint $v_k \omega_k < l_{\max}$.

2.3 Track Model

Now, let us formally describe the track on which the vehicle races, which will give us a basis for the formulation of our constraints. It is key for us that during all sequences of future states, the vehicle remains on the track. Furthermore, we must be able to define the progress of the car along the track to be able to evaluate the states, and based on that selected the optimal control command.

The track is defined by an ordered set of points in $\mathbb{R}^2$ denotes as $R = \big((x_1, y_1), \dots (x_n, y_n)\big)$, which lie exactly in the middle of the track and are equally spaced. We assume constant width w of the track, so to determine whether the car is on the track, it is sufficient to ensure that there exists a point $(x_i, y_i) \in R$ such that the distance of the model to this point is lower than w. Using the Euclidean distance, we can define the function

$D(\mathbf{x}_k) = \min\{\sqrt{(x_k - x_i)^2 + (y_k - y_i)^2}, (x_i, y_i) \in R\}$. The distance is also visualized in Figure 2. This gives rise to a constraint, as the model must stay within the bounds of the track. We consider state $\mathbf{x}_k$ to be within the track if $D(\mathbf{x}_k) < w$.

To measure the projected progress of the model along the track, we can reuse the ordered set of points R. Assume we want to measure the progress of the model that reaches a point (x_k, y_k), then we can compute the index of the closest point in the ordered set R to the given point (x_k, y_k). This can give a sense of far along a track we are, as obtaining a higher index means more progress[2]. We can now define J similarly to D by looking for the index i which will minimize the distance from the given point: $J(\mathbf{x}_k) = \arg\min_i\{\sqrt{(x_k - x_i)^2 + (y_k - y_i)^2}$ for $(x_i, y_i) \in R\}$. This gives formulation gives us the cost function that we are wishing to maximize. The higher the projected progress along the track, the higher the J.

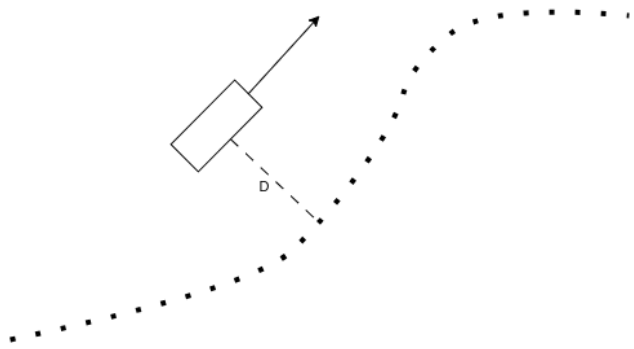


Figure 2: The visualized distance of the vehicle to the path defined by points

2.4 Optimization Problem

Using all the ingredients from the above, we can state the problem of choosing an optimal action given a position on the track as an optimization problem.

Let us define a horizon N of discretization steps of the model which will predict the trajectory of the model. From this, it follows, that our optimization problem is to choose the optimal vector $U = (u_1, u_2, \dots u_N)$ of individual commands u_k applied at each step k to the current state $\mathbf{x}_k$. Applying the whole sequence yields the final state of the model as well as all the intermediate states, based on which we can evaluate the trajectory.

Now we can state the optimization problem. From a given starting state $\mathbf{x}_0$ and for a given time horizon N, we are trying to find the optimal set of control commands $U^* = (u_1, \dots u_N)$ which are applied at each step $\mathbf{x}_k$ yields the state with the highest possible progress along

[2]This introduces problems if we try to cover more than one lap on the track, however, let us ignore that for now.

the track while staying within its bounds. Formally written as:

$$
\begin{aligned}
U^*(\mathbf{x}_0) = &\arg\min_U \; -J(\mathbf{x}_0, U) \\
&\text{subj. to } \mathbf{x}_{k+1} = T(\mathbf{x}_k, u_k), \mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \\ v_k \end{bmatrix}, u_k = \begin{bmatrix} a_k \\ \omega_k \end{bmatrix}, k \in \{1, \dots N\} \\
&\quad D(\mathbf{x}_k) < w, k \in \{1, \dots N\} \\
&\quad a_{\min} \leq a_k \leq a_{\max}, k \in \{1, \dots N\} \\
&\quad |\omega_k| \leq \omega_{\max}, k \in \{1, \dots N\} \\
&\quad v_k \leq v_{\max}, k \in \{1, \dots N\} \\
&\quad |v_k \omega_k| \leq l_{\max}, k \in \{1, \dots N\}
\end{aligned}
$$

As stated before, by obtaining the solution to this optimization problem at any state $\mathbf{x}_0$, we receive the optimal sequence of commands over the horizon of the next N discretized steps. What remains is to apply the first step of the control sequence to the model to obtain the transition to the next step, which we consider as our x_0 for solving our problem in the next step. This process repeats until a lap is completed on the track. This is usually referred to as *receding horizon control* [6].

3 Implementation

As this project is purely application driven – we are trying to autonomously race as optimally as possible – the ultimate proof of the concept is the implementation. The project was implemented in MATLAB [7] with the Optimization Toolbox version 9.4 [8] and Statistics and Machine Learning Toolbox 12.5 [9]. The whole project publicly is available on GitHub [10]. In the subsequent paragraphs, we will walk through the design choices that have been made in the implementation of the project, as well as three possible final solutions.

3.1 Constants

The model was implemented in the code according to the equations described above. This work is meant to be theoretical, as it currently cannot be deployed on a real machine. The previous section features many constants (e.g. maximum speed, maximum acceleration...), however, as those cannot be determined in practice, we have chosen arbitrary values, which are displayed in Table **??**. They are selected such that they reflect realistic behavior on the track.

Name	Symbol	Value
Minimum speed	$v_{\min}$	$1\,\mathrm{ms}^{-1}$
Maximum speed	$v_{\max}$	$100\,\mathrm{ms}^{-1}$
Minimum acceleration (braking)	$a_{\min}$	$-12\,\mathrm{ms}^{-2}$
Maximum acceleration	$a_{\max}$	$12\,\mathrm{ms}^{-2}$
Maximum angular velocity	$\omega_{\max}$	$0.25\pi\,\mathrm{rads}^{-1}$
Maximum lateral acceleration	$l_{\max}$	$50\,\mathrm{mrads}^{-2}$
Maximum steering angle	$s_{\max}$	$0.5\pi\,\mathrm{rad}$

Table 1: Constants used in the simulation

3.2 Creating the environment

3.2.1 Simulation Track

To tackle the task, we need an environment where the receding horizon MPC can be deployed. For that, we first need a track. The tracks used to test our implementation are taken from the Formula Student Driverless Simulator[3] from the hard-coded maps given as examples. These maps resemble well the tracks used in the real competition.

3.2.2 Midpoint line to follow

Then to follow the midpoint path, we need to create the midpoint path through the track. For this, an existing implementation has been used from Matlab Student Lounge[4]. Using this code, we can obtain the midpoint path of the track in the form of an ordered list of points, which we can subsequently follow.

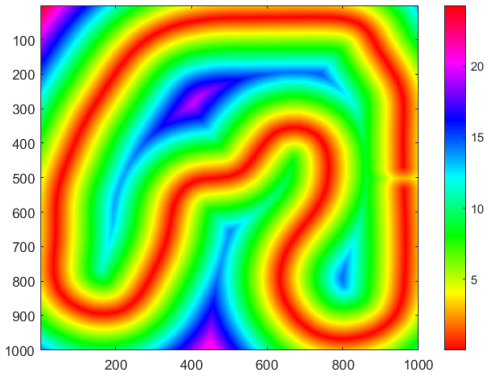
3.2.3 Distance & Progress

The midpoint line is an ordered list of points. To determine the distance of the vehicle to the midpoint of the track, we can search through this array and compute the Euclidean distance of the vehicle to each of its points. Taking the minimum over these calculated distances gives us the distance of the vehicle to the track. However, this approach poses some challenges, because, at each state, we need a linear search through the array, which makes each evaluation of the state computationally challenging. The solution to this problem was to precompute the distances and store them in a big matrix. The results are visible in Figure 3a.

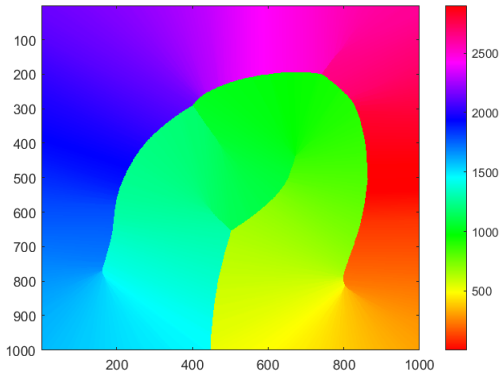
The same goes for the progress along the midpoint line, which is, as established earlier, the index of the point in the list. The same problem applies, and the same solution applies. The resulting grid is visible in Figure 3b.

[3]https://fs-driverless.github.io/Formula-Student-Driverless-Simulator/v2.2.0/

[4]https://blogs.mathworks.com/student-lounge/2022/10/03/path-planning-for-formula-student-driverless-cars-using-delaunay-triangulation/



(a) Precomputed distances with precision of 1000 steps per grid width and grid height, color intensity signifies distance of a point to the midpoint of the track



(b) Precomputed progress with precision of 1000 per grid width and grid height, color intensity signifies the progress of a point along a track

Figure 3: Precomputed distance and progress for all points in the simulation

3.3 Approaches to solving the MPC optimization problem

The optimization problem has been specified earlier, however, the specification does not give a straightforward recipe for solving the problem. In this section, we will discuss three approaches of doing so.

3.3.1 Simple Grid Search

As the spirit of this project was to start simple, we first decided to implement the most trivial searching technique. We can assume that we can only choose a single acceleration value and a single steering angle value over a horizon. That is to say, all control commands are the same ($u_1 = u_2 = \ldots = u_n$). This idea has been taken from [11]. We can consider all possible values of ω and a at a given point in time with respect to the constraints. Then we can sample these parameters at equal spacing, to search the grid of all possible future trajectories. This leads to very simple predicted trajectories, which are shown in Figure 4. On the left we can see all the possible trajectories, our job is to pick the most successful one of them according to the progress along the line, which is shown on the right.

This leads to successful control over the vehicle over the track, however, the control commands are very limited and not flexible. For example, the MPC cannot choose to slow down before a turn and then speed out of it to gain the most progress, as it cannot predict such a command horizon. For this reason, we decided to enhance the search technique in the next two approaches.

3.3.2 Recursive Search with Multiple Horizons

In order to gain more control over the profile of the command over time, we can split up the horizon into smaller parts. Instead of searching the optimal performance given one command, we split the horizon into, for example, four, horizons parts. We take all possible

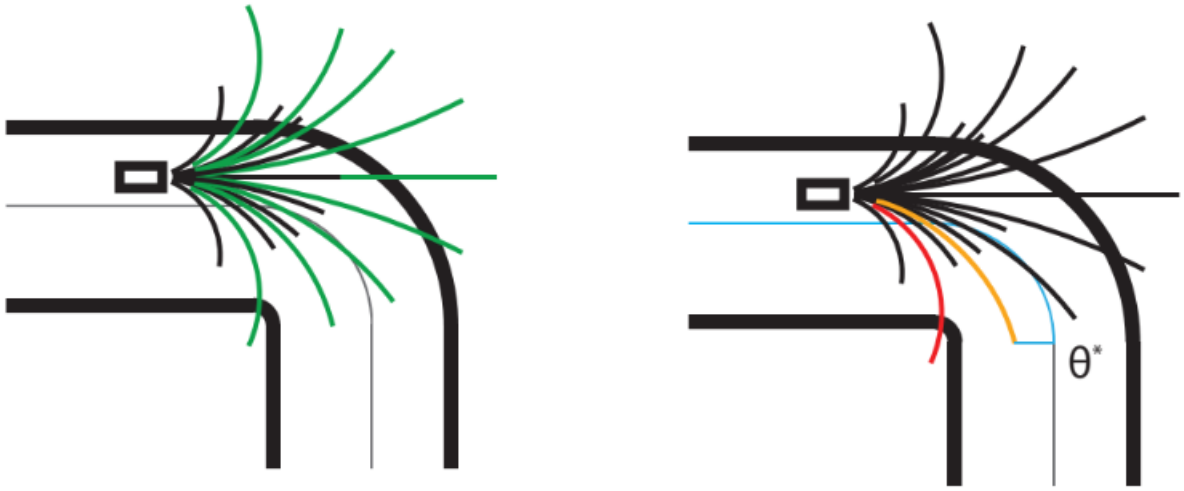


Figure 4: Projected trajectory profiles for the car a constant θ and a over the prediction horizon. Figure taken from [11]

commands on the first horizon and from all that the model is able to reach given those sets of commands, we take different sets of commands over the next sub-horizon. We repeat this recursively from each endpoint, and then we evaluate the points the model was able to reach using our cost function. We pick the command that has led to the best state. Now the model can be more flexible – taking a slower profile at the start, to sacrifice speed for a better position later on, from which it can accelerate. The first command is picked from the sequence that achieves the lowest cost overall. For each of the steps, we once again use the discretized grid of a and ω. However, this search grows exponentially with the size of the grid, as we can see in Figure 7. This means that the computational complexity of this algorithm grows exponentially with the size of the lookahead horizon. A simple solution to decrease the complexity is to each step in the recursive search predict over some smaller horizon and combine those into these recursive steps. In either case, searching into the depths of more than 3, is not computationally feasible in real-time, if we want to consider more than 3 options of ω and a per step.

Finally, due to the challenge of the computational complexity of this approach, we turn to continuous optimization as a solution, which will allow us to be more flexible, while keeping computation time low.

3.3.3 Continuous Constrained Optimization

To put continuous optimization into action, we use the Matlab function fmincon[5]. This function, given an objective function `fun`, initial point `x0`, limits on `x` by upper bound and lower bound (`ub` and `lb`) and constraint function `nonlcon` will try to minimize `fun` while not violating `nonlcon`. We are optimizing over the vector of control variables `x`. Note that we will not concern ourselves with what fmincon does, we will treat it as a black box, which gives us the solution to this problem.

[5]https://nl.mathworks.com/help/optim/ug/fmincon.html

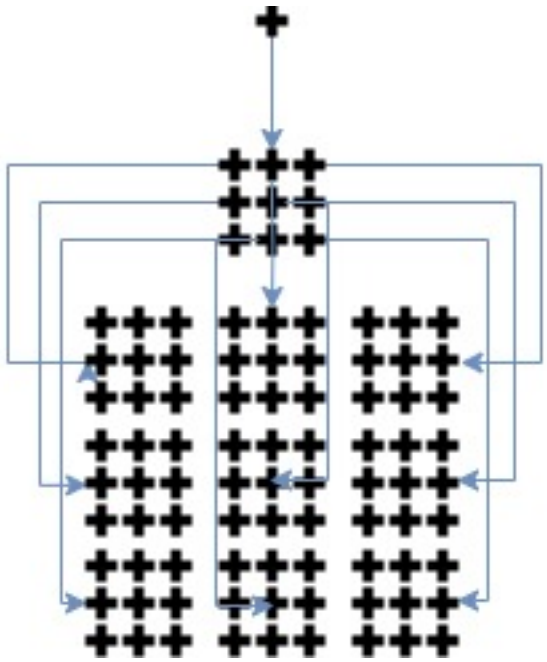


Figure 5: Recursive search on grids of ω and a

In an ideal world, this function would just give us the optimal solution no matter what the complexity of the input, however, in practice non-linear problems such as the one we are dealing with are not easy for the automated solver to solve. It is normal for the solver to fail – if that happens the previous control commands are reused – however, if it fails a lot, we lose control of the vehicle. This means a few adjustments have to be done to the problem in order to make it easier for the solver to solve the optimization.

First of all, we will warm-start the algorithm, which means that we will initialize the vector of control variables u with some plausible values so that the algorithm does not have to optimize over the whole space and can converge faster. To do that, we set small values of acceleration and no angle at the start, because we are starting with a straight lane in front of us. For the next iterations, we just reuse the previous state shifted by one forward and append a copy of the last state at the end. If the previous solution was close to optimal, the new vector should be close to optimal in the next step. Secondly, we manually increase the finite difference of the solver. The solver changes the optimization variables using some update step. By increasing the value, it will optimize more roughly, which might lead to faster convergence, however, too big value might prevent the algorithm from converging at all. By manually testing, we have determined the suitable value to be 0.5. This will give us less optimal solutions if the solver converges, however, it makes sure that it converges more often. Lastly, we reformulate the track constraint. The first possibility is to reformulate it from pass to fail (1 or 0) to a continuous constraint, the cost of which is rising of the cars going further from the track. If we alter the problem like this, the solver can optimize the constraint to find a way of satisfying it. The second possibility is to turn the nonlinear track constraint into a soft constraint – including it in the cost while increasing the cost greatly if the constraint is violated. This has turned out to be the best option, as now the solver must only optimize one variable, as it can ignore nonlinear constraints, and that makes it easier

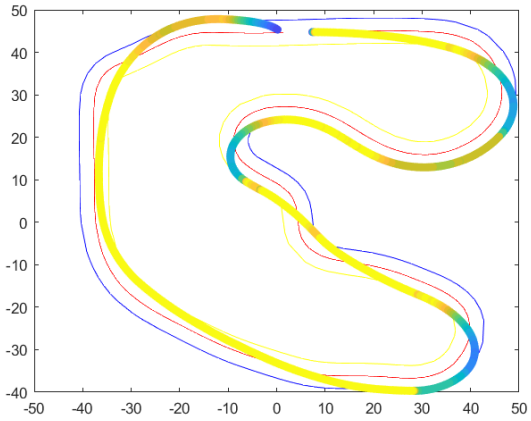


Figure 6: Resulting path from grid search approach with the horizon of 1 second, color intensity indicates speed

for it to solve the optimization problem.

4 Results

4.1 Grid Search

As already stated, this method is not flexible enough to create optimal racing trajectories, because it cannot produce flexible enough trajectories to fit the track. This can be seen in Figure 6, wherein the second turn the car speeds into the turn and then has to rapidly slow down since it cannot turn as fast anymore.

4.2 Recursive Search

This recursive technique does in comparison with simple grid search a little bit better, but far from any optimal trajectory. In Figure 7, we can see that it does not go off the track in the second turn and does not have to slow down very rapidly either. This approach, however, suffers a long execution time, since the recursion over more than 3 levels with grids bigger than 3x3 gets computationally expensive, as the computational complexity grows exponentially.

4.3 Continuous Optimization

Finally, the results from this method show the desired behavior of searching for an optimal racing line. We can now see that in the second turn in Figure 8, the model takes a wider turn while slowing down so that it can go faster later on into the main straight. This is usually called *late apex* in automotive racing slang.

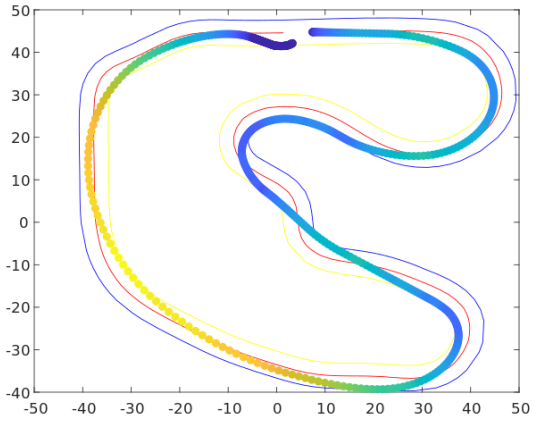


Figure 7: Resulting trajectory from recursive search approach with a horizon of 0.8 seconds (grid size 3x3 with search 4x10 steps), color intensity indicates speed, color intensity indicates speed

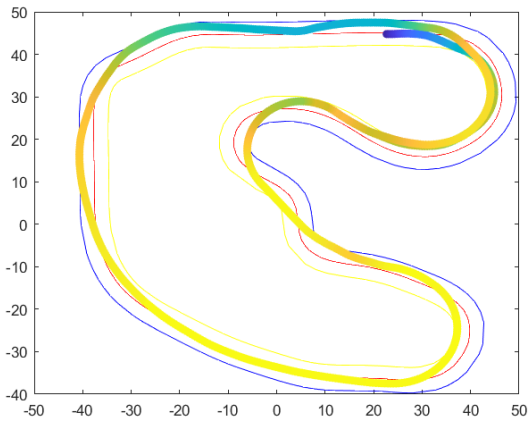


Figure 8: Resulting trajectory from continuous optimization with a horizon of 3 seconds ($\tau = 0.03$, color intensity indicates speed

5 Discussion

Overall, we have been able to analyze and implement a solution to the problem of optimal racing using model predictive control using MATLAB in simulation. We have implemented three different approaches, which result in different behaviors both on the trajectory on the track as well as computational complexity. We can now compare the results.

It is easy to see that the first two approaches, the simple grid search, and recursive search, are not able to provide sufficient flexibility for optimal planning of trajectories. They have served as a good exercise for understanding and implementing receding horizon control, however, they are not fit for real-life applications. This is mainly due to the expensive evaluation of the search techniques, which leads to one solution cycle taking more than one second, rendering the optimal solution useless as the car at high speeds would already find itself in a different situation. Moreover, if we use discretized search, we limit the model to take only a few out of all possible values of control commands that are available. This results in lower flexibility, giving less optimal trajectories. Lastly, the length of the prediction horizon plays a huge role in MPC, as the longer the horizon, the more optimal the behavior is over long periods of time. However, search techniques cannot handle many steps into the feature with appropriate discretization, meaning that it is impossible to predict over long horizons. However, the length of the horizon needs to be chosen carefully. If the complexity of the optimization is too high, the solver might fail all the time.

For all of the reasons stated above, we consider the implementation using the continuous optimization approach as the best and most successful. This approach is certainly worth developing further. To close the paper off, we discuss possible improvements to the implementation. This is to be used as guidelines for further development within the Formula Student team at Hanze Hogeschool.

5.1 Possible Improvements

5.1.1 Model

Firstly, one thing to consider is the choice of the prediction model. The model approximates the dynamics of the real vehicle, however, all models are a simplification of reality. The closer the model is to reality, the better the MPC will perform in a real-life setting. However, there is a trade-off with the complexity of the model, since the more complex, the more it takes to evaluate the future trajectory of the vehicle, making the solving time of the MPC higher if it is feasible at all. The usual choice is a bicycle model, such as in [5], however, here we have chosen the unicycle model for simplicity. All in all, implementing a bicycle model could be a nice addition.

5.1.2 Elliptic Tyre Budget

Secondly, one of the main factors limiting the model of a car is the tyre slip. In our model, we only consider maximum lateral acceleration. In reality, however, slip matters in all directions, because to accelerate as fast as possible maximum grip is needed in the longitudinal direction as well. This can be modeled using the so-called elliptic tire budget, which takes into account the slip in all directions at the same time by considering an ellipse of allowed lateral and

longitudinal forces on the tyre. This is really missing from the model, it could help a lot to make it more realistic.

5.1.3 Empirically Determining the Constants

Most of the constants in the MPC, such as the maximum speed of the vehicle and maximum lateral acceleration, are chosen arbitrarily, that is to say, without any grounding to any particular real vehicle. It would be nice to pick an exact car model and input the measured constants of maximum acceleration and braking, possible maximum lateral acceleration, maximum speed, and maximum turning angle into the MPC to see how it would perform.

5.1.4 Alternative Line Following

Substantial improvement to the project could be done by using a slightly different formulation of the problem, which does not focus on the midpoint line of the track, but follows a different line. One possibility is to follow the path with the least curvature, since turning constraints the speed of the vehicle the most. This can be done by running a global optimization algorithm on the curvature of the line, such as in [12], and then subsequently following that line. Optionally, we can run the MPC for one lap offline on the midpoint line, and let it generate a trajectory, then set up a new MPC to optimize over that line. Both of these techniques could be in principle used with the same problem formulation of the problem.

Furthermore, we could increase the precision of pre-computed distances and progresses, based on the available time and computational resources that have available. Also, we could linearly interpolate between the points on the pre-computed grid to obtain a more accurate representation of the distance and progress at a given point.

5.1.5 Coordinate System

Another improvement to the project would be the use of a different coordinate system. For line following, it is particularly efficient to use the curvilinear coordinate system, which considers each point relative to a line; on that line, a point is picked, and we can have an angle deviation from that line and distance from that point. This is a more popular choice for the line following MPC, see [13].

5.1.6 Faster Solver

To accelerate the solution of the MPC, a faster solver could be used. One frequently used by other FS teams is Embotech Forces Pro[6]. This could allow for faster solution times, which in turn would enable us to specify and solve more complex problems, yielding more accurate predictions.

[6]https://www.embotech.com/products/forcespro/overview/

5.2 Use in Simulation

What is left is to implement this solution in a simulation. For example, we could use the Formula Student Driverless Simulator[7]. This would give us the proof of concept, before deploying it on a real car.

Acknowledgements

Big thanks to Jiří Kosinka for the good tips, and quick and kind responses to emails, especially for the tip for precomputation of distances and storing them in a grid.

References

[1] I. Talmi, O. Hazzan, and R. Katz, "Intrinsic motivation and 21st-century skills in an undergraduate engineering project: The formula student project." *Higher Education Studies*, vol. 8, no. 4, pp. 46–58, 2018.

[2] M. Samuel, M. Hussein, and M. B. Mohamad, "A review of some pure-pursuit based path tracking techniques for control of autonomous vehicle," *International Journal of Computer Applications*, vol. 135, no. 1, pp. 35–38, 2016.

[3] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer science & business media, 2013.

[4] K. Holkar and L. M. Waghmare, "An overview of model predictive control," *International Journal of control and automation*, vol. 3, no. 4, pp. 47–63, 2010.

[5] J. Kabzan, M. I. Valls, V. J. Reijgwart, H. F. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan *et al.*, "Amz driverless: The full autonomous racing system," *Journal of Field Robotics*, vol. 37, no. 7, pp. 1267–1294, 2020.

[6] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[7] MathWorks, "Matlab version: 9.13.0 (r2022b)," Natick, Massachusetts, United States, 2022. [Online]. Available: https://www.mathworks.com

[8] ——, "Optimization toolbox version: 9.4 (r2022b)," Natick, Massachusetts, United States, 2022. [Online]. Available: https://www.mathworks.com

[9] ——, "Statistics and machine learning toolbox 12.5 the mathworks inc. (r2022b)," Natick, Massachusetts, United States, 2022. [Online]. Available: https://www.mathworks.com

[10] M. Tesnar, "Racing MPC," Jun. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.8003066

[7] https://fs-driverless.github.io/Formula-Student-Driverless-Simulator/v2.2.0/

[11] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.

[12] Y. Xiong *et al.*, "Racing line optimization," Ph.D. dissertation, Massachusetts Institute of Technology, 2010.

[13] A. Liniger and L. Van Gool, "Safe motion planning for autonomous driving using an adversarial road model," *arXiv preprint arXiv:2005.07691*, 2020.