KU LEUVEN

FACULTY OF ENGINEERING TECHNOLOGY

CAMPUS GROUP T LEUVEN

Boundary Estimation and Model Predictive Contouring Control

Implementation for Formula Student Race Car

Darina ABAFFYOVÁ

Andrew GORDON

Supervisor(s):

Add titles!!!

Gert Van Loock

Panagiotis Patrinos

Co-supervisor(s): Yuri Cauwerts

Formula Electric Belgium

Master Thesis submitted to obtain the degree of Master of Science in Engineering Technology: Electronics Engineering

Academic Year 2019 - 2020

Abstract

The regular abstract is a short summary that also needs to be uploaded to KULoket. It is written in the main language of your thesis text. Due to technical limitations on KULoket, this part may not exceed 3500 characters.

Contents

List of Figures

1 INTRODUCTION

The main goal of this thesis is to develop a robust, on-board software controller, which will be able to steer a Formula Student electric race car even at the limits of its operation. This goal will be achieved in two stages, Boundary Estimation and Model Predictive Contouring Control. Put differently, the eventual thesis result should be capable of both path planning and path tracking. The path planning algorithm is responsible for finding a feasible reference path that describes the shape of the track and its track boundaries, and the path tracking algorithm directs the car outputs in order to achieve following of the reference path while traversing the track at the fastest rate possible, taking into account the dynamic possibilities of the car.

1.1 Problem Description

This thesis involves developing a component in the new autonomous racing software package intended to be run on the *Umicore Eclipse*, the Formula Electric Belgium (FEB) electric power-train race car that is being converted for driverless racing. The car is custom-produced with an engine output power of 145kW and a 0-100 km/h time of 2.6 seconds. The overall goal of the team is the prototype development of a combined software and hardware package capable of safely operating the car to race a track.

The goal of the this component of the complete autonomous software package is to be able to process sensor data about the car's current state and the cones indicating the track's boundary, and signal the appropriate outputs to efficiently race the track. This involves problems such as developing a robust map of the track in the form of a centerline path describing the track and an understanding of the track boundaries, solving an optimization problem to achieve that the race car follows a racing line based on the centerline reference, given its dynamic properties.

Specific challenges directing the design choices of this thesis are the achievable driving speeds and the need for the software package to operate on the car without previous information about the track layout. This presents a variation on the challenges associated with the development of autonomous driving software, where the focus is commonly on object detection and collision avoidance. For the autonomous operation of a racing car on an isolated track, the focus moves instead, after the required level of robustness is achieved, onto computational efficiency to allow the reliable achievement of the minimum possible lap time, ideally approaching the dynamic limitations of the car. Accurate track mapping, path following and boundary detection is crucial, for both the safety of spectators and the car itself. The Umicore Eclipse maximum racing speed of 32.5m/s and Formula Student rules dictating a minimum track width of 3m, make the risks of an insufficient frequency of output adjustment self-evident.

The input data to the system originates in a number of other software and hardware packages located on the car. Cone location and mapping is done using LiDARs employing a Simultaneous Location and Mapping (SLAM) algorithm, in conjunction with convolutional neural network colour

classification of the cones. This allows the identification of the absolute position of the boundary-indicating cones, plus semi-robust classification data of the cone colour, as different colour cones are used to indicate the left and right track boundaries. Absolute car location data is provided by an on-board inertial measurement unit (IMU), along with correlation with readings from sensors around the car. This provides the positional and dynamic data necessary for the system, which is communicated via ROS middleware libraries.

As this thesis will act as the foundation for extensive development over following years by the FEB team, a significant focus is on achieving design goals with an architecture that is well-documented and extensible. There must be sufficient logging of data and an in-depth visualisation of each stage of processing between the receipt of inputs and the delivery of desired outputs. A sufficiently low loop processing time is crucial, and compromises between processing efficiency and driving accuracy must be clearly justified.

The final result of this thesis is the simulated driving of the car around a complete prepared track. While traversing through the track, during each iteration, the developed system must simultaneously map the boundaries and the centreline, used as a reference line, and solve the optimisation problem to pursue a racing line based on the reference. This must be fully visualised and have a sufficiently low loop time.

1.1.1 Adaptation of thesis goals due to Coronavirus protective measures

This master's thesis was done over the period of 2019-2020, with the majority of the practical working time in February-July 2020. This coincided with the COVID-19 pandemic, which prevented on-site work at FEB, stalling all car development and preventing real-life testing before the due date of the thesis. All testing had to be done purely in simulation with little warning or time to prepare a simulation environment, making it considerably more difficult to guarantee the robustness of the end solution.

Additionally, it proved impossible to gather the data required about dynamic properties of the car in order to develop the car models necessary for the MPCC component. Simplified models had to be used as proofs of concept, that did not require this information.

The focus of the thesis shifted during development, to better suit the adjusted needs of the FEB team, to a more comprehensive visualisation and focusing on only running on a simulated track. This thesis result will then act as the first step for continued development of the autonomous system for FEB on into future years. All future Chapters have been adjusted to reflect this thesis outlook, and where appropriate more detailed notes have been added to better describe the impact on a specific development.

1.1.2 Literature Review

Boundary Estimation

Boundary estimation and path planning are research topics under constant development that contribute to fields including robotics, autonomous vehicle operation and video game development(Abd Algfoor et al., 2015). The evolution of new techniques has led to a wide array of approaches to the

problem of developing a path through an unknown track, including more classical, though still very much in development, approaches like convex optimization and discrete path-finding algorithms to newer approaches like genetic algorithms. No single technique has proven itself to be clearly advantageous so analysis of all and comparison of their relative strengths and weaknesses with the requirements of a project is necessary.

A key factor for the relevance of different approaches to this project was the choice of a controller that balanced maximal traversal speed in a given time with minimizing contouring error. This required the development of a reference path that described the centreline of the track rather than finding an optimal racing line, which invalidated the advantages of some techniques, as will be described further.

One of the most common approaches are the application of graph search patterns such as Dijkstra, Best-first Search, A* and D* as generic path optimisation algorithms (Robotin et al., 2010) whose general procedure involve creating nodes for each possible traversal point with a pointer indicating its parent node and operators to find out possible successor nodes. At each expansion of a node to find its successors, a check is done to find if one of them is the goal node, at which point the path to the target is found by stepping through the parent pointers to the first node.

An example of this technique being applied practically is the use of a Funnel Search algorithm to search between adjacent triangles on a triangulation graph projected onto a polygonal world description (Demyen and Buro, 2006). The ability of the triangulation to represent any polygonal environment while significantly reducing the search space typically to 2n-2 operations for n triangles makes it both flexible and efficient and highly interesting for this application.

The convex optimization approach, as implemented by Bodart (2018) shows the finding an optimal path via the solving of a non-linear optimization problem with additional control constraints where appropriate, then attempting to solve the equations. He finds that considerable accuracy and dynamic modelling complexity is possible with this approach, but for our purposes the significant issue is the processing time necessary. His study recommended a technique that required a calculation time of 11 hours and 15 minutes on a quad-core desktop processor. Considering that the Formula Student competitions stipulate that no electronic information prior to the car being placed on the track to drive are possible, this technique would be difficult to practically apply without a highly simplified track path.

Genetic algorithms are a process by which algorithms are progressively improved with extensive computation, as the best performing algorithms according to a fitness function are chosen to act as the basis for the next set of child algorithms. In this way, they are driven to improvement by mixing and matching parent algorithms till the result is optimal. An example of an application of genetic algorithms to path planning include using connected Bezier curves with each gene defining a small portion of the racing line (Botta et al., 2012) and each successive iteration adjusting those control points and analysing the resulting algorithm performance for improvement. Given sufficient training time and well-designed constraints, it is possible for genetic algorithms to notably outperform graph search methods with the performance of their paths (Cardamone et al., 2010). The cost for this, similar to convex optimization, is time. Processing times in the cited papers to reach sufficiently effective algorithms required often reached multiple hours, which would be entirely unsuitable for this task.

Specific to Formula student, a well-regarded example is the AMZ Formula student driverless project (Kabzan et al., 2019b), which chose to focus on a graph search solution, similarly seeming to rea-

son that the computational load of other approaches was not compatible with the Formula Student competition.

Model Predictive Contouring Control

Use of Model Predictive Control (MPC), also known as Receding Horizon Control (RHC), emerged in the late seventies mainly as a process control algorithm, and has developed substantially from then (Camacho and Alba, 2013). Due to the technological evolution over the last few decades, with considerably faster processing units, and with development of several techniques to improve the method, this robust algorithm is nowadays further used in applications ranging from thermal management of fuel cells (Zhang et al., 2020), through economy and global warming (Bréchet et al., 2014), to multi-agent systems control (Shanbi et al., 2012).

Model Predictive Contouring Control (MPCC) is a widely used approach in the area of autonomous driving. Fredlund and Sulejmanovic (2017) use MPCC with the main goal of driving with minimal error from the reference path, with additional interest in minimization of the traversal time. The problem solved here is convex and linearised, which provides shorter computation time, but compromises the accuracy of the solution. Due to the use of the kinematic bicycle model, the experimental vehicle could only be driven at low velocities.

Furthermore, use of MPCC can be found not only in autonomous *driving* control, but also in driverless *racing* applications (e.g. Zeilinger et al., 2017; Kabzan et al., 2019a; Curinga, 2017). In this case, the main goal is to drive the vehicle through usually an unknown track as fast as possible, hence an optimal racing line should be followed. This can be done either by calculation of optimal racing line and its tight following (Liniger et al., 2015, Chapter III.A), or by specifying the objective function of the optimization problem such that the main focus is on fast driving rather than minimization of the contouring error (Liniger et al., 2015, Chapter III.B). The latter approach is the one used in this thesis.

While racing, the vehicle needs to be driven at its limits of handling, which allows investigating of behaviour which is difficult to experiment with in regular driving conditions. Kabzan et al. (2019a) replace the highly non-linear, complex and impractical vehicle model by a simpler model improved using machine learning tools. This provides the ability of the car to continuously learn and improve the vehicle and tyre model, depending on the current driving and environmental conditions and hence more accurate predictions in the MPCC calculations are possible, without increasing the computational time, i.e. more aggressive driving is allowed by the learning-based controller.

In order to compute accurate predictions and hence obtain optimal control, it is essential to choose an appropriate vehicle model, which provides an acceptable trade-off between computational simplicity for real-time implementation and accuracy for high-performance control of the autonomous race car. Most commonly, bicycle model is used to approximate the behaviour of the racing car. An interesting method is proposed in (Kabzan et al., 2019b), where a kinematic bicycle model is combined with the dynamic one. This is advantageous due to the decreasing accuracy of the kinematic model with increasing velocity and vice versa for the dynamic model.

As the dynamic bicycle model incorporates the effect of the contact between the tyres and the driving surface, these forces also need to be modelled. They play a crucial role when it comes to the safety of the vehicle (Bhoraskar and Sakthivel, 2017). For this purpose, the well-known empirical Pacejka tyre model (Pacejka, 2012), also known as the Magic Formula, is commonly

used (Novi et al., 2019; Yakub et al., 2016; Curinga, 2017; Kabzan et al., 2019b).

To efficiently solve, within a few milliseconds, the embedded nonconvex optimization problem, similarly to Mattingley et al. (2010), where convex optimization code generator is discussed, Optimization Engine (OpEn) code generator (Sopasakis et al., 2020) is used in this thesis. The problem, defined by the cost function and the constraints, is specified in Python and passed to OpEn, which generates a Rust implementation of the solver, which is further interfaced with the Boundary Estimation algorithm using C++. More details regarding OpEn can be found in Chapter **??**.

1.2 Paper Outline

In the following Chapters, details on design and implementation of the algorithms are discussed. In Chapter 2, the design of the two algorithms is described, including the underlying concepts. Chapter 3 explains the details of the implementation, which are then evaluated in Chapter 4. Chapter 5 contains the discussion of the obtained results and explains the possibilities of further work, and Chapter 6 concludes the paper.

2 DESIGN

2.1 Boundary Estimation

The Boundary Estimation component of the thesis acts as the software interface between the stereo-camera processing and Simultaneous Location and Mapping (SLAM) outputs and the Model Predictive Contouring Control (MPCC) input. Receiving absolute cone positions and car location data, the goal is to provide robust centreline calculation and boundary checking to recognise if the car has crossed boundary conditions that would require an emergency stop state for safety.

The rules indicated as key for the development of the track mapping system are those of the Formula Student Germany handbook [stated in DE 6.3.1 FSG Competition handbook]:

- The track is marked with cones.
- The left borders of the track are marked with small blue cones.
- The right borders of the track are marked with small yellow cones.
- The maximum distance between two cones in driving direction is 5m. In corners the distance between the cones is smaller for a better indication.

Additionally [stated in FS Rules handbook 2020 D 8.1.1]

- The minimum track width is 3 m.

Additional design goals for the Boundary Estimation system that influenced design decisions were as follows:

- Minimising reliance on classification, as stereo-camera colour classification of blue (left boundary) and yellow (right boundary) cannot be guaranteed.
- Mapping time kept to minimum possible, as the Umicore Eclipse has a top speed of approximately 117km/h or 32.5 m/s. Given the minimum track width of 3m, the worst-case scenario for the car violating boundary conditions would be a travel of 1.5m which would at top speed require 0.05 seconds. While these conditions would be highly unlikely in practice, they acted as an indicator for the targeted loop refresh rate of 40Hz in order to ensure that there is opportunity for the car to predict the upcoming out-of-bounds state and adjust.

The system itself can be seen as a data processing pathway containing the following sections:

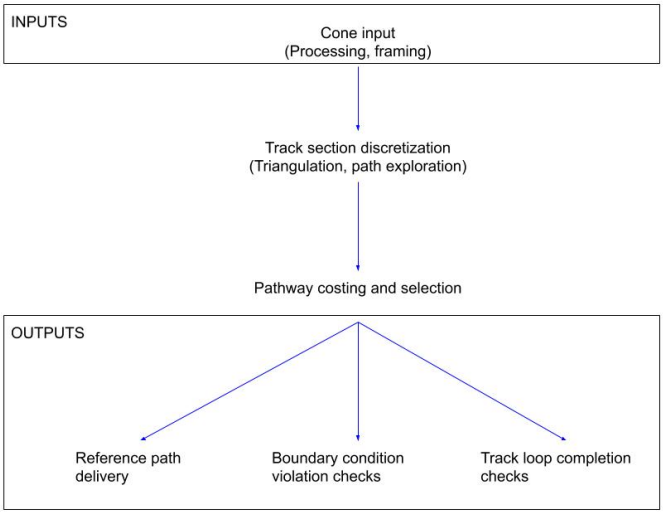


Figure 2.1: Data pathway of Boundary Estimation software component

Using this outputted data, the following services must be provided:

1. centreline delivery to MPCC component.
2. Boundary condition violation checks.
3. Track loop completion checks (at which point the centreline calculation can be deactivated).

2.1.1 Cone input

Cone data is received from the data pipeline of the autonomous car in the format:

Double X position, Double Y position, String cone colour

Upon receipt, first it must be confirmed that the cone has not been previously received. Due both to double imprecision and the details of the SLAM algorithm, it is possible that as the car moves, the same cones will be indicated as being in slightly different absolute locations. If the cone is found to have been received before, the location of the existing cone object should be updated. Otherwise, a new cone object should be created and stored in existing data.

Accurate cone locations can only be guaranteed by the vision elements of the car within certain ranges, so mapping of the track should only be performed on cone locations that can be accurately located. In this way, a framing of connected track sections is performed, with each frame containing all viable cone locations inside a single system refresh loop.

Within that frame, other actions can be performed to improve the performance of the following processing steps, such as attempting to classify cones as left or right boundary in cases where the stereo cameras have not been able to do that with a sufficient margin of accuracy. While the system should not be reliant on cone boundary classifications, if a sufficient number of cones can

be successfully classified, then additional tests and less computationally-intensive search patterns can be used during path exploration.

2.1.2 Track section discretization and exploration

Triangulation

The area bound by the framed cones should then be discretized, to allow for efficient path-finding algorithms to explore the continuous space. This process is described by Abd Algfoor et al. (2015) as ”a fundamental component...in the fields of GPS, video games, robotics, logistics, and crowd simulation”. They continue to specify that the two typical techniques for discretization of continuous spaces are recognised to be:

- Skeletonization, when a topology is formed across the traversable space by mapping a set of line segments to the continuous environment. Typically, these line segments then indicate the traversable paths between discrete points, producing visibility or waypoint graphs.
- Cell decomposition, where the continuous environment of the space is decomposed into a set of cells, each of which encloses an area of obstruction-less space where travel within the cell can be done without path planning.

In the case of this application, applications derived from robotics have more parallels in that, unlike purely virtual applications, real-life terrain properties define the artificial terrain map. As the points of the terrain map are defined by the boundary cones, the skeleton that would be formed between those points would not be a network of viable paths.

Instead, cell decomposition is a more logical application of available discretization techniques, dividing the track area into a set of convex polygons with their line segments drawn between cone locations. By considering the traversable area to be inside each of these cells and the boundary cones to always be the outermost points of the track, we can define the traversable area of a particular polygon to always be either wholly within the boundary conditions or wholly without.

A well-recognised mathematical technique for finding the line segments connecting the points is Constrained Delaunay Triangulation (CDT). The abstraction of an environment via CDT and the efficient finding of a path has been been present in academia for some time (Demyen and Buro, 2006), and the low memory requirements and speed of calculation are a good fit for the technical requirements of the project. This has been shown in other, well-recognised Formula Student autonomous driving projects where CDT was also used. (Kabzan et al., 2019b)

CDT is the process of forming a triangular mesh from unstructured sample points. It is similar to standard Delaunay triangulation, but constraints are added in the form of the addition of specific line segments into the triangulation to produce results that are more optimal for certain applications. In contrast, the standard variant only searches for line segments that meet the Delaunay criterion:

1. No triangle may be formed of a set of three collinear points - that is, a triangle which would result in a circumcircle (the circle that can be drawn connecting all three points of the triangle) of infinite radius.

2. All triangles in the mesh must produce circumcircles that do not contain any line segments other than those that make up the triangle itself.

As this application requires the formation of line segments along the boundaries of the track that delineate the boundary conditions, it is a clear case where CDT is the more appropriate variant.

Pathway Exploration

With a framed area discretized via CDT, the centreline (the path that describes the track) must then be found via the exploration of all possible paths that cross the given area and, where the judgement can be made, do not cross the boundary.

As the points that form the triangles are the boundary cones, travel between the points of the triangles is clearly undesirable. Instead, the line segments of the triangles act as the traversal points and, as can be seen on Figure 2.2, the midpoints of those line segments act as logical waypoints to minimise the complexity and computational time of the path exploration algorithm. By reducing the travel points to the midpoints of each triangle line segments, the traversal of each triangle is thus reduced to a tree structure. Within each triangle, it is possible to form a path from the origin waypoint through the triangle's area to the waypoint at the centre of either of its two opposing line segments. More broadly, as each vertex can be a member of up to two triangles, each waypoint can have up to four possible destination waypoints.

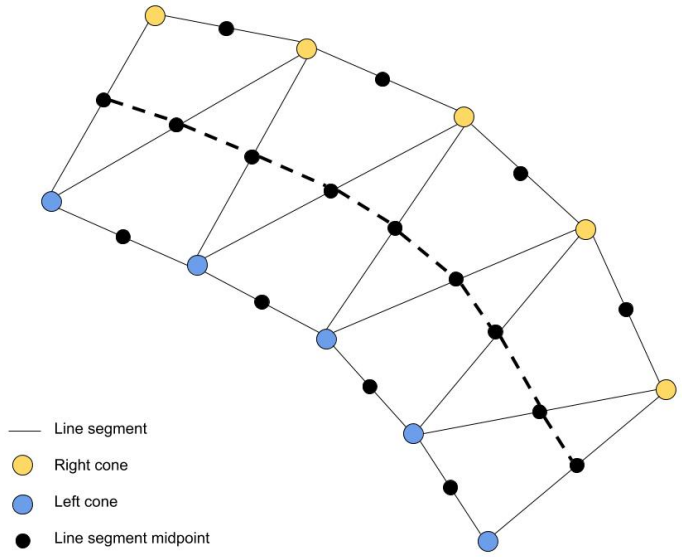


Figure 2.2: Typical but simple triangulation example with available centreline

In order to form the path tree, a root waypoint should be provided. This root should be the origin point of the car when the system is first activated, and henceforth the latest point along the centreline. From this point, waypoints can be added to the tree structure with indicators to their parent waypoint and indicators available to direct at child waypoints. Once all midpoints inside a section have been added to the tree, the tree can be searched to find all viable paths from the root node to a variable final length, with the length indicating the number of waypoints in the path.

2.1.3 Pathway costing and selection

The paths that have been collected must cover all possible routes within the framed track area, and a robust process for selecting the path that best describes the track centreline must be applied. This process must lead to the selection of a path that has the following properties:

- Does not cross boundary conditions, i.e. it does not exit the track.
- Describes the centreline as accurately as possible from the beginning of the track section to the end. Detours and divergence from the centre of the track should be avoided where possible, and minimised where not.
- Avoids loops within the track or repeated movement onto the same waypoint, which could lead to paths that the car is physically incapable of following or that could disrupt the proper operation of the MPCC component.
- Travels through the complete track portion rather than stopping early. Failure would lead to an incomplete track centreline connection from one track section to another, potentially leading to incorrect behaviour.

In order to do this, left and right boundary classifications of cones can be taken advantage of when available, but should not be relied upon. As such, all cost parameters should have both classified and unclassified variants where applicable, with the classified elements weighted according to the reported accuracy of the cone classifications.

The centreline, once selected as the pathway with the lowest cost of all those tested, is then added as a series of discrete points to the reference coordinates that have been selected from previous track sections. Segment-by-segment, the centreline is built until the track is completed and the track mapping process is deactivated.

2.1.4 Outputs

Reference path delivery

As the MPCC component updates, it dynamically determines a required distance of upcoming track according to current car conditions, requesting it in the form of a number of points and a distance between those points. At each of those points, the following must be provided:

1. A coordinate on the centreline.
2. Two boundary coordinates for both the left and right constraint of the track, describing the track width.
3. Slopes at both the left and right constraints of the track describing the shape of the boundary at their respective points.

As the centreline is held as a list of discrete coordinates whose distance from one another is determined by the cone positions and the resulting positions of the triangulation line segments,

interpolation between those points is necessary in order to provide the necessary number of points over the required length of the track. Matching interpolated points must then be calculated to describe the boundary constraints. Additionally, checks must be performed in order to ensure the desired length of the track is available. As the MPCC component has no awareness of how much of the track has currently been processed, which will be determined by physical limitations of sensor performance and cones present in the vision cone of the car in its current direction, the best possible length of the track should be returned in cases where the desired length cannot be provided.

Boundary condition violation

At every refresh of the overall autonomous driving system, new IMU data will be received and the updated car position should immediately be checked against all currently known boundary information. In cases where the car is found to have broken the boundary conditions, this must trigger an immediate emergency stop and car shutdown to ensure the safety of spectators and prevent damage to the car or other property.

This check should have the following properties:

- Efficient calculation to minimise reaction time to the car breaking boundary constraints.
- Robust to cases of an incomplete track map or sensor failure, ensuring that the car triggers an emergency stop even in cases where track mapping has failed and there is no track data in the near vicinity of the car's current position.
- Does not excessively limit the traversable track area within the track boundaries, as this would restrict the driving performance of the car while it is still operating correctly.
- Recognises the physical dimensions of the car, as the car position is considered to be a point mass for most calculations.

Track loop completion

The track finish line is indicated with large orange cones [stated in DE 6.3.1 FSG Competition handbook] but these should not be relied upon for judging completion of the track map, as they are not currently recognised by the stereovision component of the car's vision system, and cone classification is not guaranteed to be reliable.

A sufficiently robust indicator of track completion should satisfy the following requirements:

- Final centreline coordinate is within a minimum distance of the first centreline coordinate. Given the worst case scenario of the cones being 5m apart, the maximum distance between reference coordinates should be 7.5m.
- It should be possible to project multiple points in a linear path between the final centreline coordinate and the first coordinate without breaking boundary constraints. This is necessary to ensure that the centreline provided to the MPCC component remains valid.

2.2 Model Predictive Contouring Control

Model Predictive Control (MPC) is a robust process control method where the next control input is determined by solving an optimal control problem, subject to system constraints, over a finite time horizon. At each time step, when a control input should be given to the plant, this optimization problem is solved, i.e. a series of control inputs is computed. To solve this optimization problem, the current state of the plant and its future states, that would be achieved when applying the sequence of control inputs, are used. In order to predict the future states, a model of the plant is needed. Only the first value of the computed control sequence is applied to the real system and it is solved again on the next time step.

This is represented in the block diagram in Figure 2.3. A reference state $\mathbf{r}_j$ is given as an input to the controller, together with the measured current state of the plant $\mathbf{y}_j$. The controller then minimizes a cost function, taking into account the constraints posed to the system. These constraints can be limits on the actuators, e.g. on how fast the steering angle can be changed, or physical limitations of the states of the plant, such as the maximal speed of the vehicle.

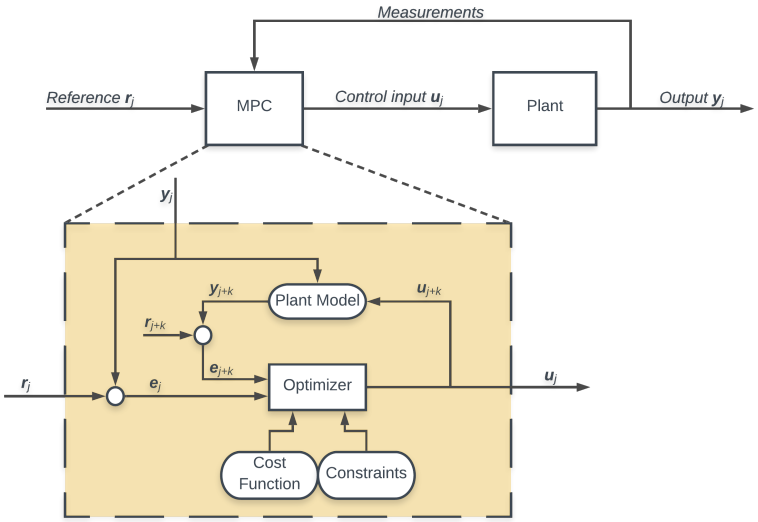


Figure 2.3: MPC Block Diagram

Inside the MPC, the future states $\mathbf{y}_{j+k}$, with $k = 1...N$, are predicted for N steps, the so-called *prediction horizon*, using the model of the plant. In this way, the optimal control inputs can be calculated such that the plant reaches the given reference state. This reference can remain constant over the prediction horizon ($\mathbf{r}_{j+k} = \mathbf{r}_j$ for $k = 1...N$) or it can be a list of reference states different for each step of the prediction. Only the first calculated control input $\mathbf{u}_j$ is given to the plant and the problem is solved again on the next time step, keeping the prediction horizon constant. Figure 2.4 shows this prediction/optimization step, where one of the possible solutions considered by the optimizer is plotted. The control inputs are calculated for N next steps such that the predicted output reaches the desired reference value.

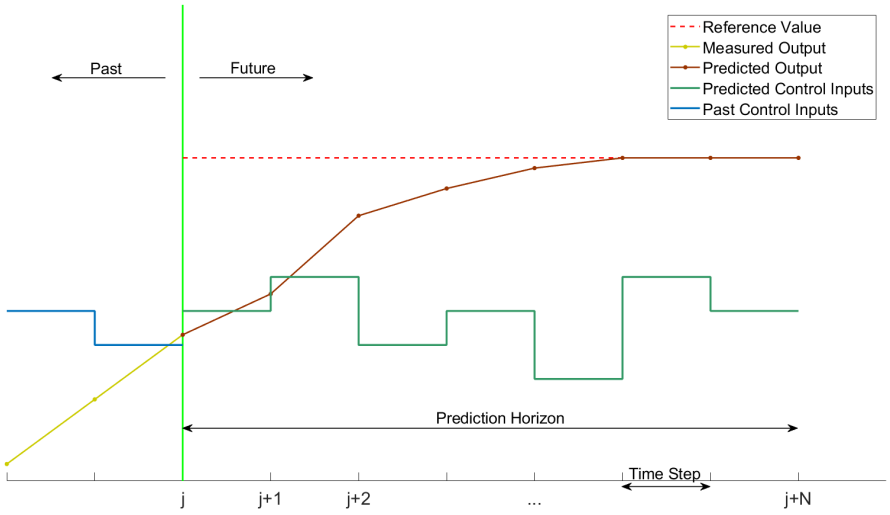


Figure 2.4: MPC Prediction Graph

Model Predictive Contouring Control (MPCC) is a control method which uses the MPC algorithm to solve contouring problems. Contouring is simply the act of following a certain line, or a contour.

A *vehicle model*, i.e. a set of equations describing the behaviour of the car; a number of *system constraints*, which pose the limits to the optimization problem based on physical limitations; and a *cost function*, which reflects the trade-off between set objectives, e.g. between contouring accuracy and traversal speed, are the main building blocks of the MPCC algorithm used in this thesis. The solver is generated using the open source *Optimization Engine* code generator. These are discussed in detail in the following sections.

2.2.1 Vehicle Model

Bicycle model is the vehicle model frequently used for the purposes of a driving MPC controller and a number of papers is dedicated to comparison between the kinematic and the dynamic version of this model (e.g. Kong et al., 2015; Polack et al., 2017). In general, in the bicycle model, the two front and the two rear wheels are lumped together into two single wheels placed in the centre of the front and the rear axle, as shown on Figures 2.5 and 2.6. The symmetry of the appearing forces is consistent with the lateral geometrical symmetry of most cars (Verschueren, 2014).

In order to be able to drive the vehicle at its operational limits, where highly nonlinear behaviour takes place, it is crucial to obtain accurate predictions of the states of the vehicle, hence an appropriate vehicle model must be used. Additionally, the complexity of the model has to be considered so that the optimization problem can be solved efficiently at every time step. In general, simpler vehicle models are favoured (Verschueren, 2014), therefore the bicycle model is used also in this thesis. The kinematic and dynamic bicycle model are discussed in this section, keeping the complexity and accuracy in mind.

Kinematic Bicycle Model

The model is visualised in Figure 2.5. The equation of motion (Kong et al., 2015) is given by

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v\cos(\psi+\beta) \\ v\sin(\psi+\beta) \\ \frac{v}{l_r}\sin(\beta) \\ a \end{bmatrix}, \tag{2.1}$$

where

$$\beta = \tan^{-1}\left(\frac{l_r}{l_f+l_r}\tan(\delta)\right) \tag{2.2}$$

is the slip angle at the centre of gravity, (x,y) are the coordinates of the centre of gravity in an inertial reference frame (X,Y), ψ the heading angle and v the velocity of the vehicle. l_f and l_r are the distances from the centre of gravity of the vehicle to the front and rear axles, respectively. The control inputs are the front wheel's steering angle δ, and the acceleration of the vehicle a.

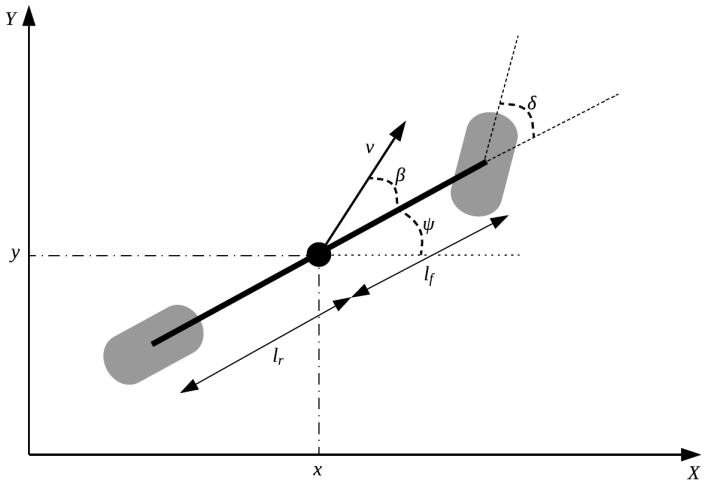


Figure 2.5: Kinematic Bicycle Model

The kinematic model is simpler than the dynamic one (discussed below), as it assumes no slip and is based only on the geometry and the speed of the vehicle (Polack et al., 2017). This makes the system identification easier, only two parameters are needed l_f and l_r, and also porting of a controller using this model to another vehicle is simpler (Kong et al., 2015). However, as the slip and tyre forces acting on the vehicle are neglected, this model cannot be used at high velocities, as it would model the vehicle behaviour inaccurately.

Dynamic Bicycle Model

The model is visualised in Figure 2.6. The equation of motion (Liniger et al., 2015) is given by

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \cos\psi - v_y \sin\psi \\ v_x \sin\psi + v_y \cos\psi \\ \omega \\ \frac{1}{m}\left(F_x - F_{f,y}\sin\delta + m v_y r\right) \\ \frac{1}{m}\left(F_{r,y} + F_{f,y}\cos\delta - m v_x r\right) \\ \frac{1}{I_z}\left(F_{f,y} l_f \cos\delta - F_{r,y} l_r\right) \end{bmatrix}, \tag{2.3}$$

where (x, y) are the coordinates of the centre of gravity in an inertial reference frame (X, Y), ψ the heading angle, v_x and v_y the velocities along the longitudinal and lateral axes of the vehicle, respectively, and ω is the yaw rate. l_f and l_r are the distances from the centre of gravity of the vehicle to the front and rear axles, respectively. The lateral forces $F_{i,y}$, with $i \in \{f, r\}$ represent the interaction between the track surface and the front and rear tyre, respectively, and the longitudinal force F_x acting on the vehicle depends on the applied driver command D (Kabzan et al., 2019b). The lateral forces can be modelled using one of the many tyre models, with the Magic Formula and its adaptations being the most popular. The control inputs for this model are the driver command D, i.e. the PWM duty cycle of the electric motor, and the steering angle δ.

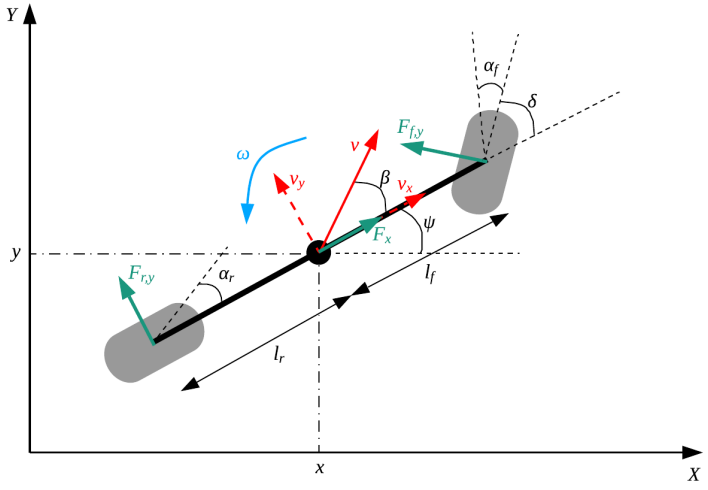


Figure 2.6: Dynamic Bicycle Model

Compared to the kinematic model, the dynamic model is more complex and computationally more expensive. It models the vehicle behaviour accurately at high speeds, but is ill-defined at low ones (Kabzan et al., 2019b).

Chosen Vehicle Model

For the purposes of this thesis, the kinematic bicycle model is used. The reason for this is that due to the coronavirus pandemic, it was not possible to obtain the empirical parameters for the dynamic model. The main implication of using the kinematic model is that the slip forces acting on the tyres are not accounted for, hence the model is accurate only at lower speeds at which the Umicore Eclipse will be driven. This is sufficient for the goal of the thesis, which is to provide a starting point for the future implementation of a driverless student formula.

2.2.2 Model Predictive Controller

Constraints

A number of physical constraints is imposed on the vehicle. These need to be considered in the predictions, to avoid infeasible solutions of the optimization problem, which is fully stated in a later section (2.2.2). Firstly, the **control inputs** (i.e. acceleration and steering angle) need to be within certain limits:

$$\mathbf{u}_{\min} \leq \mathbf{u}_k \leq \mathbf{u}_{\max}. \tag{2.4}$$

There is also a limit on the rate of change of the control inputs:

$$\Delta\mathbf{u}_{\min} \leq \Delta\mathbf{u}_k \leq \Delta\mathbf{u}_{\max}. \tag{2.5}$$

Secondly, the vehicle must remain between given **track boundaries**. For this, similarly to Liniger et al. (2015), linear constraints are specified as illustrated on Figure 2.7. For every step of the finite prediction horizon, reference points on the centreline (green dots on the figure) are computed as points which the vehicle should reach at every time step if driving at maximum velocity, starting from the projection of the vehicle's position (blue rectangle on the figure) on the centreline. This can be expressed as follows

$$dist_k = dist_{k-1} + time\_step \times (max\_velocity + time\_step \times max\_acceleration) \tag{2.6}$$

where $dist_k$ is the distance of a reference point ref_k from the point on the centreline which is nearest to the current position of the race car. The reference point (x_ref_k, y_ref_k), with $k = \frac{N}{2}$, where N is the length of the prediction horizon, is projected onto the track boundaries (red dots on the figure). Then, for each of these boundary points, a tangent line is obtained and used to limit the position of the vehicle. The constraint is expressed as follows:

$$y = ax + b \tag{2.7}$$

$$y = cx + d \tag{2.8}$$

$$(ax_k + b - y_k) \times (cx_k + d - y_k) < 0 \tag{2.9}$$

where equations 2.7 and 2.8 are the equations of the left and right tangent lines, and (x_k, y_k) is a predicted position of the vehicle. This constraint should assure that the vehicle remains between the tangent lines at all times and hence between the track's boundaries.

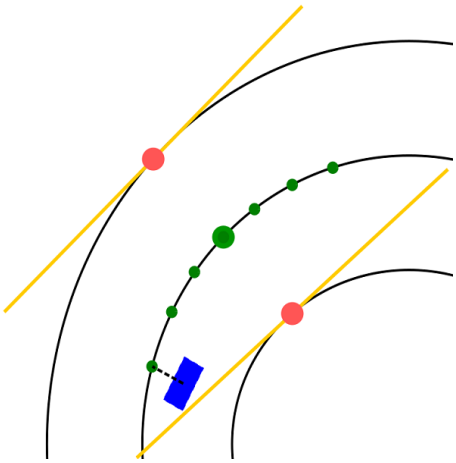


Figure 2.7: Track boundaries

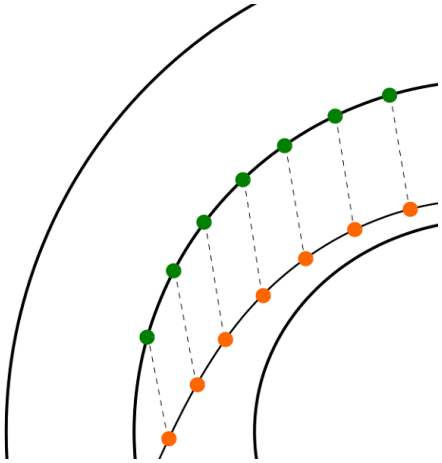


Figure 2.8: Tracking cost

Cost Function

In a cost function, the weighted objectives of the controller are specified. The main objectives of MPCC as a racing controller of this thesis are maximizing the traversal speed of the vehicle, i.e. maximizing the progress along a given reference track, and maximizing the path-tracking accuracy, between which there is a clear trade-off. On top of that, rapid changes of control inputs are penalized in the cost function. The goal of maximizing the progress along the reference can be achieved either by maximizing the velocity of the vehicle or by minimizing the travelled distance, or combination of both.

The **tracking accuracy cost**, is defined as a sum of the euclidean distances between the predicted positions of the vehicle and the reference points on the centreline for given prediction horizon, as illustrated in Figure 2.8.

Problem Formulation

Putting all of the above together, the optimization problem that is solved at each time step is specified as follows:

$$
\begin{aligned}
&\min \sum_{k=0}^{N} e_k^T \mathbf{w}_1 e_k + \Delta\mathbf{u}^T \mathbf{w}_2 \Delta\mathbf{u} \\
&\text{s.t. } \mathbf{x}_0 = \mathbf{x}(0), \\
&\quad \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k), \\
&\quad (ax_k + b - y_k) \times (cx_k + d - y_k) < 0, \\
&\quad \mathbf{u}_{\min} \leq \mathbf{u}_k \leq \mathbf{u}_{\max}, \\
&\quad \Delta\mathbf{u}_{\min} \leq \Delta\mathbf{u}_k \leq \Delta\mathbf{u}_{\max},
\end{aligned}
\tag{2.10}
$$

where $\mathbf{w}_1$ and $\mathbf{w}_2$ are the weight matrices given to the costs and $f(\mathbf{x}_k, \mathbf{u}_k)$ is the function 2.1 describing the behaviour of the vehicle.

2.2.3 Optimization Engine

As the optimization problem needs to be solved at every time step, it is crucial that the optimizer computes the control inputs efficiently, in order of milliseconds, so that the vehicle is controlled accurately and smoothly. For this purpose, the Optimization Engine (OpEn) (Sopasakis et al., 2020) framework is used, together with the CasADi tool (Andersson et al., 2019), which assists with the implementation of some of the OpEn functionalities. One of the advantages of using this code generator is that users can focus on the design of the problem rather than the numerical optimization. Figure 2.9 shows the high-level steps of using this code generator.

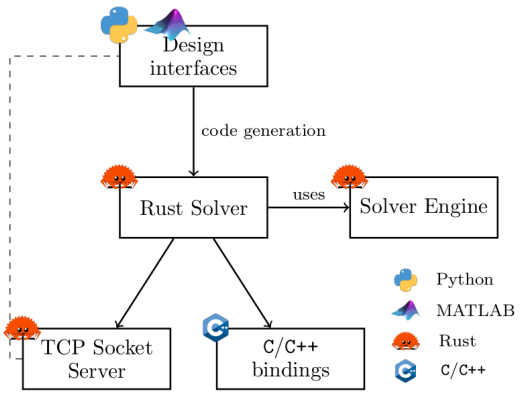


Figure 2.9: Structure of the prototyping and embedded code generation framework of OpEn (Sopasakis et al., 2020)

In this thesis, the parametric optimization problem is designed in Python, making use of the `opengen` library. The code can be generated in different ways:

- Directly in Rust,
- Over a TCP/IP socket based on JSON, accessed from any programming language,
- In C/C++, using automatically generated bindings,
- In ROS using automatically generated ROS package.

The user simply needs to specify the optimization problem, i.e. the cost function and the constraints, adjust the configuration of the solver depending on the problem (e.g. maximum amount of inner/outer iterations, maximum time to solve problem) and run the generation of the code in whichever interface is the most convenient.

This solver uses the proximal averaged Newton-type method for optimal control (PANOC), which outperforms other commonly used methods, and has significantly faster convergence speed, and hence is well suited for embedded applications (Sopasakis et al., 2020). For more information, the interested reader can consult the paper of Sopasakis et al. (2020) or visit the website of the framework.

3 IMPLEMENTATION

3.1 Boundary Estimation

The implementation of the Boundary Estimation component of the autonomous system was entirely hand-coded in C++ to the C++14 standard, to enable compatibility with ROS middleware which is used by the FEB driverless team to form the communication backbone of the car. The project was coded in an object-oriented style for easier parsability during continued development in the following years. In this section, implementation choices and details are expanded upon.

3.1.1 Cone input

Initial cone processing

During cone addition, an attempt is made to classify cones using previously classified cones and the car's initial position. In the case that a cone is received that has not been classified previously by the cone classification, there are two different approaches taken to attempt a classification depending on whether previous cone classifications have been received.

In the case that there are no previously classified cones available, classification is attempted according to the car's direction, by projecting a line in front of the car and testing the sign of a calculated cross product with the cone position and a projected vector in the car's direction. With the car at point A, the projected point along the car's position at point B and the cone to be classified at point C, the determinant of vectors $(\vec{AB}, \vec{AC})$ is given by $(B_x - A_x) \times (C_x - A_y) - (B_y - A_y) \times (C_y - A_x)$ with the sign of the resulting determinant indicating on which side of the vector $\vec{AB}$ the cone at point C can be found. This is illustrated on Figure 3.1. This test is clearly susceptible to turns in the track, and is intended only for classifying the very first cones of the track, upon car start-up, in case of a failure to classify by the stereovision component.

In the case that there are sufficient previously classified cones available, a different approach is taken to take advantage of the rules of track construction put forward by the Formula Student competitions. With a minimum track width of 3m and a maximal distance between cones of 5m, a cone area of length 5m and maximal width 3m, as shown on figure 3.2 is projected in the current direction of the boundary at the closest previously classified left and right boundary cones. In the case that the cone area originating at only one of the previously classified cones contains the unclassified cone, it is classified as belonging to the same boundary. In the case that both or neither cones contain the unclassified cone, no classification is made.

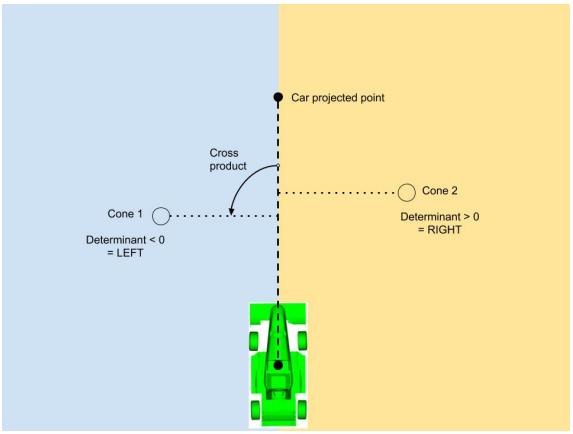


Figure 3.1: Artificial classification as left or right boundary using vector math

Framing of section to be processed

The framing of cones is implemented through three separate lists of cones that are held in memory:

- Cones that have already been processed - used in order to develop a segment of track centreline.
- Cones that have yet to be processed - cones that have yet to be used to develop a segment of racing line.
- Cones that are currently being processed as part of a framed track segment - cones that have been moved out from the list of cones that have yet to be processed. Upon completion of the centreline, these cones will be moved into the list of cones that have already been processed.

The decision for whether a cone that has yet to be processed is within the next framed track section is made according to the checking whether a cone is located within or without a circle section that is projected in front of the car, as shown on Figure 3.3. This is a stand-in technique to mimic the camera vision cone that will exist on the real car, and is adjustable both by arc length and radius. Once the software is being operated on the final driverless car, this check can be replaced by a simpler Euclidean distance check. The distance check will still be necessary as the accuracy of cone detections can only be assumed within certain ranges of the car, due to physical limitations of the vision sensors. The range values that are appropriate will be found experimentally in testing of the Umicore Eclipse in following years.

3.1.2 Track section discretization and exploration

Triangulation

To simplify the development process, an existing C++ library called `Delaunator` was used for Delaunay triangulation. The list of points added for triangulation includes the points of all cones

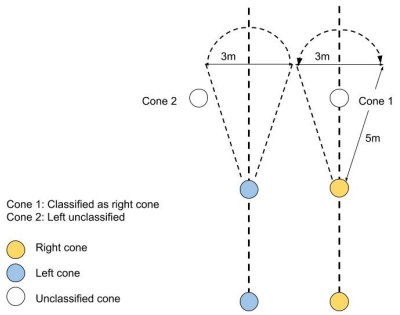


Figure 3.2: Artificial classification with existing classified cones

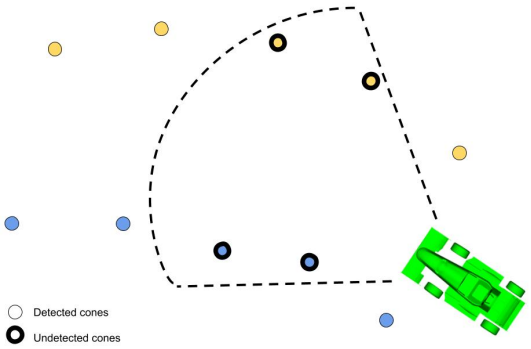


Figure 3.3: Cone framing mimicking camera vision

within the framed track section and a section entry point. This section entry point is chosen to be either the position of the car, for the first track section, or the last known centreline point, for all following sections.

Pathway Exploration

Once triangulation is complete, the list of triangles, along with the section entry and exit points, is used to fill the tree. To limit the number of parent-child waypoint connections to reduce computational time, the best first distance is used to indicate if a particular parent-child link is moving towards or away from the section exit point. A tolerance variable allows for a limited amount of movement away from the section exit by allowing the link to still be considered viable when the distance to exit increases by a sufficiently small value. This is necessary to account for the centre line not always travelling directly towards a section exit during more extreme track features, such as sharp turns. The typically small framed sections of track should minimise this effect, allowing for a relatively small tolerance that can be adjusted experimentally, with the compromise that greater tolerances will increase the number of viable paths and the resulting computational load.

The parent-child waypoint links are formed through the following process:

1. For a given waypoint (henceforth referred to as the parent waypoint), all bordering triangles are found for which this point is on one of their line segments.
2. The possible child waypoints (triangle line segments midpoints) are found on all other line segments of those triangles.
3. The Euclidean distance from the parent waypoint to the section exit point is calculated and compared with the Euclidean distance from each of the child waypoints to the section exit point.
4. All child waypoints whose distance from the exit is within best first distance tolerance of the parent waypoints distance are added to the tree with the parent waypoint indicated as the previous stage in the travel link.

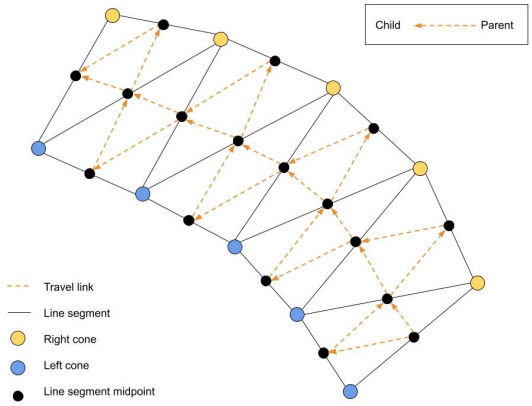


Figure 3.4: Cone framing showing tree links formed across track section

This process is repeated until all possible waypoints in the framed track section have been collected and added to the tree, and paths can be developed. A slight exception to the process is necessary at the section entry point, as it is added to the triangulation as a point but, unlike all other triangulated points, it does not mark a boundary as it is a traversable point (that is, a point to which travel is viable). As such, there is slightly different behaviour for creating the first parent-child travel link, where the two opposing points of the triangle are found and the midpoint of the line segments between is treated as the first child waypoint.

Once the tree is filled with all parent-child waypoint connections (a simple example of the waypoint links possible shown in Figure 3.4) that appear to be potentially valuable travel paths according to their best first distances, the paths must themselves be developed and collected.

The tree itself is made of nodes, each representing a waypoint in the framed track section. These nodes have the following structure:

Coordinate XY
: Doubles representing X and Y positions of waypoint.

bool checked
: A boolean to check if a waypoint has been checked to find its possible children.

Node parentPointer
: Indicates the node in the tree that acts as its parent waypoint in the track section.

list childrenPointers
: A list to indicate all the children nodes in the tree for which this waypoint is the parent link.

int childrenIterate
: An iterating value that acts to indicate which children have already been collected into paths.

The tree itself is built as an artificial data structure of pointers linking to nodes held in a list, to take advantage of the speed benefits of a list thanks to its data contiguity allowing for storage on upper-level (with respect to the processor) caches.

A pseudo-code representation of the tree operation is given in Algorithm 1, to demonstrate how the path extraction is performed. In this way, the path is filled up to the maximum desired length

with only the first children of each node, and the path is added to the collection of paths with each added node. Once the path is at the longest desired length, the tree begins moving up the tree to the parent of the furthest node, then down to its next child and adding the path, until all its children have been collected into paths. The loop continues throughout the tree, up until the root node has exhausted all of its children and all paths have been collected from the tree. These are returned as a list of lists of coordinates.

Algorithm 1: Tree operation

```
currentTreeDepth ← 1;
while paths not all collected do
    if current node has childrenIterate = 0 then
        Add node to path;
        Add path to collection of paths;
    Increase value of childrenIterate for current node by 1;
    if current node has childrenIterate > 4 OR currentTreeDepth = maximum desired path
     length then
        if current node is root node then
            All paths collected, exit tree;
        else
            Move to make parent of current node the new current node;
            Remove last node from currently working path to make room for next child of
             current parent;
            Reduce currentTreeDepth by 1;
    else if current node has an existing child at index childrenIterate then
        Move current node to child at index childrenIterate;
        Increase currentTreeDepth by 1;
    else
        Current node does not have an existing child at index childrenIterate, so all paths that
         can be derived from current node are done. Set childrenIterate to 5;
```

3.1.3 Pathway costing and selection

Pathway costing is done by creating a list of costs associated with the list of lists of coordinates that makes up the pathways. The cost parameters that are used to just the suitability of a track for use as a centreline are summarised in Table 3.1.

Name	Description	Calculated at each point	Averaged over length
Classified distance to nearest cone (Figure 3.5)	Square of the Euclidean distance to nearest left boundary cone minus the euclidean distance to nearest right boundary cone.	Yes	Yes
Unclassified distance to nearest cone (Figure 3.5)	Square of the Euclidean distance to the nearest cone at point n minus Euclidean distance to the nearest cone at the point n-1.	Yes	Yes

Distance from track section exit (Figure 3.6)	Square of the Euclidean distance from the final point in path the the track section exit point.	No	No
Classified compare number of cones on each side (Figure 3.7)	Square of the number of left boundary cones minus the number of right boundary cones found in a rectangular projection around the point at point n minus the same check at point n-1. If no cones found, additional cost applied.	Yes	Yes
Unclassified compare number of cones on each side (Figure 3.7)	Square of the number of cones found in a rectangular projection around the point at point n minus the same check at point n-1. If no cones found, additional cost applied.	Yes	Yes
Advancing towards goal	Best first distance to section end at point n minus same check at point n-1. If value negative, squared difference added as cost.	Yes	No
Check for repeat nodes	Fixed cost added for each node found to have been visited twice in path.	No	No

Table 3.1: Summary of cost parameters for path selection

All parameters that have an additional coefficient applied which checks the ratio of left boundary cones vs right boundary cones. In cases of a small imbalance, such as during a track section during a turn, this will help compensate for the slight inaccuracies this may lead to in the classified checks relative to the unclassified checks.

Additionally, when the ratio is found to be beneath a certain value, that coefficient is set to zero to remove classification parameters from path costings. This is to reduce the likelihood of situations where either the stereo camera or artificial classification attempts have failed from affecting the centreline mapping in a way that could prevent robust mapping of the track centreline. While a more robust check for cone boundary classification failure would be a valuable expansion on the Boundary Estimation software component, in most cases classification failure should lead to excessive classification of a single boundary which should quickly cause classification parameters to be removed.

3.1.4 Outputs

Reference path delivery

The centreline coordinates are recorded as a set of discrete coordinates at variable distances from one another dictated by the Delaunay condition and cone positions. Interpolation is required to provide the required inputs for the MPCC component.

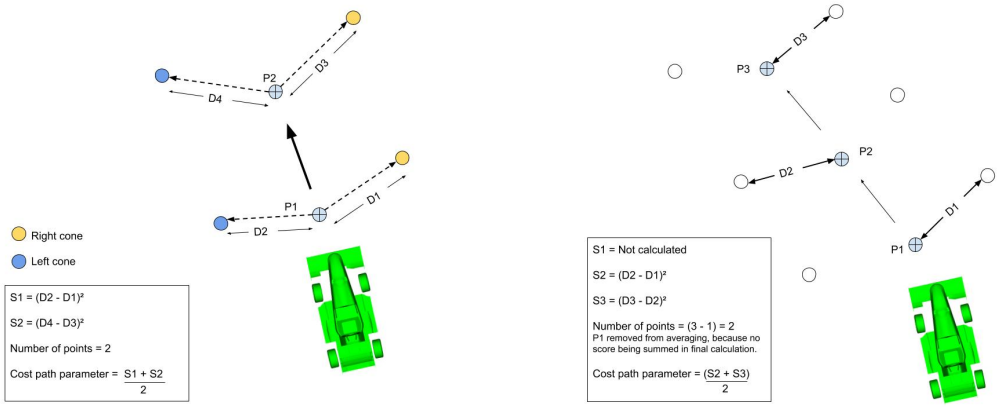


Figure 3.5: Example of classified (left) and unclassified (right) distance to nearest cone test

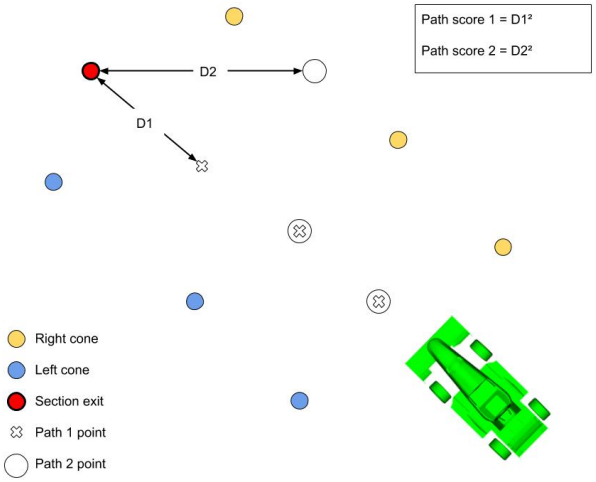
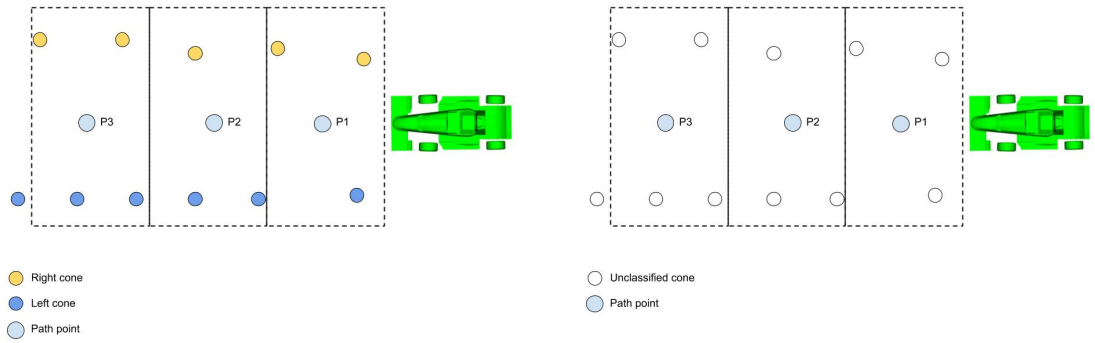


Figure 3.6: Example of distance from track section exit test



$S_1 = |(1-2)| = 1$ $\quad S_1 = 3$

$S_2 = |(2-1)| = 1$ $\quad S_2 = 3$

$S_3 = |(2-2)| = 0$ $\quad S_3 = 4$

$Pathscore = (S_2 - S_1)^2 + (S_3 - S_2)^2 = 0 + 1 = 1$ $\quad Pathscore = (S_2 - S_1)^2 + (S_3 - S_2)^2 = 0 + 1 = 1$

Figure 3.7: Example of classified (left) and unclassified (right) compare number of cones on each side test

The steps for providing this are as follows:

1. Finding the nearest point on the centreline to the car's current position. The nearest two centreline coordinates are retrieved by Euclidean distance, and the nearest interpolated point is found by finding the perpendicular vector that travels through the car's position using the dot product.
2. The number of requested points are interpolated over the requested length of centreline (after a check to ensure the requested length of centreline is available). This is done by forming linear connections between successive centreline coordinates and stepping through them by the required distance, saving each point into a vector.
3. For each of those interpolated centreline points, a point is interpolated on the boundary line by finding the closest point on the line between the two closest cones on the left and right boundaries. The slope between those points and their closest cones is then found.

Boundary condition violation

The process chosen for expressing the boundary conditions in a computationally efficient way is as follows:

1. The closest point on the centreline to the car is found, via linear interpolation between the closest two centreline coordinates. This point will act as the centre of a circle whose area will be regarded as 'within the boundaries'.
2. The closest cone to that point is found. The distance between the centreline point and the closest cone is judged to be the radius of the circle originating at the centreline point. The edge of the circle is judged to be the boundary, and beyond the circle is outside of track boundaries.
3. The car is projected out from its point mass coordinates to four corners in a rectangle describing the physical dimensions of the Umicore Eclipse.
4. The furthest car corner point from the centreline point is found via Euclidean distance checks.
5. The furthest car corner point is then tested to find if it is within the boundary circle area. If the answer is yes, the car is judged to be within track boundaries and normal operation continues. If not, the emergency stop state is activated.

Track loop completion

Track completion checks begin after a number of centreline coordinates have been collected. This is to prevent the possibility of the track being falsely registered as complete at the earliest stages of the race, as the first centreline coordinates may satisfy the track loop completion state without more advanced, computationally expensive checks. The process for track loop completion is as follows:

1. The last centreline coordinate is checked against a circle of radius 7.5m around the first centreline coordinate. If the last coordinate is within the circle area, the following checks are done. Otherwise, the track loop is judged to be incomplete.

2. A number of points are interpolated on a linear path between the last reference coordinate and the first. Each of these points is then checked for being out of track boundaries. If any of those points are judged to be outside of track boundaries, the track loop is judged to be incomplete. Otherwise, the track completion check is satisfied and track centreline mapping is deactivated

3.2 Model Predictive Contouring Control

In order to implement the controller, the OpEn framework was used via the Python `opengen` library. The problem was specified in Python and for the testing purposes throughout the implementation process, the solver was called over a TCP server that uses JSON data format, which was accessed also from Python code. Eventually, the generated solver was executed from C++ code with the help of the automatically generated C++ bindings. In this section, the necessary steps for the generation of the solver and their implementation are discussed.

Dynamics of the plant

The first step in the implementation of MPCC was the modelling of the dynamics of the plant, i.e. the vehicle. As explained in section 2.2.1, the vehicle model plays a crucial role in the accuracy and complexity of the optimization problem and the kinematic model (2.1) was chosen for the purposes of this thesis. The 4^{th} order Runge-Kutta method was applied to discretise the model, to calculate the change of state at each time step, given certain control inputs.

Cost function

The cost function was implemented as a function that takes as parameters the state x_k, the corresponding reference point r_k , the control inputs u_k and their change $\Delta u_k = u_k - u_{k-1}$ and the weight given to the tracking accuracy cost for step k.

The weight given to the tracking accuracy cost, i.e. the Euclidean distance between the car's position and the reference position (cfr. section 2.2.2), increases at every time step, starting from $k = \frac{2}{3}N$, by $\frac{4k}{N}$%, where N is the length of the prediction horizon, to give more importance to reaching the further reference points and the end goal rather than to the close following of the centreline. These values ($k = \frac{2}{3}N$ and $\frac{4k}{N}$%) were obtained experimentally.

The goal of maximizing the progress along the reference line, on top of the tracking accuracy cost, was implemented by having the desired velocity being controlled by the length of the reference line and by the distance between consecutive reference points. The control inputs should lead to the vehicle reaching the reference points and therefore, due to the way in which the reference line was calculated (cfr. equation 2.6), the car needs to reach the maximum velocity in order to be able to follow the reference points. This approach was chosen because the kinematic bicycle model does not take into account the slip forces and so the velocity has no other constraint than a selected minimum and maximum.

Constraints

In order to specify the constraints to the code generator using the Augmented Lagrangian

method (Sopasakis et al., 2020), a vector of to-be-constrained expressions $F_1(u,p)$, with u being the decision variable and p a parameter vector of the problem, and a set of constraints C was defined, such that $f_{\min} \leq F_1(u,p) \leq f_{\max}$. A function from the `opengen` library, `og.constraints.Rectangle(fmin, fmax)`, was used to compose the set C to describe the min-max constraints. To constrain the decision variable, i.e. the control inputs u_k, set U was defined, also using the `Rectangle` function. These were specified in the problem as explained below.

OpEn problem definition and settings

To put all this together, the problem had to be defined in a way that would be compatible with the OpEn code generator. The problem was designed in Python and to generate the solver, these steps were followed:

1. **Specifying the cost function and constraints**
   The cost function was summed over the span of the prediction horizon and the constraints were applied for each prediction, hence they were defined in an iterative way as shows in Algorithm 2.

   Algorithm 2: Cost function and constraints as a function of decision variable (= control inputs) and parameter vector (= initial state, track boundaries, reference points)

   Result: $\mathrm{cost}(\mathrm{u},\mathrm{param}), \mathrm{F_1}(\mathrm{u},\mathrm{param})$

   $u_{seq} \leftarrow$ cs.MX.sym("u", num_control_inputs $\times$ prediction_horizon);
   $param_{seq} \leftarrow$ cs.MX.sym("params", num_parameters $\times$ prediction_horizon);
   $cost \leftarrow 0$;
   $F_1 \leftarrow [\,]$;
   for *i = 0 to prediction_horizon* **do**
       Update $cost$: cost ← cost + cost_func(state, reference, delta_u, track_weight);
       Update F_1: F1 ← cs.vertcat(F1, state, track_constraint);
   end

   Here, u_{seq} and $param_{seq}$ denote CasADi (cs) symbolic matrices which each contain a list of variables that can be appropriately substituted when the solver is called. The resulting expression for $cost$ and the symbolic matrix $F1$, whose length is equal to the $prediction\_horizon$, are composed of these variables, which are passed to the problem definition as explained below.

2. **Problem definition**
   The problem was then defined using the functions of `opengen` (og) as follows:

```
C = og.constraints.Rectangle(f_min, f_max)
U = og.constraints.Rectangle(u_min, u_max)
problem = og.builder.Problem(u_seq, param_seq, cost) \
          .with_aug_lagrangian_constraints(F1, C) \
          .with_constraints(U)
```

   Here, `f_min` and `f_max` are vectors of minima and maxima, respectively, which express the limits on the expressions in the set `F1`. Similarly, `u_min` and `u_max` are vectors of minima and maxima, respectively, which express the limits on the control inputs in `u_seq`.

3. **Settings and build**

   Finally, a number of settings was defined and the code was generated executing the `og.builder.OpEnOptimizerBuilder.build()` function.

   The metadata contain some general information about the auto-generated solver. The most important specification is the `optimizer_name` which is the name of the folder where all the generated files are stored.

```
meta = og.config.OptimizerMeta() \
       .with_optimizer_name("mpcc_optimizer") \
       .with_authors("Darina Abaffyova & Andrew Gordon")
```

   The build configuration contains more information about how the solver should be built, such as the path where the generated files should be stored, the build mode, `DEBUG` or `REALEASE`, and depending on whether it should be accessed via the C/C++ bindings or via a TCP socket, the appropriate function is executed (i.e. either `with_build_c_bindings()` or `with_tcp_interface_config()`).

```
build_config = og.config.BuildConfiguration() \
          .with_build_directory("mpcc_c_build_kin") \
          .with_build_mode(og.config.BuildConfiguration.RELEASE_MODE) \
          .with_open_version("0.7.0-alpha.1") \
          .with_build_c_bindings()
```

   Last but not least, some solver parameters need to be specified. These influence the speed of convergence of the solver and should be tuned depending on the problem. More detailed information on the used algorithm and the parameters involved can be found in (Sopasakis et al., 2020).

```
solver_config = og.config.SolverConfiguration() \
                .with_initial_tolerance(0.01) \
                .with_tolerance(0.01) \
                .with_delta_tolerance(0.01)  \
                .with_initial_penalty(20) \
                .with_penalty_weight_update_factor(4) \
                .with_max_duration_micros(50000)
```

   At last, all the above information can be provided to the generator and the solver can be built.

```
builder = og.builder.OpEnOptimizerBuilder(problem,
                                  metadata=meta,
                                  build_configuration=build_config,
                                  solver_configuration=solver_config)
builder.build()
```

Generated code

The generated code was then used to compute the control inputs provided to the car. At first, for the purposes of simplicity in testing during the implementation phase of the algorithm, the solver was generated with the TCP interface configuration and accessed from a simulator programmed in Python as follows:

```
# Create a TCP connection manager
mng = og.tcp.OptimizerTcpManager(generated_optimizer_path)
# Start the TCP server
mng.start()
# Run simulation
for k in range(simulation_steps):
    solver_status = mng.call(parameters, initial_guess_for_control_inputs)
    # Use the solver
    control_inputs = solver_status["solution"]
```

```
    ...

# Kill the server
mng.kill()
```

The parameters provided to the solver were the state of the race car, a list of reference points, whose length corresponded to the prediction horizon, and a slope and y-intercept of each of the tangent lines defining the boundaries of the track.

The `solver_status` outputted by the manager contained the cost, exit status, Lagrange multipliers, number of inner and outer iterations, final penalty, the solve time in milliseconds and the actual solution which were the result of the problem solved for the given parameters. For the `initial_guess_for_control_inputs`, the control inputs calculated at the previous iteration were used. In this stage, different approaches were tested and evaluated based on a visualisation of how well the vehicle followed the track, on the length of the driven track in comparison with the given reference, and on the achieved processing time.

Eventually, once both the MPCC and the boundary estimation algorithms were in place, the solver was accessed via the generated C++ bindings:

```
/* obtain cache */
mpcc_optimizerCache *cache = mpcc_optimizer_new();

/* solve */
mpcc_optimizerSolverStatus status = mpcc_optimizer_solve(cache,
    control_inputs, parameters, initial_guess_lagrange_multipliers, &
    init_penalty);

/* apply the control inputs */
control = {control_inputs[0], control_inputs[1]};
car->updateCar(control, time_step);

/* free memory */
mpcc_optimizer_free(cache);
```

Further testing and adjustments were performed in the C++ environment combined with visualisation implemented in ROS.

All in all, making use of the Optimization Engine has greatly simplified the implementation of the controller, as the main focus could be given to the design of the problem. On top of that, it facilitated the fast solving of the optimization problem, which is essential for this application. With more time to delve into the details of the advanced mathematical concepts involved and with opportunities for testing on the actual race car with real-life values, significant improvements of the problem design and hence the results can be expected.

4 RESULTS

4.1 Boundary Estimation Results

4.1.1 Setup

To test and visualise the results of the track mapping and boundary estimation, RViz, a 3D visualisation of the ROS software package, was used. Test cone data was input into the package and track mapping data was output for visualisation. The cone data was prepared according to Formula Student competition rules.

The goal of these tests was to visually assess the accuracy of the centre line produced and the ability of the system to accurately detect out-of-boundary conditions, judge track completion, provide lap counts and provide reference paths. As the boundary estimation and track mapping acts as a bridging system between cone detection and MPCC, the metric for achievement of design goals was the minimum computation time, targeting the desired refresh rate of 40Hz and assuming a significant portion of available time would be used computing the MPCC portion of the system, a general target guideline of 5ms was be used. This computation time or lower should be achieved while not negatively influencing combined system performance as analysed in the final section where the overall system results are observed.

4.1.2 Cone input

In Figure 4.1 a simple representation of the physical grid with a set of cones placed within, representing input to the Boundary Estimation system, is shown. The coloured cones above the cones represent their boundary classification (as per Formula Student rules, blue indicating left and yellow indicating right boundary). As new cones are added, they are framed according to proximity to the car and the physical vision cone of the car vision sensors, which is currently simulated to mimic as close to real-life behaviour as possible.

Section exit

The selection of the section exit from a framed section can be seen in Figure 4.2 where through a mixture of Euclidean distance checks from the section entrance and cone distance comparisons, to avoid edge-case scenarios during short turns, a section target is selected. The process for acquiring the section end is not foolproof and can fail in cases of extreme track contours or when track frames are allowed to be large enough that distance measurements are no longer an adequate metric. An example of this can be seen in Figure 4.3 where the section end indicator has not been placed at the exit to the frame, reducing the number of centre coordinates that will be extracting and causing a larger 'jump' into the next frame.

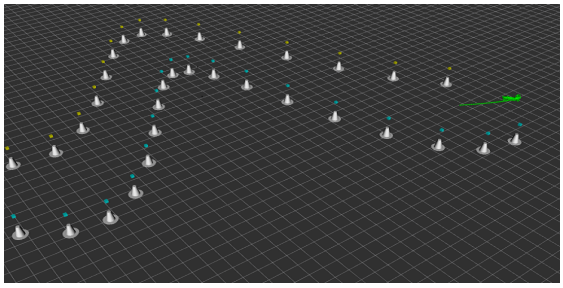


Figure 4.1: Classified cone boundaries

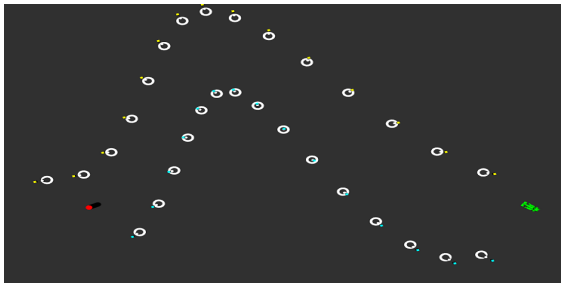


Figure 4.2: Selection of section exit to act as path target end - section exit indicated with red cylinder

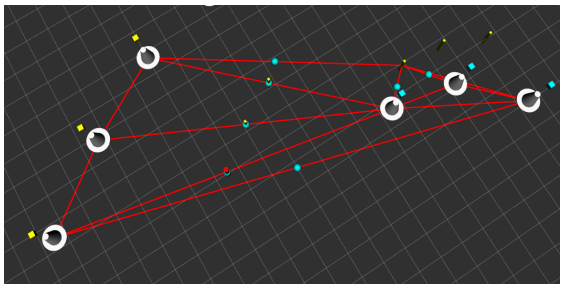


Figure 4.3: Failed section end

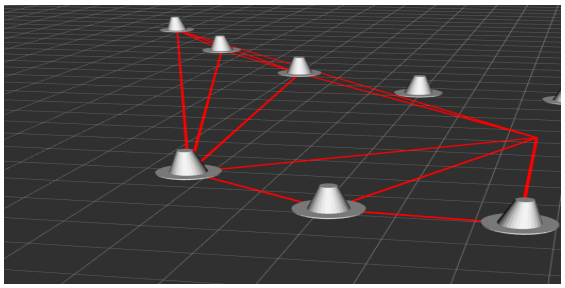


Figure 4.4: Delaunay triangulation example across six cone frame

4.1.3 Track section discretization and exploration

Triangulation

Figure 4.4 shows an example of Delaunay triangulation being applied to a frame consisting of three left boundary and three right boundary cones. The line sections (drawn in red) clearly connect vertices consisting of cone positions and the section entry point.

By applying a graph search algorithm to midpoints on line sections, the computational load of the path-finding algorithm is greatly reduced but this necessarily reduces the potential centre coordinate points that can be found. An example of the impact that this can have is shown in Figure 4.5 where the left boundary and right boundary cones in a frame are shifted relative to each other. This leads to a large triangulation 'gap' in which no centre coordinates can be found.

Pathway Exploration

The developing of the pathways that describe potential travel routes in a section involves the placement of traversal nodes at the midpoint of line sections. An example of this being done is shown in Figure 4.6, where nodes have been placed at the midpoints of line sections within the frame, with information on their parent node.

Within the process of placing nodes in a section, a Best First algorithm is used to attempt to reduce computational complexity by restricting the addition of new nodes to only those that have a

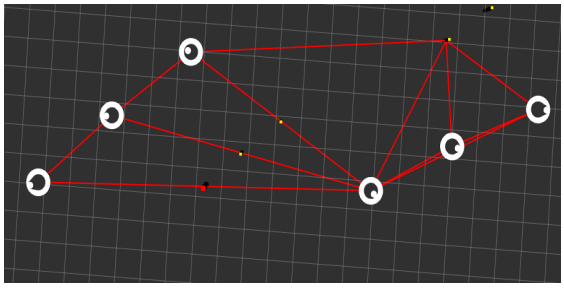


Figure 4.5: Example of a centre coordinate 'gap' due to a shifted cone frame

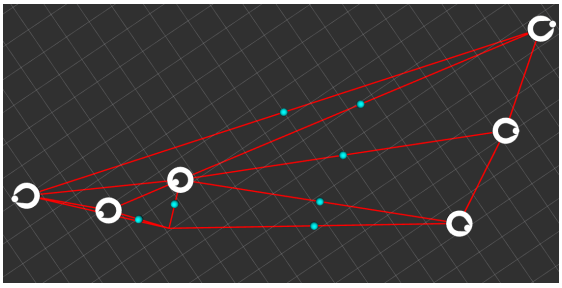


Figure 4.6: Example of traversable nodes (blue spheres) placed at triangulation line section midpoints

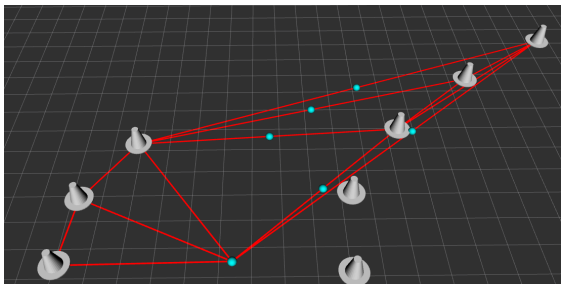


Figure 4.7: Example showing nodes only placed at midpoints satisfying Best-First algorithm (moving towards section exit)

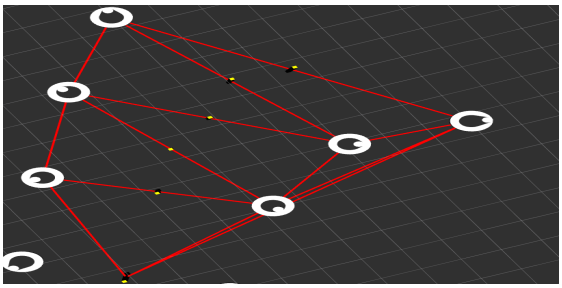


Figure 4.8: Coordinates (yellow cuboids) selected describing path through section

reasonable prospect of being part of the final centre line through the section. This can be seen in Figure 4.7, where the section ends to the top, and only line sections with midpoints between the section entry and the section exit have been considered. This removes the need to develop paths including nodes on line sections to the left, reducing possible path length and the number of total paths significantly.

In Figure 4.8 an example is shown of the path chosen through a framed section, avoiding undesirable behaviour and describing the centre line as best as possible given the assumption of only using line section midpoints.

4.1.4 Outputs

Reference path delivery

The reference path provided to the MPCC component can be seen in Figure 4.9, showing the linear interpolation between centre line coordinates, boundary slopes given at the half-way point along the centre path and points to indicate out of boundary states. To reduce the risk of the car moving out of the boundaries, the option exists to shift them closer towards the centre line as shown in Figure 4.10. This allows for the vehicle to make small errors on remaining within the reported boundaries without triggering total car-shutdown.

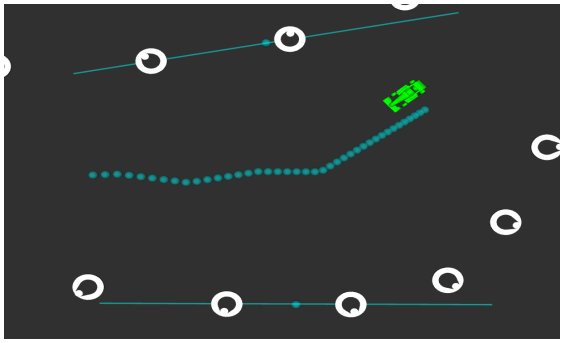


Figure 4.9: Centre-line interpolation, boundary points and slopes as required by MPCC component)

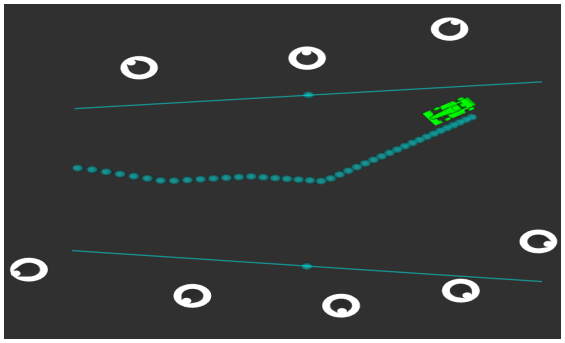


Figure 4.10: 'Compressed' boundary slopes to lessen risk of exceeding boundary conditions)

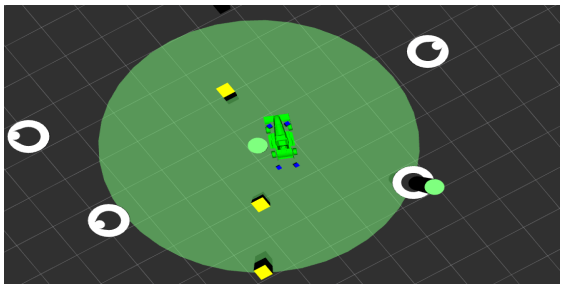


Figure 4.11: Area judged to be 'in boundary' during a single boundary violation check

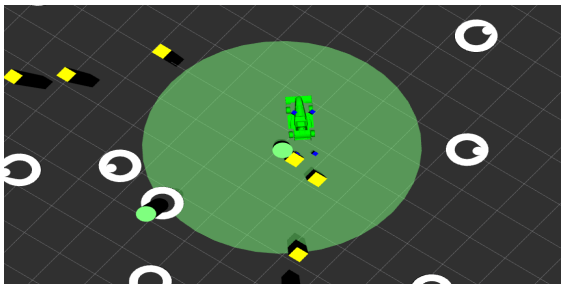


Figure 4.12: Constriction of boundary area due to linearisation error on centre line of track

Boundary condition violation

In Figure 4.11 an example of the boundary area drawn between a single check is shown. The small green circle interpolated between two centre line coordinates (represented by yellow squares) is taken to be the centre of the in-boundary area. The other circle is placed on the nearest cone, and the distance between the two circles is taken as the radius of the circle which describes the in-boundary area.

The four small blue cuboids around the car model represent the boundaries of the car (the 3D model of the Umicore Eclipse is slightly shifted compared to the point-mass taken to be the centre of the car inside the program, so they do not align precisely). If the furthest of those cuboids from the centre line is found to be outside the drawn green circle, the car will be considered to be out of bounds and will enter emergency stop state.

In Figure 4.12 we see an example where the linearisation of the connection between centre coordinates has led to an in boundary area which does not closely describe the actual width of the track.

Track centre line frame sizes

As a result of the application of Delaunay triangulation, in combination with the coordinate gaps described in Section 2.1.2 different results are found depending on the number of cones included

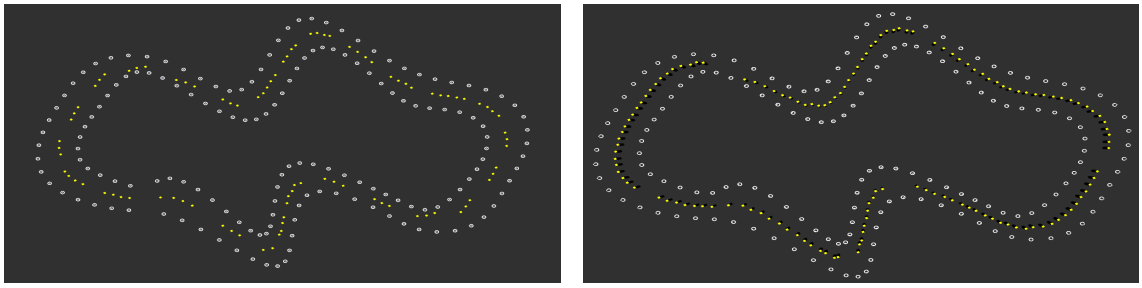


Figure 4.13: Centre line developed when average section frame contains 6 (left) and 15 (right) cones

Longest processing time in a loop	526
Shortest processing time in a loop	25
Average processing time	58.375
Number of loops to process entire track	24

Table 4.1: Timing information (in μs) to develop centre line in Figure 4.13 (left) in 6 cone frames

Longest processing time in a loop	597
Shortest processing time in a loop	314
Average processing time	456.4
Number of loops to process entire track	5

Table 4.2: Timing information (in μs) to develop centre line in Figure 4.13 (right) in 15 cone frames

in frames on average. In general, the results are better with larger frame sizes, with fewer gaps and less linearity error from interpolating between points. This can be seen by the sparser groups of points in the left panel of Figure 4.13 as opposed to the comparatively smooth and consistent centre line shown in the right panel.

Processing time

The computation time is given in Tables 4.1 and 4.2 for both six and fifteen cone frames, to demonstrate the impact on computational time from more cones being held in a single frame (requiring more triangulation, more and longer paths to be developed and costed, etc).

4.2 MPCC Results

4.2.1 Setup

In order to test the different aspects of the control algorithm, first a simulation environment was developed in Python. The track used for evaluation of the implementation is shown in Figure 4.14. The track is approximately 267.64m long and the track width ranges from 4.89m to 9.21m, with the average being 7.35m. These dimensions are consistent with the Formula Student competition rules. Each of the lines is composed of 15000 points, to improve the precision of calculation of distance along the line where needed.

A number of values had to be chosen to simulate the dynamics of the race car. These were the maximum/minimum velocity, maximum/minimum steering angle and maximum/minimum acceleration. Due to the corona protective measures, except for the steering angle which was measured beforehand, these values could not be assessed experimentally, hence they were estimated based

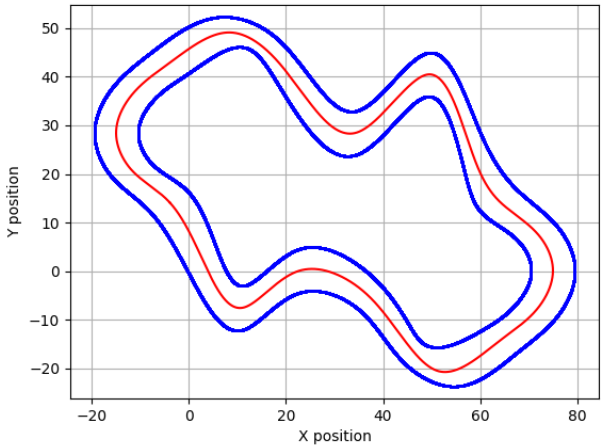


Figure 4.14: Testing track

on the properties of the used components combined with values encountered in similar projects.

Furthermore, the performed tests were aimed at evaluating the progress of the car along the reference path and the time needed to solve the problem at each time step. For this purpose, the values that were tweaked, to observe their influence on these factors, were the length of the prediction horizon N and the tracking accuracy weight $\mathbf{w}_1$ in the cost function 2.10. The prediction time step was set to $dt = 0.05s$, which allowed for shorter prediction horizon for the same time frame than if 0.025s, corresponding to the sampling frequency of 40Hz, was used (i.e. for a look-ahead of 2s, N has to be 40 for 0.05s $dt = 0.05s$ and 80 for $dt = 0.025s$), and hence shorter solve time was needed. It is shown in this section that the resulting solve time is compatible with the set sampling frequency. The tuning of the solver parameters in order to achieve optimal solutions given the constraints and cost function was a relatively lengthy process based largely on trial and error from which the values as displayed in table 4.3 were obtained. Parameters not listed in the table were kept at their default values as assigned by the OpEn developers.

Due to the use of the kinematic rather than the dynamic bicycle model in this thesis, the effect of increasing the maximum velocity parameter of the car was not meaningful, as the simulated car would simply reach any velocity without having the slip forces limit this in corners, as they are not part of the kinematic model. Therefore, the traversing speed was evaluated on the basis of the shape and length of the driven track rather than the driving time of the modelled formula, keeping the maximum velocity equal for each trial.

Initial tolerance	0.01
Tolerance	0.01
Delta tolerance	0.01
Initial penalty	20
Penalty weight update factor	4
Maximum duration in μs	100

Table 4.3: Solver parameters

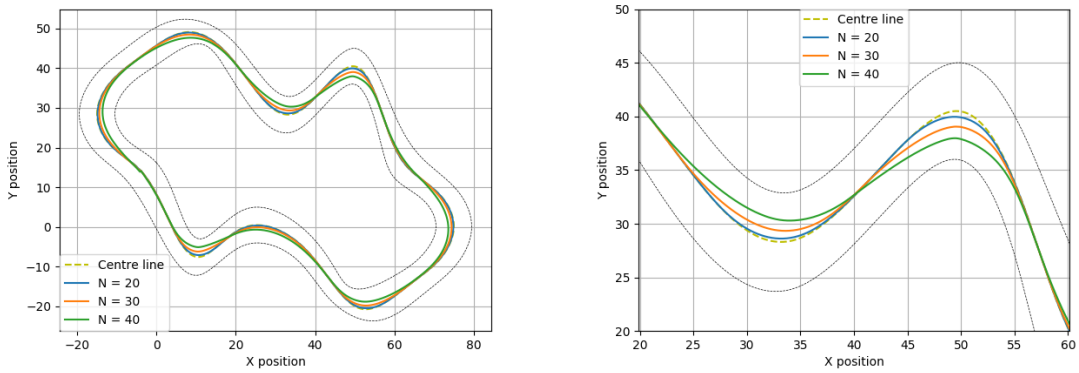


Figure 4.15: Effect of prediction horizon on progress along the track (with $\mathbf{w}_1 = 0.1$)

It is important to note that ideal scenarios were evaluated, meaning that the noise occurring during measurements of the state of the vehicle was neglected, the control inputs were assumed to result into an ideal output according to the vehicle model, and the reference path was smooth for the complete track.

4.2.2 Progress along the track

Different paths were followed by the simulation vehicle depending on the prediction horizon and the tracking accuracy weight, i.e. the weight given to the tracking accuracy cost (cfr. section 2.2.2). Every simulation presented in this section has been run such that the complete track was traversed exactly once and the weights for input rate cost, the acceleration and steering angle respectively, was kept at $\mathbf{w}_2 = [0.01, 0.1]$.

Prediction horizon

While keeping the tracking weight at $\mathbf{w}_1 = 0.1$ and increasing it starting at time step $k = \frac{N}{2}$ by $(\frac{4k}{N})\%$, and with time step $dt = 0.05s$, prediction horizons $N = (20, 30, 40)$ were tested. The comparison is shown in Figure 4.15, where the left panel presents the complete driven paths and the right one zooms in on a section of the track to better show the difference. The lengths of the track driven corresponding to each value of N are shown in Table 4.4. It can be seen that the configuration with the shortest prediction horizon follows the reference line more closely and hence driving the longest track, while the longest prediction horizon allows for shorter driving distance.

For prediction horizons above 40, the solver began to not be able to converge to a solution within

$\mathbf{w}_1 = 0.1$	
N = 20	265.18m
N = 30	258.71m
N = 40	249.91m

$N = 40$	
w1 = 0.05	249.56m
w1 = 0.1	249.17m
w1 = 0.2	248.94m

Table 4.4: Effect of prediction horizon (left) and tracking accuracy weight (right) on progress along the track

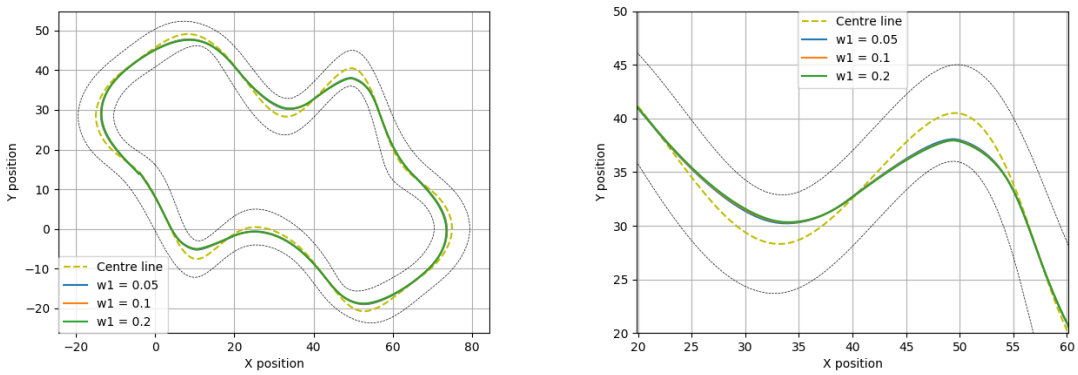


Figure 4.16: Effect of tracking accuracy weight on progress along the track (with $N = 40$)

the given time at the corner sections of the track, which tended to lead to unpredictable or unstable behaviour and often to violating of the imposed constraints which caused the car to leave the track.

Tracking accuracy weight

While keeping the prediction horizon at $N = 40$ and with time step $dt = 0.05s$, tracking weights $\mathbf{w}_1 = (0.05, 0.1, 0.2)$ were imposed on the tracking cost, increasing them by $(\frac{4k}{N})$% starting at time step $k = \frac{2}{3}N$. The result of each of this is displayed in Figure 4.16, where the left panel presents the complete driven track and the right one zooms in on a section of this track. As can be seen on the figure, the resulting tracks are virtually overlapping. The lengths of the track driven, corresponding to each value of $\mathbf{w}_1$, are shown in Table 4.4. They are indeed approximately equal, around 249m.

4.2.3 Solve time

Because of the importance of the processing time in this application, the time needed for the optimizer to solve the problem at each time step, i.e. to calculate the control inputs for the race car, had to be observed. The solve time is one of the outputs provided by the solver, and this was plotted in a histogram for each of the cases discussed above, as can be seen on Figure 4.17. The maximum time allowed for the optimizer to solve the problem was set to 100ms and a sampling time of 25ms was considered for evaluation.

As expected, the solve time increases with both increasing prediction horizon and increasing tracking accuracy weight, since more calculations are needed to predict and constrain the future states of the vehicle and higher cost is imposed on solutions that do not follow the reference closely. Nonetheless, in all five cases the solve time remains well below 25ms.

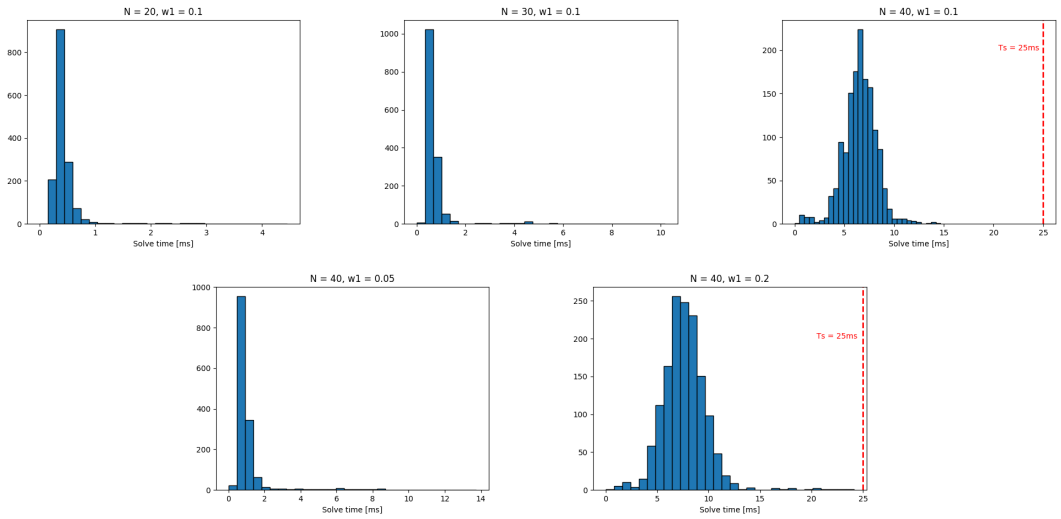


Figure 4.17: Solve time histograms for the different configurations

4.3 Complete System Results

4.3.1 Setup

A similar track was used for testing of the joined algorithms as a combined system, as shown on the left panel of Figure 4.18. It is composed of 73 cones on each side with minimum distance between two consecutive cones of 1.97m and a maximum of 5.02m. The centre line and track boundaries provided to the MPC controller were calculated by the Boundary Estimation component and, similarly to the previous section, the noise in measurements of the state of the vehicle and the the control inputs were assumed to result into an ideal output according to the vehicle model.

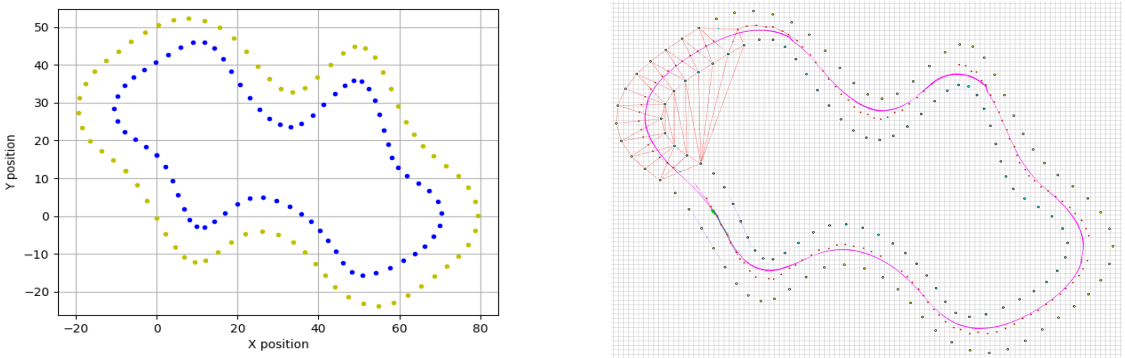


Figure 4.18: Track cones and driven path

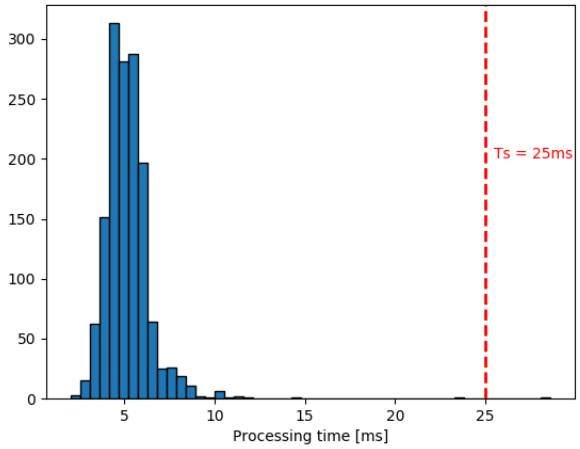


Figure 4.19: Solve time histogram for the complete system

4.3.2 Solve time and track completeness

The complete system was evaluated based on the time needed to process the track sections by the Boundary Estimation algorithm and to compute the control inputs needed to drive through the track by MPCC, and whether the car was able to traverse the complete track without crossing the boundaries.

For the development of the centre line, 15 cone frames were used and data on the processing times for the Boundary Estimation component can be found in Table 4.2. Figure 4.19 shows a histogram of times needed for the complete system to compute the reference path and boundaries and the control inputs for the vehicle for a single lap around the given track. 99.93% of the solve times are below the sampling time of 25ms, with the maximum being 28.631ms.

Lastly, the right panel of Figure 4.18 shows the fully mapped track with the centre line described by a list of discrete coordinates, and the path the modelled vehicle drove. While the divergence from the centre line is small, it clearly shows that a racing line was followed rather than simply the centre line given to the MPCC controller. The track was driven successfully without exceeding boundary conditions and stopped after a single lap. The image was captured from the ROS visualisation software package with data inputs coming directly from the operation of the software package in real time, thus accurately representing the actual state of the program.

5 DISCUSSION

The main goal of this thesis was the development of a sub-system of a complete autonomous racing system for the Formula Electric Belgium team that would provide the control element of the car. This control element should accept the input of cone locations and positional data about the car and output control commands for acceleration and steering angle. It should also provide meta-data such as lap information and provide safety checking like the ability to recognise when the car has left the track. All these features must be available on a previously unknown track, which is simultaneously being mapped while the car is racing it.

This presented the following challenges to the system which acted as design goals:

1. The system must process all required data in a single iteration in a short enough time to remove the possibility of the car moving outside of the track before the system has an opportunity to react. The top speed of the Umicore Eclipse and track dimensions rules of the Formula Student competitions led to a targeted refresh rate of 40Hz.

2. The track must be mapped section by section, as the number of visible cones is decided by the physical characteristics of the vision sensors placed on the car, along with the operational details of the software packages communicating with the sensors. The controller must operate efficiently despite limitations on the known track, and ensuring the car is within the known track boundaries should operate at all times.

3. Rather than focusing on tightly pursuing the centreline, the controller should instead prioritize the progress along the track to improve the racing performance of the car. This should be possible even while the track map is only partially complete.

In order to satisfy these goals, the overall control system was split into two main components. The first one is the Boundary Estimation, or path-finding, component for which discretization and graph search algorithms for finding the centreline were used. This is a well-proven method which provides the needed processing optimizations to allow for achievement of the required mapping speed. The second component is aimed at path following, where the to-be-followed reference path and track boundaries are developed by the above mentioned Boundary Estimation algorithm, and the Model Predictive Contouring Control method is exploited for this purpose. This is a robust algorithm that works on a basis of solving an optimization problem, which in this application is designed such that the vehicle attempts to pursue a path that maximizes the progress along the given reference line. In order to minimize the solve time of this component, the Optimization Engine framework is used.

The path-tracking and path-following system presented in this paper serves as a starting point for the future FEB teams for the task of creating a completely autonomous racing software. It contains a functional implementation for every step needed, and can be built up towards a more robust and efficient controller. In this section, the achieved results and possible future improvements of the algorithms are discussed.

5.1 Boundary Estimation

The Boundary Estimation, or path mapping, component of this thesis sought to map the track in the minimal possible time, with the maximal possible robustness and producing a centreline that as accurately as possible describes the centreline. Specifically, efficiency was vital in order to leave the most processing cycles possible available for the MPCC component, the system should be robust to errors in cone location, unusual track contours and failed cone boundary classifications (misidentifying the colour and thus detecting cones as belonging to the wrong boundary) and accuracy to facilitate the best possible performance from the MPCC component.

Overall, the results in terms of processing efficiency were excellent, outstripping requirements with average timings consistently in the tens to hundred of microseconds and only exceeding a millisecond if the algorithm was presented with artificially difficult mapping scenarios. The choice of highly efficient graph search algorithms and discretization through Delaunay triangulation allowed for a significant reduction of possible travel points to allow a potentially highly dense path-finding problem to be simplified considerably. This, in combination with careful selection of the programming language, data structures and algorithm design, allowed the solution to achieve the present degree of robust track mapping at a speed that would be viable on the car while racing.

Robustness was more of a challenge, particularly given the high degree of uncertainty in how to design the system with respect to its final usage on the car. The lack of practical data and testing opportunities to see typical numbers of cones in a single batch, frequency and features of misclassification of cones and how the reach of the car's sensors influences cone visibility and the frames that should be constructed, meant that generalised approaches needed to be taken which may be highly susceptible to failure in case the assumptions made are inaccurate.

Examples of robustness concerns include:

- The vision of the car permitting visibility of the cones along only one boundary rather than both. While the system will attempt to balance the frame with approximately equal quantities of left and right boundary cones, this is reliant on accurate classification of cones, and relies on receiving some cones of both boundaries. In the case that either of those states fail, it is possible that the result will either be a flawed track mapping or a failure to map, in which case the car will reach out-of-boundary state quickly and stop. Considerable real-life testing and further development should allow for a fuller understanding and mitigation of these risks
- Attempts were made throughout the system to reduce or remove the reliance on successful boundary classification by the stereo camera and image processing systems that make up part of the full autonomous racing system. Efforts included the addition of alternative path costing features which are not classification reliant, and the minimal use of boundary knowledge in constructing path mapping algorithms. Additionally, algorithms were developed to artificially classify cones using positional information relative to other successfully classified cones. Satisfactory replacements were not found at all stages of the mapping process though, such as when finding a section exit, which is a crucial stage in mapping a track section. Given the lack of any real-life knowledge on the typical length of frames and therefore what metrics could be relied upon for successfully localizing a section exit, boundary classifications were used instead. As a result, the system should now still be described as classification-dependent.

- The lack of extensive real-life testing raises concerns over the robustness of the path costing parameters and their weights. As there is a risk of a particularly poorly selected path, such as one that directs the car to move towards the boundary, influencing the following sections and leading to a broadly flawed track mapping, further experimentation with possible parameters to reduce the probability of this event would be beneficial.

Accuracy of the final centreline produced is satisfactory given the compromises made in graph search complexity in favour of faster mapping. The mapping quality improves when frames can be constructed from more cones as the triangulation tends to be more densely packed and therefore provides more traversable nodes in viable locations for forming part of the final centreline. There is a trade-off though, as there are robustness concerns once a frame becomes large enough that some of the simpler operations such as distance checks cease to act as accurate metrics for decision-taking due to track contours. Generally, the compromise is that larger frames allow for higher accuracy while smaller frames reduce the risk of error, as large track features like sharp turns are less likely to be completely included and increase the risk of assumptions being false (like assuming the exit point of a section is the furthest point from the entrance).

A factor negatively impacting accuracy is linearisation along the final centreline. It was chosen to maintain a discrete centreline made up of coordinates along the path, and linearly interpolate coordinates between them. While this is an efficient solution, it provides a degree of error which could lead to unexpected behaviour in extreme circumstances - such as forcing the car to move out of boundary in the case that a linearised connection between two coordinates is large enough to cause it to attempt to 'cut' a corner. A degree of smoothing would be beneficial to reduce this error.

The overall Boundary Estimation component is found to be adequate for the problem, with better than required efficiency and acceptable accuracy, supporting the choices of discretization and path finding algorithm. The robustness of the system is reasonable, but would benefit from considerable further testing and continued development (for which some possible improvements will be discussed in Section 5.4) taking advantage of the processing time headroom still available.

5.2 MPCC

The purpose of the path-following element of the system was to calculate the control inputs to the vehicle such that it would drive a racing line. To this end, the Model Predictive Contouring Control method was used. A model of the vehicle, a cost function and a number of constraints were defined in this method and used to generate a solver through the Optimization Engine tool. The problem was designed such that the race car would attempt to progress along the centreline calculated by the Boundary Estimation element as quickly as possible, rather than trace the centreline closely. Most importantly, the solve time of this component had to be kept at less than 25ms.

The results of the implemented control algorithm shown in the previous chapter demonstrate its performance, based on the appropriate criteria, for different configurations in a simplified environment. It can be observed that the solve time increases with prediction horizon. The main consequence of this is that with shorter prediction horizon the controller sees a shorter segment of the reference line and hence cannot adapt to sharp turns as early on the track as would be desirable. The configuration chosen as most suitable for the complete system was one with prediction horizon $N = 40$, with time step of 0.05s resulting in a look-ahead of two seconds, and with tracking accuracy weight

$\mathbf{w}_1 = 0.1$. This lead to a solution with acceptable balance between the length of the path taken to traverse the track and the solve time needed for the optimizer to compute the control inputs for the race car. The outcome is regarded as adequate for the application, considering the set requirements.

Making use of the Optimization Engine framework for this implementation has provided the needed efficiency of solving the optimization problem and is highly recommended for future use, especially with the newly developed generation of a ROS package, which was only released at a late stage of the implementation process of this thesis. However, it is important to note that more time needs to be invested into learning about the underlying concepts and methods of OpEn in order to be able to better tune the generated solver and fully exploit the capabilities of the tool.

Similarly for the vehicle dynamics, as this is a crucial component of the MPC controller, it is advised to investigate in more depth the pros and cons of the different models and find a more fitting and more accurate version for the problem at hand, putting emphasis on obtaining the needed parameters for more reliable performance.

Additionally, one of the most critical weaknesses of the design of the optimization problem is that it is possible that the current track boundary constraint implementation gets violated in cases with sharp turns without the car actually being outside the track. This is due to the calculated tangent lines crossing the track such that the car's position or a reference point's position lies just outside of the permitted area. Consequently, as can be expected, the solver cannot find a viable solution for this situation. In order to minimize the chance of this happening, the prediction horizon had to be kept at $N = 40$.

All in all, the chosen approach is found acceptable for this problem as defined above with respect to the required processing time and the driven line. Suggestions for its further improvements are discussed within Section 5.4.

5.3 Complete System

The targeted goal of the complete control system was that it should be possible to present it with an unknown track, begin driving it immediately after mapping the first section and continue to do so until it completes the lap, which it should be aware of. It must also be capable of registering out of boundary states, and driving a racing line rather than simply pursuing the centreline of the track. These features must be provided all while requiring a sufficiently small computational load to permit a processing time of 25ms or less for a single refresh of the control outputs.

For these goals, the system can be judged to be satisfactory, with fewer than one percent of control updates exceeding sampling time and by a maximum of less than 4ms, and majority of control updates having a solve time between 2 and 10ms. This is within the target parameters and thus the efficiency requirement can be said to have been met.

5.4 Future Work

As this thesis should be used primarily as an initial setup to build upon by the FEB team, this section is essential to explain the next steps needed to take in order to improve the current implementation.

Cone colour classification

As described in Section 5.1, the system cannot currently be described as cone classification independent despite the desirability of this. Finding robust but generic solutions to many track-mapping problems is difficult without the option to distinguish between left and right boundaries, but the advantage in terms of avoiding error states is considerable. The disadvantage is that algorithms that do not rely on classification are typically more computationally intensive, as fewer assumptions can be made resulting in more calculations.

Centreline smoothing

In order to reduce the linearisation errors as described in Section 5.1, a smoothing algorithm to be applied across the extracted centreline could improve the resulting centreline measurably. One method that was repeatedly discussed for doing this was the use of cubic spline interpolation. This is a set of piece-wise third order polynomials which pass through a set of given points. The issue with implementing this was the fact that splines could be constructed in a single direction (positive or negative) along an input axis so any attempt to construct a spline across a track feature that loops back on itself caused a faulty output. While solutions like the detection of direction shifts resulting in the linking of a set of splines is possible, there was insufficient time to add it as a feature.

Vehicle model, cost function and constraints

A number of improvements can be introduced in the controller implementation. By replacing the kinematic bicycle model with the dynamic one, or a combination of the two as proposed by Kabzan et al. (2019b), in order to account for the forces acting between the tyres and the driving surface, higher driving velocities can be achieved while keeping the vehicle safe. Consequently, the constraints for the tyre forces according to the friction ellipse, a property of a tyre, can be imposed, which increases control over the vehicle's velocity. The parameters needed for the dynamic bicycle model must be obtained experimentally.

As mentioned in Section 5.2, there are occasions in which the definition of the boundary constraint causes the solver to not find a viable solution in the given time or number of iterations. It is essential to improve this aspect by defining such constraint that is able to more closely follow the track boundaries. One possible solution for this is implemented also by Kabzan et al. (2019b), who formulates the track constraint by locally approximating the track by circles centred on the middle line. This allows for use of longer prediction horizon and hence more optimal line can be achieved.

Last but not least, the tracking accuracy cost can further be improved by using the contouring control problem formulation as explained in (Lam et al., 2010), where arc-length parametrisation of spline curves is utilized to define the contouring and the lag error, providing a more accurate cost for the goals of the controller. This would, however, require the reference line provided to MPCC to be defined as a spline curve, rather than a discretised path.

Practical testing

It is impossible to assess all the qualities and pitfalls of the current implementation without having tested it on the actual formula and as a part of the complete driverless system. This should not only be done for the evaluation of performance, but also for tuning of the weights of the costs in developing the centreline and of the costs in driving around the track. It is also important to calculate the delays starting from the measuring of the car's state and capturing of the surrounding cones and ending with the adjustment of the control inputs causing the car's state to change. Once this is possible, more detailed requirements can be identified.

Formula Student challenges

The Formula Student driverless competition involves multiple challenges which have not yet been tackled in this system. The target throughout the design was developing an autonomous control system which could map and control the Umicore Eclipse in laps around an unknown track, which describes the Trackdrive and Efficiency event. Other driverless events include the Acceleration and Skidpad events, whose challenges have not been tackled. For example, the Skidpad event is based on a track of two overlapping circles where the car is expected to choose the left or right circle according to the lap number. This kind of lap-based path selection and storage of two viable paths is not considered in the system developed so far, and must be the target of future work.

6 CONCLUSION

For the first time, Formula Electric Belgium team decided to participate in the driverless category of the Formula Student competition.

BIBLIOGRAPHY

Abd Algfoor, Z., Sunar, M. S., and Kolivand, H. (2015). A comprehensive study on pathfinding techniques for robotics and video games. *International Journal of Computer Games Technology*, 2015.

Andersson, J. A. E., Gillis, J., Horn, G., Rawlings, J. B., and Diehl, M. (2019). CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36.

Bhoraskar, A. and Sakthivel, P. (2017). A review and a comparison of dugoff and modified dugoff formula with magic formula. In *2017 International Conference on Nascent Technologies in Engineering (ICNTE)*, pages 1–4. IEEE.

Bodart, G. (2018). Development of an efficient trajectory optimization approach for a formula student electric race car. Master's thesis, KU Leuven.

Botta, M., Gautieri, V., Loiacono, D., and Lanzi, P. L. (2012). Evolving the optimal racing line in a high-end racing game. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 108–115. IEEE.

Bréchet, T., Camacho, C., and Veliov, V. M. (2014). Model predictive control, the economy, and the issue of global warming. *Annals of Operations Research*, 220(1):25–48.

Camacho, E. F. and Alba, C. B. (2013). *Model Predictive Control*. Springer Science & Business Media.

Cardamone, L., Loiacono, D., Lanzi, P. L., and Bardelli, A. P. (2010). Searching for the optimal racing line using genetic algorithms. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 388–394. IEEE.

Curinga, F. (2017). Autonomous racing using model predictive control.

Demyen, D. and Buro, M. (2006). Efficient triangulation-based pathfinding. In *Aaai*, volume 6, pages 942–947.

Fredlund, J. K. and Sulejmanovic, K. S. (2017). Autonomous driving using model predictive control methods.

Kabzan, J., Hewing, L., Liniger, A., and Zeilinger, M. N. (2019a). Learning-based model predictive control for autonomous racing. *IEEE Robotics and Automation Letters*, 4(4):3363–3370.

Kabzan, J., Valls, M. d. l. I., Reijgwart, V., Hendrikx, H. F. C., Ehmke, C., Prajapat, M., Bühler, A., Gosala, N., Gupta, M., Sivanesan, R., et al. (2019b). Amz driverless: The full autonomous racing system. *arXiv preprint arXiv:1905.05150*.

Kong, J., Pfeiffer, M., Schildbach, G., and Borrelli, F. (2015). Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1099. IEEE.

Lam, D., Manzie, C., and Good, M. (2010). Model predictive contouring control. In *49th IEEE Conference on Decision and Control (CDC)*, pages 6137–6142. IEEE.

Liniger, A., Domahidi, A., and Morari, M. (2015). Optimization-based autonomous racing of 1:43 scale rc cars. *Optimal Control Applications and Methods*, 36(5):628–647.

Mattingley, J., Wang, Y., and Boyd, S. (2010). Code generation for receding horizon control. In *2010 IEEE International Symposium on Computer-Aided Control System Design*, pages 985–992. IEEE.

Novi, T., Liniger, A., Capitani, R., and Annicchiarico, C. (2019). Real-time control for at-limit handling driving on a predefined path. *Vehicle system dynamics*.

Pacejka, H. (2012). *Tire Characteristics and Vehicle Handling and Stability*, pages 1–58.

Polack, P., Altché, F., d'Andréa Novel, B., and de La Fortelle, A. (2017). The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 812–818. IEEE.

Robotin, R., Lazea, G., and Marcu, C. (2010). Graph search techniques for mobile robot path planning. In *Engineering the Future*. IntechOpen.

Shanbi, W., Yi, C., and Penghua, L. (2012). Distributed model predictive control of the multi-agent systems with improving control performance. *Journal of Control Science and Engineering*, 2012.

Sopasakis, P., Fresk, E., and Patrinos, P. (2020). OpEn: Code generation for embedded nonconvex optimization. In *IFAC World Congress*, Berlin, Germany.

Verschueren, R. (2014). Design and implementation of a time-optimal controller for model race cars. Master's thesis, KU Leuven.

Yakub, F., Abu, A., Sarip, S., and Mori, Y. (2016). Study of model predictive control for path-following autonomous ground vehicle control under crosswind effect. *Journal of Control Science and Engineering*, 2016.

Zeilinger, M., Hauk, R., Bader, M., and Hofmann, A. (2017). Design of an autonomous race car for the formula student driverless (fsd). In *Oagm & Arw Joint Workshop*.

Zhang, B., Lin, F., Zhang, C., Liao, R., and Wang, Y.-X. (2020). Design and implementation of model predictive control for an open-cathode fuel cell thermal management system. *Renewable Energy*.

FACULTY OF ENGINEERING TECHNOLOGY
CAMPUS GROUP T LEUVEN
Andreas Vesaliusstraat 13
3000 LEUVEN, Belgium
tel. +32 16 30 10 30
fax +32 16 30 10 40
fet.groupt@kuleuven.be
www.iiw.kuleuven.be