Final Report – GBDP

TBRe-AI Control: Master Controller

Candidate Number: 13757

Word Count: 6042

CONTENTS

List of Figures

List of Acronyms

Acronym	Meaning
FS-AI	Formula Student Artificial Intelligence
TBRe-AI	Team Bath Racing Electric Artificial Intelligence
GBDP	Group Business Design Project
MPC	Model Predictive Controller
SR	System Requirements
SSR	Sub-system Requirements
RK-4	4^{th} Order Runge Kunta Method
NLP	Nonlinear Programming
MIMO	Multiple Input Multiple Output

1 INTRODUCTION

1.1 Formula Student Artificial Intelligence (FS-AI) and Background

Each year the Institution of Mechanical Engineers hosts competitions for university students to race cars against one another. Events include the autonomous racing competition; whereby students must design and build a fully autonomous electric race car. [1]

All events are performed in controlled environments reducing the need for safety measures such as obstacle detection and avoidance. In addition, all cars must have manual intervention capabilities to stop the cars in emergency scenarios.

1.2 Brief and Scope

Team Bath Racing Electric Artificial Intelligence (TBRe-AI) is a team within the University of Bath that competes in the FS-AI competition. TBRe-AI assigned work for the TBRe-AI GBDP teams to complete for implementation on the 2022 car. The work packages have been broken down into subsystems dependent on the design of the 2022 car.

As a result, each GBDP team member was given a subsystem to design and develop; with a goal of producing the best technical piece of work possible. These systems would need to interface with the existing car and any other subsystems being developed simultaneously. The TBRe-AI team would then decide whether to use the work done by the GBDP team or use existing/other solutions. Thus, the goal of the GBDP is to produce a working and justified subsystem of the 2022 car that would in turn perform well in competition.

1.3 Autonomous System

A full autonomous vehicle system is made up of several constituent parts working together to allow for full control of the vehicle. As part of the design brief from the TBRe-AI team, a full breakdown of the different functions was provided, Figure 1.1. The assigned sub-systems would need to integrate with any of the other given sub-systems.

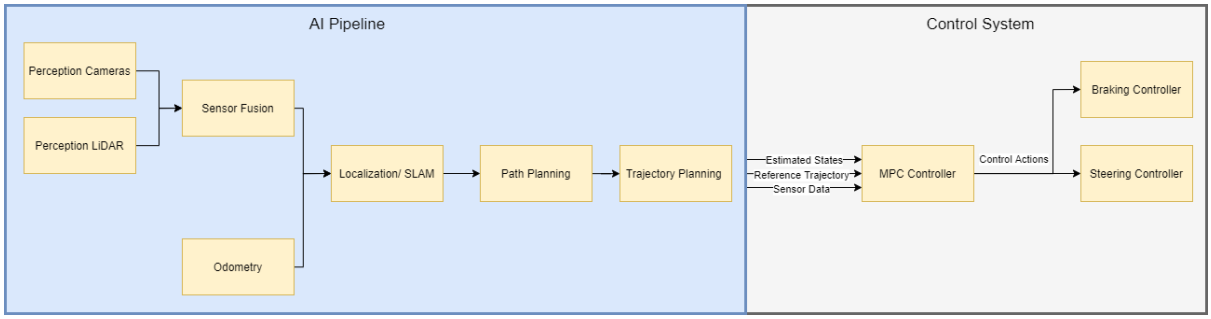


Figure 1.1 Full Autonomous System Breakdown

1.4 Control System and Interfaces

The control system had to follow the format provided in Figure 1.2; whereby, low level controllers (slave controllers) receive the control actions calculated by the MPC (Model Predictive Controller) controller (i.e. master controller). The detailed design of the separate sub-systems will determine the details of the data being transferred between them. As the subsystems are being designed in parallel only basic information about the interfaces can be known before design of the MPC controller.

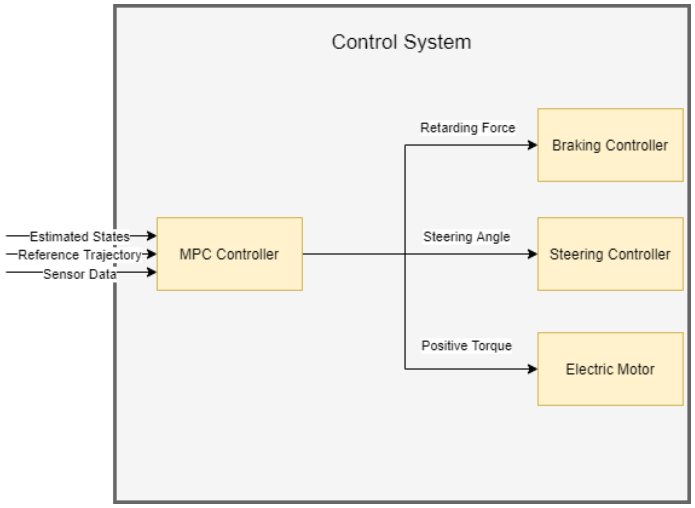


Figure 1.2 Interfaces of MPC controller

1.5 MPC Controller and Control Problem

This report covers the design of the master controller for the vehicle. As part of the brief a MPC controller was selected for its ability to optimize and constrain control problems as well as having a predictive capability. Figure 1.3 Generic MPC block diagram for autonomous demonstrates the generic workings for a MPC controller in this application.

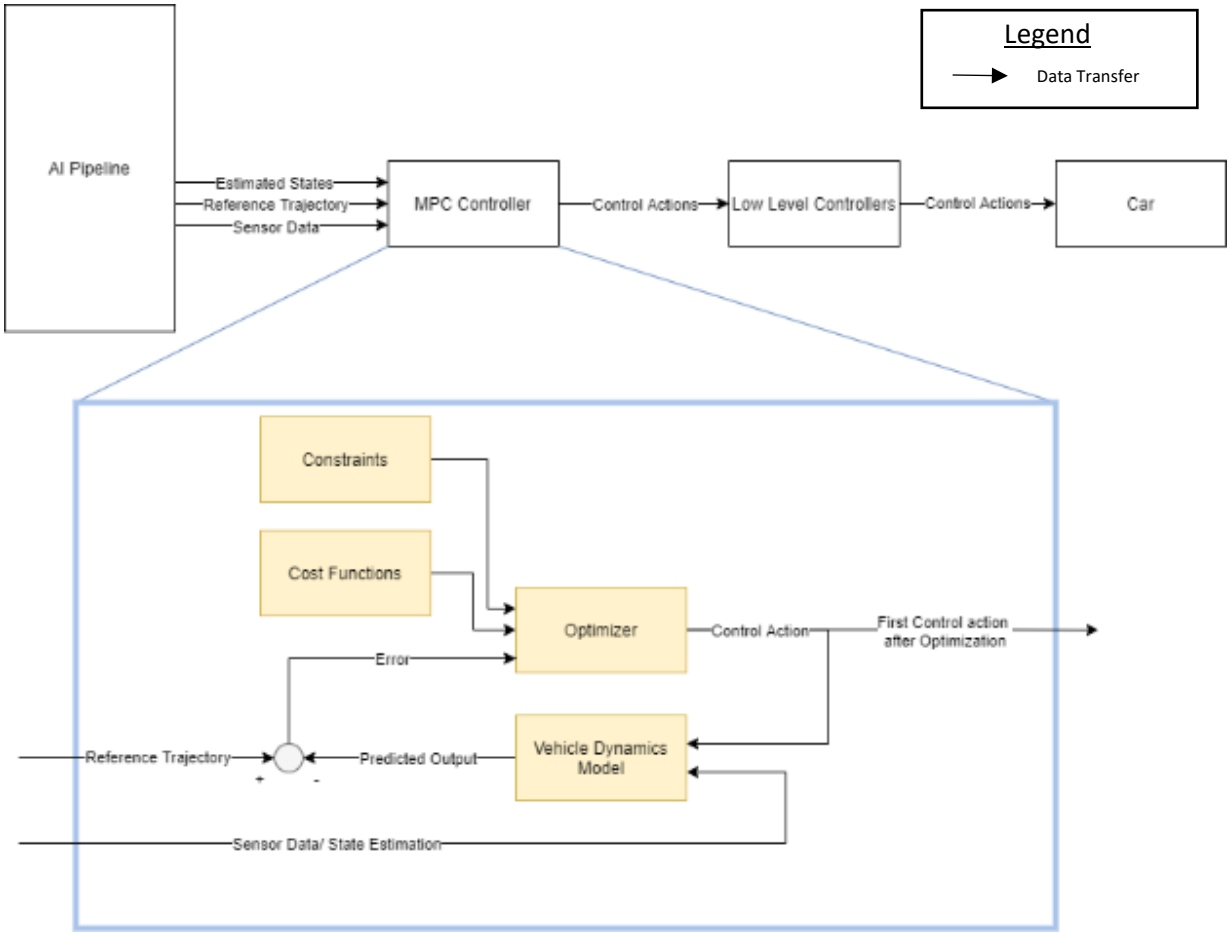


Figure 1.3 Generic MPC block diagram for autonomous path following

There are effectively two types of FS-AI events the car must compete in. Track based events (skidpad, autocross, trackdrive), which require the car to traverse a track consisting of straights and corners; and acceleration events, which require the car to traverse a straight track.

The general control problem is similar for both track and acceleration events. The goal of the controller is to provide control actions to the steering, braking system and motor that allows the car to follow the reference trajectory provided by the AI pipeline as fast as possible while not exceeding the track limits. The format of the reference trajectory is a set of waypoints in an XY coordinate plane, of which the vehicle must follow. For the first lap of the track event and acceleration events the full set of waypoints are not known so only a small number can be sampled based on the range of the sensors of the car.

2 CUSTOMER NEEDS ANALYSIS

2.1 Control System Requirements

Using the design brief covering the competition and work packages a set of system requirements (SR) were developed for the control system (Appendix A). To best capture the voice of the customer a summary of the most basic wants and needs was created, Table 2.1. In this application TBRe-AI represent the primary stakeholder and customer.

The full system requirements are broken down based on the type of the requirement specified, with each requirement given a priority and a verification method.

Table 2.1 Initial Customer Wants and Needs

Number	Wants and Needs
1	Control System shall adhere to Formula student Rules.
2	Control System should perform as best as possible in competition.
3	Control System shall interface with the car.
4	Control System shall control all steering, braking and motor torque of the car.

2.2 Preliminary Research on MPC Autonomous Racing Applications

There exist several papers and resources on the design of MPC controllers for use on autonomous vehicles. For example, MATLAB has tutorials and functions for implementing MPC controllers on Vehicle Lane Following [2]. Unfortunately, many of the papers and resources for MPC autonomous systems are not racing based.

The design of an MPC controller for racing is quite different to that of an MPC used for normal driving conditions. This is due to the changing objectives found in racing (i.e. corner types and track position) as well as the increased complexity in vehicle modelling when running a vehicle at high speeds.

AMZ Driverless is the FS-AI Team associated with ETH Zurich and are one of the top performing FS-AI teams in the world. Their work provides a good insight into how an MPC controller can be used in a racing scenario [3] and are used as a datum during design of the controller.

2.3 MPC Sub-System Requirements/ Product Design Specification

Using the System Requirements and informed by the research conducted into MPC design, a set of sub-system requirements (SSR) were formulated (Appendix B). The link between the system requirements and each requirement is clearly shown as well as a description of the verification method. As the system needs to comply with formula student rules; specific rules that need to be adhered to have been referenced where appropriate and form the basis of some of the requirements.

To ensure that the core functional requirements of the sub-system are appropriate, a House of Quality analysis matrix was conducted (Appendix C). The matrix ensured functional requirements were appropriate when compared with a selection of criteria based on the stakeholder needs.

3 System Development and Design Process

3.1 MPC Concept System Development

MPC controllers can be broken down by the Linear or Non-Linear optimization problems. Non-Linear problems can be approximated to linear ones by computing Jacobians about a stationary point. This can in some cases be beneficial as it allows the solution to be solved more quickly.

However, modern non-linear optimization solvers have improved to the point whereby the programming problems can be non-linear and still be efficiently solved [4] with the added benefit of producing a more accurate solution. But the solutions are often not global optimal for the problem and can suffer from instabilities.

For these reasons and ease of implementation a non-linear MPC architecture shown in Figure 3.1 was chosen for use. It is a direct development from a generic MPC shown in Figure 1.3 but now includes the relevant functions for the autonomous race-car application.

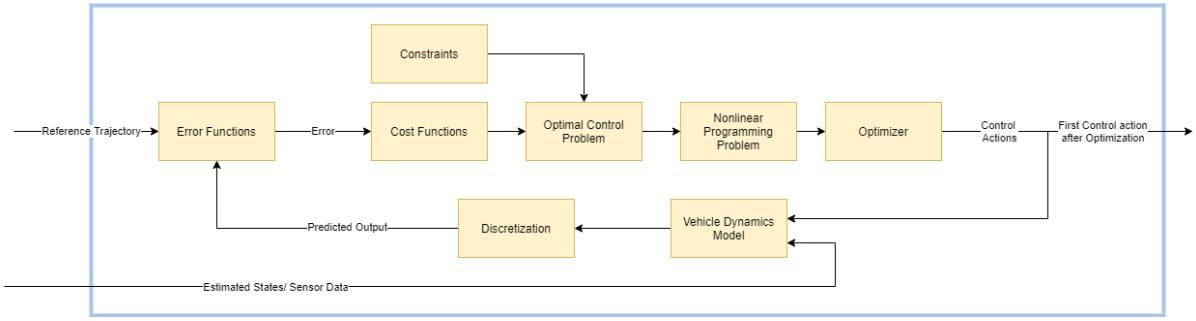


Figure 3.1 Non-Linear MPC Controller

3.2 Design Process

A design process depicted in Figure 3.2 was used for its ability to develop optimal designs for the key functions while also ensuring a functional output. This method reduces the possibility of the built MPC controller being unsuitable for the application as the underlying key functions are unsuitable. The key functions highlighted are the vehicle model and the error functions which are interlinked with the discretization method and the constraints.

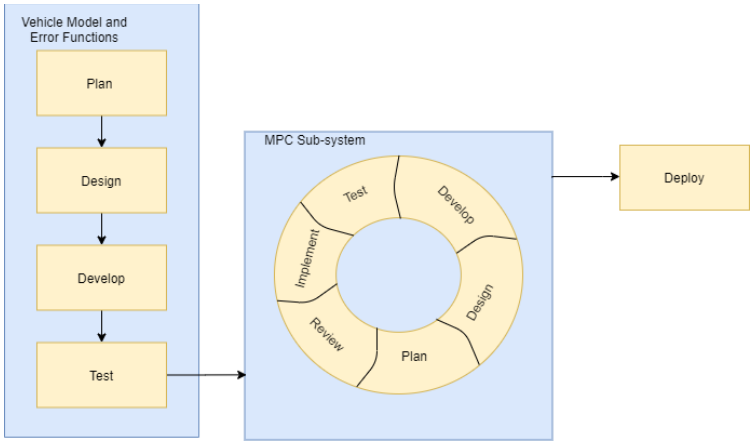


Figure 3.2 Design Process for MPC Controller

3.3 Programming Language

MATLAB was chosen as the programming language to develop the MPC controller because of the useful inbuilt functions, proprietary toolboxes, debugging tools and testing environments that allow for fast development times. MATLAB also offers C code generation tools so any code produced can then be used on the car which uses ROS [SSR-20].

Other options were to develop directly in C or in Python. However, it was deemed these would be less suitable due to increased development times and in the case of python computational inefficiency as code cannot be readily converted to C.

4 Vehicle Model and Error Functions Developments

4.1 Vehicle Dynamics Model and Discretization

The controller uses the current state of the vehicle calculated from the AI pipeline and propagates them forward along with the control inputs using a discretized form of the vehicle model. The vehicle model needs to be efficient enough to be discretised 10 times per second [SSR – 9] and accurate enough to ensure the car can accurately track the reference trajectory and not exceed cone/track boundaries [SSR-6,7].

4.1.1 Concept Generation

4.1.1.1 Kinematic Bicycle Model

The model [3] [5] [6] in Figure 4.1 is a commonly used vehicle model that accurately describes behaviours of a car at low speeds. It uses the symmetry of the vehicle and only considers in-plane motion about the frame of reference at the centre of gravity and takes inputs of the steering angle δ (rads) and the force F_r (N). States include the position to some reference (X,Y) , the heading angle θ, longitudinal and lateral speeds in the body frame v_x and v_y respectively, and the yaw rate ω. Additional vehicle parameters include: m – mass, Iz – moment of inertia, $L_{f,r}$ – distance from the centre of gravity to the front and rear wheels, respectively

The state equation (2) [3] has been derived in such a way that the steering angle and force rear wheel are the inputs to the equation which reflect the actual control actions of the car. The values $\dot{\omega}$ and $\dot{v_y}$ have been derived using derivatives of $\omega = \frac{\tan(\delta)v_x}{L_f+L_r}$ and $v_y = L_r\omega$ respectively. δ is assumed to be small and the approximations $cos(\delta) \approx 1$ and $tan(\delta) \approx \delta$ are made. The $\dot{\delta}$ can be approximated by taking the gradient between the current and the previous steering angle.

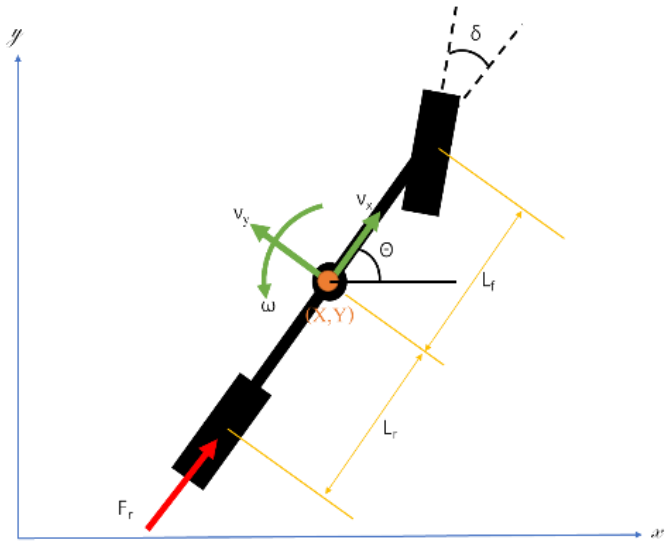


Figure 4.1 Kinematic Bicycle Model Schematic

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \cos(\theta) - v_y \sin(\theta) \\ v_x \sin(\theta) + v_y \cos(\theta) \\ \omega \\ \frac{F_r}{m} \\ \left(\dot{\delta} v_x + \delta \dot{v_x}\right) \frac{L_r}{L_r + L_f} \\ \left(\dot{\delta} v_x + \delta \dot{v_x}\right) \frac{1}{L_r + L_f} \end{bmatrix} \quad (2)\ [3]$$

4.1.1.2 *Dynamic Bicycle Model*

The model in Figure 4.2 [3] [5] [6] is an extension of the model in Figure 4.1 by including the lateral tire forces on the front and rear wheels F_{ry} and F_{rx}. It models the slip associated with the tires α and has the same input parameters and states as in (2). The tire model used (4) is a simplified version of the Pacejka Tyre Model [7] and only includes the lateral forces exerted on the tyres. The terms D, C and B are experimentally determined coefficients.

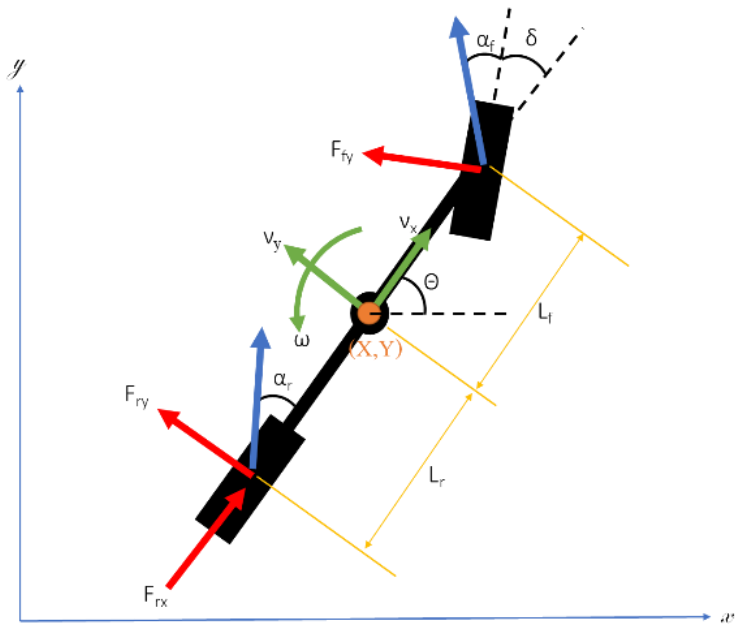


Figure 4.2 Dynamic Bicycle Model Schematic

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \cos(\theta) - v_y \sin(\theta) \\ v_x \sin(\theta) + v_y \cos(\theta) \\ \omega \\ \frac{1}{m}(F_{rx} - F_{fy} \sin(\delta)) + v_y \omega \\ \frac{1}{m}\left(F_{ry} + F_{fy} \cos(\delta)\right) - v_x \omega \\ \frac{1}{I_z}\left(F_{fy} * \cos(\delta) * L_f - F_{ry} L_r\right) \end{bmatrix} \quad (3)$$

$$F_{fy} = D_f \sin\left(C_f \, arctan\left(B_f \alpha_f\right)\right) \tag{4}$$

$$\alpha_f = -\arctan\left(\frac{\dot{\theta} L_f + v_y}{v_x}\right) + \delta \tag{5}$$

$$F_{ry} = D_r \sin\left(C_r \, arctan\left(B_r \alpha_r\right)\right) \tag{6}$$

$$\alpha_r = \arctan\left(\frac{\dot{\theta} L_r - v_y}{v_x}\right) \tag{7}$$

$$D_{f,r} = \frac{L_r}{L_r + L_f} * m * 9.81 * D \tag{8}$$

4.1.1.3 Initial Model Evaluation

A Pugh matrix (Appendix D) was used to evaluate the models. The datum chosen was the model used in the AMZ driverless car [3]. This car uses a combination of the dynamic and kinematic models based on the velocity of the vehicle. The criteria developed is based on system requirements and project timeframe; with implementation time also considering risks that may hinder development. The matrix demonstrated the dynamic model appears to be superior aside from problems arising at low velocities due to the slip angle formulation (5)(7). These issues are the main reason for the combined model used in the datum.

4.1.2 Model Development and Testing

To better understand the properties of the Dynamic and Kinematic models a set of simulations were performed across various inputs. Euler and 4th-Order Runge Kunta methods (RK-4) were used to discretize the models and a range of time steps were used to compare the accuracy of the models against approximate continuous versions of the two models. The continuous models have been derived by taking very small timestep RK-4 simulations.

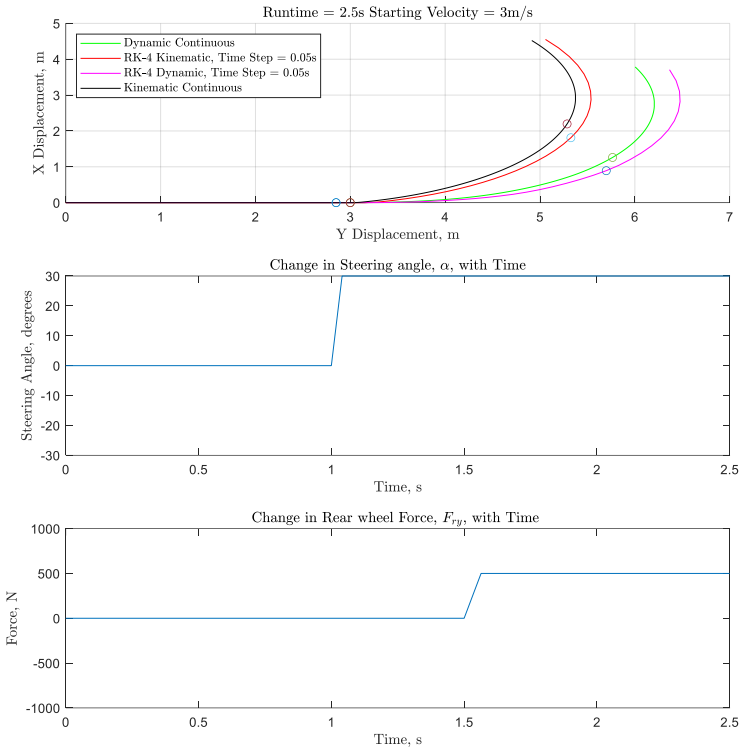


Figure 4.3 Sample of Test Simulations for Dynamic and Kinematic Bicycle Models

Figure 4.3 is taken from various tests carried out. The first graph shows the projected path of the different models with the car positions at second intervals given by the circles on the lines. The remaining graphs show the inputs sent to the model. Appendix E contains additional samples from tests performed. In addition, the run time for each of the methods was measured.

It was found that below 1 m/s the Dynamic models no longer behave as expected due to the formulation of the slip angle and the Kinematic models likely produce more accurate solutions. While the RK-4 Kinematic model with 0.05s time step provides a good trade-off between accuracy and speed being only slightly slower than the Dynamic RK-4 with 0.05s time step, which becomes less important at low speeds.

The RK-4 Dynamic model with 0.01s time step most closely follows the Continuous Dynamic model at high speeds and cornering. However, with 0.01s Time Step the Dynamic Runge is over twice as slow as the RK-4 Kinematic model with 0.1s time step.

The RK-4 Dynamic model with 0.05s time step proved to be the best compromise between accuracy and computational speed, at speeds above 1 m/s. Below this speed the model becomes inaccurate. This is characterised by its ability to have a smaller turning radius than the Kinematic model.

4.1.2.1 *Kinematic Model with Adjustable Discretization*

A possible way to improve the simulation [6] is to use a Euler discretization with an intentionally large time step, that due to model inaccuracies can closely follow the Dynamic continuous model. As a result, the computation time is reduced, and the car can be run at higher speeds. Using this method several tests were re-run.

Figure 4.4 is a selected sample and demonstrates this method well. Based on initial testing this method does not provide value for these applications as it requires the car to be running at higher speeds to accurately track the Dynamic continuous model and begins to become unstable at time steps greater than 0.1s.

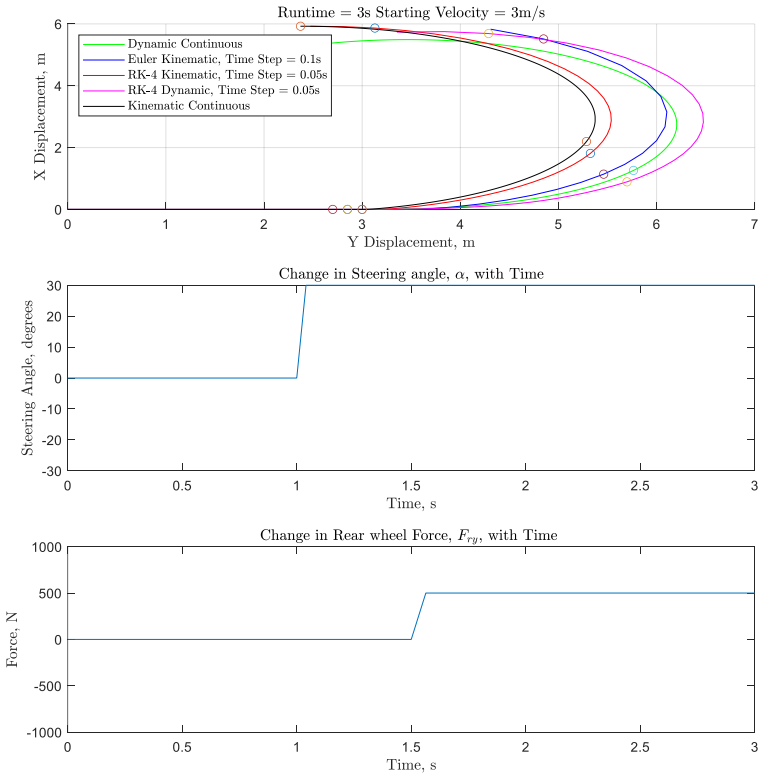


Figure 4.4 Sample of tests performed using methodology from [6]

4.1.2.2 *Combined Dynamic and Kinematic model*

A simpler version of the model used in the datum was developed. It works by evaluating the cars longitudinal velocity at each time step and changing to a kinematic model if below a threshold. The simulation seen in Figure 4.5 used a threshold velocity of 3.5 m/s. This change can be seen in the longitudinal velocity graph where the slope starts to change despite change in input.

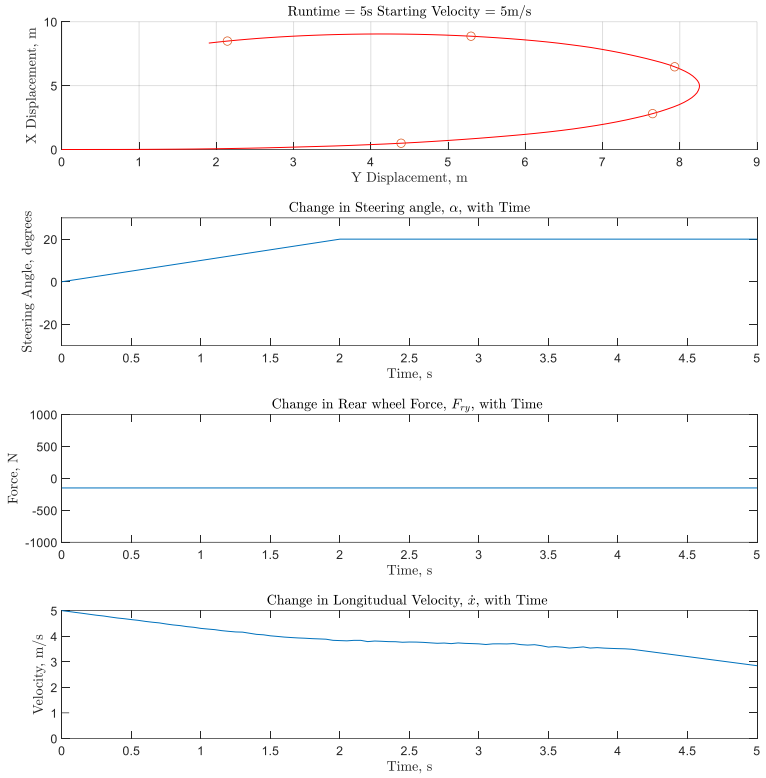


Figure 4.5 Combined Kinematic and Dynamic Model

4.1.3 Concept Review and Future work

The combined dynamic and kinematic model produced the most accurate results across various cornering types and speeds. A Pugh matrix (Appendix G) was used to determine which of the two developed models is most suitable for implementation. The datum was kept the same as the previous Pugh matrix.

The matrix indicates that the combined model should perform best for the project and outperforms the datum due to complexity issues associated with the latter. And so, the combined model is carried forward into the MPC implementation.

Although, the model selected performs best based on the criteria provided it may prove to be inadequate during implementation. If this happens then simplified versions of the model will be used.

4.2 Path Error, Speed Error, and Track Constraints

For the controller to track the reference trajectory it needs to formulate the path error, speed error and track constraints which are all tightly interlinked.

4.2.1 Concept Generation

The following details apply to all concept drawings:

- Blue and yellow markers = cone locations
- Encircled X = centre line way points
- Blue line = centreline approximation, linear splines between the way points
- Dotted line red with arrow = distance minimization
- Dashed line red with angle = heading minimization
- Dot dashed black line is the heading of the car
- Triangle encircled X with car= current car position

4.2.1.1 Concept 1

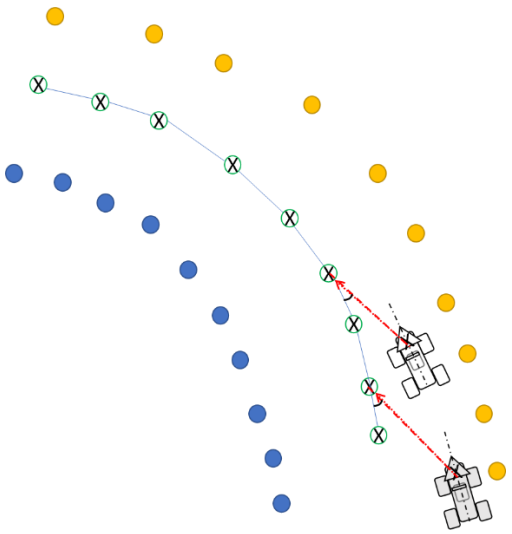


Figure 4.6 Error Concept 1

In concept 1, the controller minimizes the distance and angle to the spline after the next way point and minimizes the difference between the current velocity and the target velocity. Track limits are constrained by limiting the angle between the spline and car depending on the distance from the point.

4.2.1.2 Concept 2

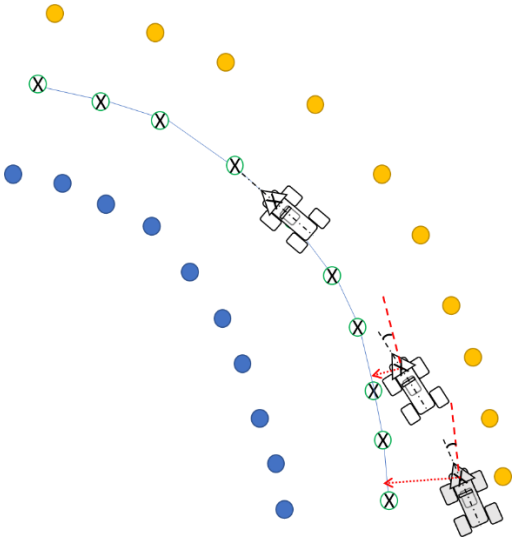


Figure 4.7 Error Concept 2

In concept 2, the controller minimizes the perpendicular distance from spline between the next and previous way point, the angle between current spline (red dashed line) and the heading angle and sets a target velocity at each way point. Track limits are constrained by limiting the perpendicular distance from the centreline.

4.2.1.3 Concept 3

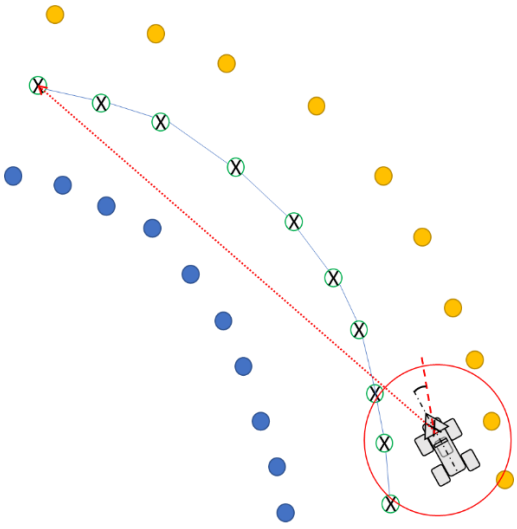


Figure 4.8 Error Concept 3

In concept 3, a circle around car is projected and constrained so that centre line must always pass through the circle, the circle radius is sized so that it does not exceed track boundaries. The angle between current spline (red dashed line) and the heading angle is minimized, as well as the distance from current position to final centre line waypoint. A target velocity is set at the final waypoint.

4.2.1.4 Concept 4

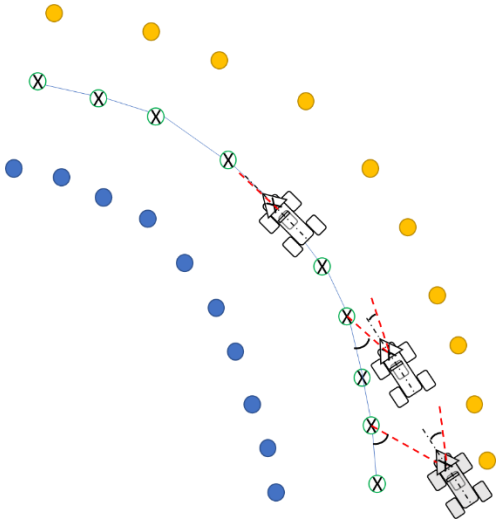


Figure 4.9 Error Concept 4

In concept 4, the controller minimizes the angle between the current spline and the vehicle as well as the angle between the next spline and the car heading angle, with a target velocities at each way point. The car is constrained to the track by limiting the angle between the current splint and the vehicle position, dependent on the distance from the waypoint.

4.2.2 Initial Evaluation

Initial code implementations showed that concepts 2 and 4 required similar code to be implemented. Concept 2 results in the most suitable way to constrain to the track boundaries. This was due to Concept 3 being difficult to realize in code while still being efficient. Concepts 1 and 4 have issues with determining what the correct angle to constrain the car is, as it gets closer to the waypoint.

A Pugh matrix, Appendix H, was used to evaluate the concepts against each other and against a datum. The datum was taken from the same paper [3] from which the datum for the model was chosen. The datum uses high order splines to approximate the centre line and an angle to measure the progress along the path, which it maximises. It then measures the perpendicular distance between the track and the vehicle but uses a minimization function which introduces the need for additional measurement techniques. The criteria for the matrix were derived from the system requirements. The results showed that Concept 2 performs the best and even outperforms the datum for this project.

4.2.3 Concept Development

Concept 2 was developed further, and it was found that determining the distance from the centreline was not efficient enough to be run in real time. This is because the MPC calls this function at every step within the simulation and repeats for multiple optimization steps.

A new method was developed based on testing various calculation times and using ideas from Concept 3, Figure 4.10. This new formulation sets a desired point of which the controller attempts to direct the car towards as in Concept 3. However, to stop the car from exceeding the track boundaries it sets the cones as objects that the car has to avoid. This is computationally much more efficient as the cone's location are already known so the controller can easily set constraints to avoid them.

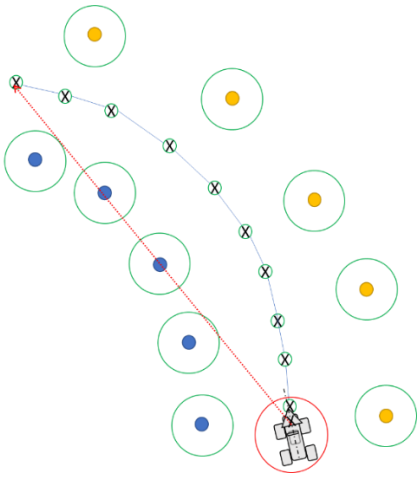


Figure 4.10 Concept 3 Development

4.2.4 Concept Review and Future Work

The main problems with the concept in Figure 4.10 arise when the spacing of the cones increases; the maximum distance between cones on the same side is 5m while the minimum track width is 3m [8]. This may cause the controller to calculate an optimal route that goes between cones exceeding the track boundaries. Reviewing footage of past Formula Student tracks showed it is unlikely that this will be an issue as only on straights are cones spaced further apart than the track width as on the straight the optimal route is easily computed for the controller.

This method has additional benefits as the controller no longer minimises the current heading angle and so is free to orientate itself to the optimal trajectory. The method ends up resembling a similar method used by racing drivers by looking into the apex of the corner while steering [9].

The developed concept 3, Figure 4.10, appears to perform the best and so is implemented first. If this method is deemed unsuitable then earlier concepts will be tested and implemented.

5 MPC Controller Design Development

5.1 Optimization Problems Formulation

5.1.1 Optimal Control Problem

A mathematical formulation for the optimal control problem needs to be derived before code can be written. The problem can be better understood through the visualization in Figure 5.1. The x and u terms denote the relevant states of the system [X Y θ v_x], and control inputs [F_{rx} δ] respectively. x_u denotes the states at any given time subject to u and x^r denotes the reference state.

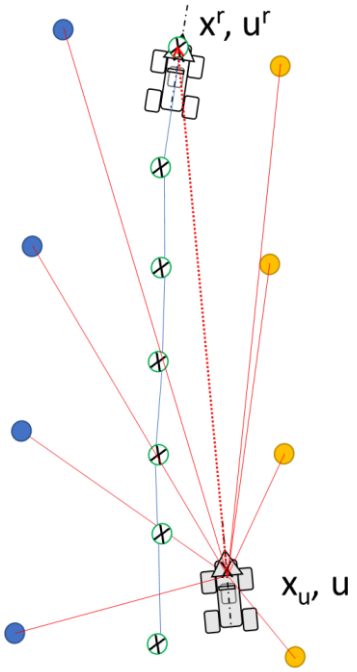


Figure 5.1 Optimal Control Problem Formulation

The loss function (9) for system is then formulated based on the reference and the state of the car at any point. This is normalized and squared to ensure it is always positive and combined with a weighting matrices Q and R for tuning.

$$\ell(x,u) = \|x_u - x^r\|_Q{}^2 + \|u\|_R{}^2 \tag{9}$$

All errors across the entire prediction horizon (how far into the future the model predicts for, an important characteristic of MPC) are calculated (10). The prediction horizon is given by the number of samples N which is determined by the simulation length and discretization.

$$J_N(x,u) = \sum_{k=0}^{N-1} \ell\big(x_u(k), u(k)\big) \tag{10}$$

The full optimization control problem can then be formulated (11). Each step in the objective function, J, is given by the simulation (11b). (11f) gives the constraints for the avoidance of the cones (position X_c,Y_c; radius r_c) by approximating the car to a circle (radius r) and constraining the distance.

$$minimize\ J_N(x_0, u) = \sum_{k=0}^{N-1} \ell\big(x_u(k), u(k)\big) \tag{11a}$$

Subject to (s.t.):

$$x_u(k+1) = f\big(x_u(k), u(k)\big) \tag{11b}$$

$$x_u(0) = x_0 \tag{11c}$$

$$u(k)\ \epsilon\ U, \qquad \forall k\ \epsilon\ [0, N-1] \tag{11d}$$

$$x_u(k)\ \epsilon\ X, \qquad \forall k\ \epsilon\ [0, N] \tag{11e}$$

$$-\sqrt{(X - X_c)^2 + (Y - Y_c)^2} + (r + r_c) \leq 0 \tag{11f}$$

5.1.2 Nonlinear Programming Problem

The nonlinear programming function differs from the optimal control problem in that it consists only of an objective function and inequality and equality constraints. Equation (11b) requires converting into a form for use in the objective function.

So called shooting [10] methods can be used to convert from one problem to the other. Single shooting propagates the prediction over the time horizon and solves subject to the decision variables, it then applies constraints to this propagation. Equations 12 demonstrate the basic principle behind the single shooting where u_N is the control input, $\boldsymbol{x}_0$ is the system initial state and t_k is the time.

$$\boldsymbol{w} = [u_0 \dots u_{N-1}] \tag{12a}$$

$$\min_{w} \Phi(F(\boldsymbol{w}, \boldsymbol{x}_0, t_k), \boldsymbol{w}) \tag{12b}$$

$$\text{s.t. } \mathbf{g_1}(\mathrm{F}(\mathbf{w}, \boldsymbol{x}_0, \mathrm{t_k}), \mathbf{w}) \leq 0 \tag{12c}$$

Multiple shooting lifts the shooting method to more variables making it less nonlinear. This makes the NLP larger but also sparser allowing better convergence. The user can also use an initial guess of the solution to speed up solve times. Equations 13 show how multiple shooting differs from single shooting.

$$\boldsymbol{w} = [\boldsymbol{u}_0 \dots \boldsymbol{u}_{N-1}, \boldsymbol{x}_0 \dots \boldsymbol{x}_N] \tag{13a}$$

$$\min_{w} \Phi(\boldsymbol{w}) \tag{13b}$$

$$s.t.\ \boldsymbol{g_1}(\boldsymbol{w}) = \begin{bmatrix} g_1(\boldsymbol{x}_0, \boldsymbol{u}_0) \\ \dots \\ g_1(\boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}) \\ g_1(\boldsymbol{x}_N) \end{bmatrix} \leq 0, \quad \boldsymbol{g_2}(\boldsymbol{w}) = \begin{bmatrix} \overline{\boldsymbol{x_0}} - \boldsymbol{x_0} \\ \boldsymbol{f}(\boldsymbol{x}_0, \boldsymbol{u}_0) - \boldsymbol{x_1} \\ \dots \\ \boldsymbol{f}(\boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}) - \boldsymbol{x}_N \end{bmatrix} = 0 \tag{13c}$$

As multiple shooting is not significantly more difficult to implement and in general provides better performance [10] it has been used for this MPC controller.

5.1.3 Optimization Solvers

Solving the NLP requires the use of an optimization problem solver. Choosing an appropriate solver depends on the application and formulation of the programming problem. MATLAB has several built-in solvers that work well for applications where the input parameters are easily computed. However, as the optimization problem is reformulated at each time step and would require the prediction to be calculated at each time this solver becomes extremely inefficient.

CasADi is a free open-source symbolic framework for optimal control that is compatible with MATLAB. CasADi allows the prediction horizon step to be predefined before the controller starts. It does this by using a symbolic representation of the control problem. When the controller starts the only conditions that are required are the initial states of the system, the initial control action, and a prediction (for multiple shooting). Due to these characteristics and being highly optimized, CasADi was chosen as the starting solver for the controller design.

5.2 Design Procedure and Scope Review

To ensure that the code written works and is the best iteration available, an implementation procedure that starts from a basic control objective and builds upon it was used (Appendix N). This method was chosen as it works within the agile framework outlined in section 3.2 and reduces the risk of having no deliverable by the end of project.

The initial scope of the project was to provide a working MPC that could be implemented on the vehicle. Due to workload, project structure and timelines it became clear that the GBDP team would not be able to provide a working solution for the car by the end of the project. It was determined from consultation with the TBRe-AI team that code will be provided up to step 8 (Appendix N) as the remaining steps are beyond the scope of the GBDP. This would show a proof of concept and allow the TBRe-AI Team to convert it for use on the car.

5.3 Controller Code Flowcharts and Testing Structure

The flowcharts help to outline the structure of the code before it is written. The testing flowchart was designed in such a way to clearly separate the elements that code that define the problem set-up and would only need to be defined before the controller is run and the code that would be continually looped and changed.

The MPC controller needs to operate in first lap and known track operating conditions (section 1.5). The formulation of these two problems from the perspective of the MPC controller is the same. However, for the purpose of testing, the full track layout will be known. The flowchart in Appendix I and Appendix J describe the process used to develop the MPC controller including the necessary code required for it to work in a simulated environment. Whereas the flowcharts in Appendix K, Appendix L, and Appendix M outline the functionality for the code deployed on the vehicle.

The controller needed for the acceleration event is essentially the same controller but with different tunings and prediction horizons and so no flowchart has been provided.

5.4 Controller Testing and Development

The code used to develop and test the controller can be found under Control-MPC at: https://github.com/lecheuklun/gbdp-tbreai

5.4.1 Development

During development it was found that some of the design concepts posed in section 4 were unsuitable for use.

The first issue was with the formulation of the magic formula tyres. Due to the number of variables involved, CasADi was unable to solve for the given model. To overcome this problem, the tire model has been approximated to a linear tire model. Although unsuitable for high performance application, it was found that the vehicle would only be operating in this region due to the constraints posed on the maximum speed on the car. Thus, a linear tire model was deemed suitable for the application.

The next issue was formulating the kinematic model using CasADi, as the equation (2) involves an approximation of the rate of change of the steering angle. This could normally be linearly approximated given the current and previous steering angle. However, a way to store previous values for future use within each propagation step of the MPC could not be found in CasADi. This affects the low-speed performance of the vehicle; in addition, the vehicle cannot be initialized from zero velocity. To overcome these two issues, for the purpose of simulation and testing, the speed of the vehicle is constrained to 1 m/s and the initial velocity of the vehicle is greater than 1 m/s. This was chosen as it was found that the car can operate at above 1m/s and effectively complete the track while above this speed the dynamic vehicle model behaves similarly to the expected, see Appendix E – Vehicle Model Simulation with Varying Inputs

To allow the controller to operate from a standing start a reformulation of the slip angle was tested. The longitudinal velocity variable in the slip angle formulations (5) and (7) is added with a small constant value. As the car is never allowed to go in reverse, $v_x \geq 0$, the denominator is always greater than zero and the equation is solvable. However, this does not improve the low-speed performance and does result in slight model inaccuracies across all speeds.

5.4.2 Stability Analysis

The input signals for the stability analysis were chosen to provide the most useful information to the design of the controller. As the system is MIMO (multiple input multiple output) rather than alter all the inputs for analysis only those inputs that would provide relevant information were chosen. For example, for the Step Input analysis of the controller rather than step all the input values from zero only the reference Y-Coordinate, and speed were stepped while the reference X-coordinate was increased in regular intervals. The advantage of doing it this way allows for an analysis of a more realistic situation that the controller would have to encounter. This method was also used for the impulse and ramp stability analysis for the same reasons (Appendix O).

Due to the formulation of the control problem and how the errors are calculated improving the rise time and overshoot subject to standard inputs does not necessarily imply better performance. What is more important is ensuring that the controller is stable under these types of inputs to ensure that it is stable under various scenarios in track-based events. Figure 5.2 demonstrates the stability of the controller subject to a step input as well as the control action calculated to be sent to the vehicle. It is important that the control actions are stable and do not fluctuate excessively as this is not realistic for actual control of the vehicle.

As seen from the results the controller produces a stable control action with little overshoot. The output signals for Frx and σ show a function with several step-like perturbations. This is a result of the formulation of the control problem as the reference position updates the control actions change producing this property. This may cause problems if the controller is not effectively tuned such that sufficient “damping” is applied to the control actions.

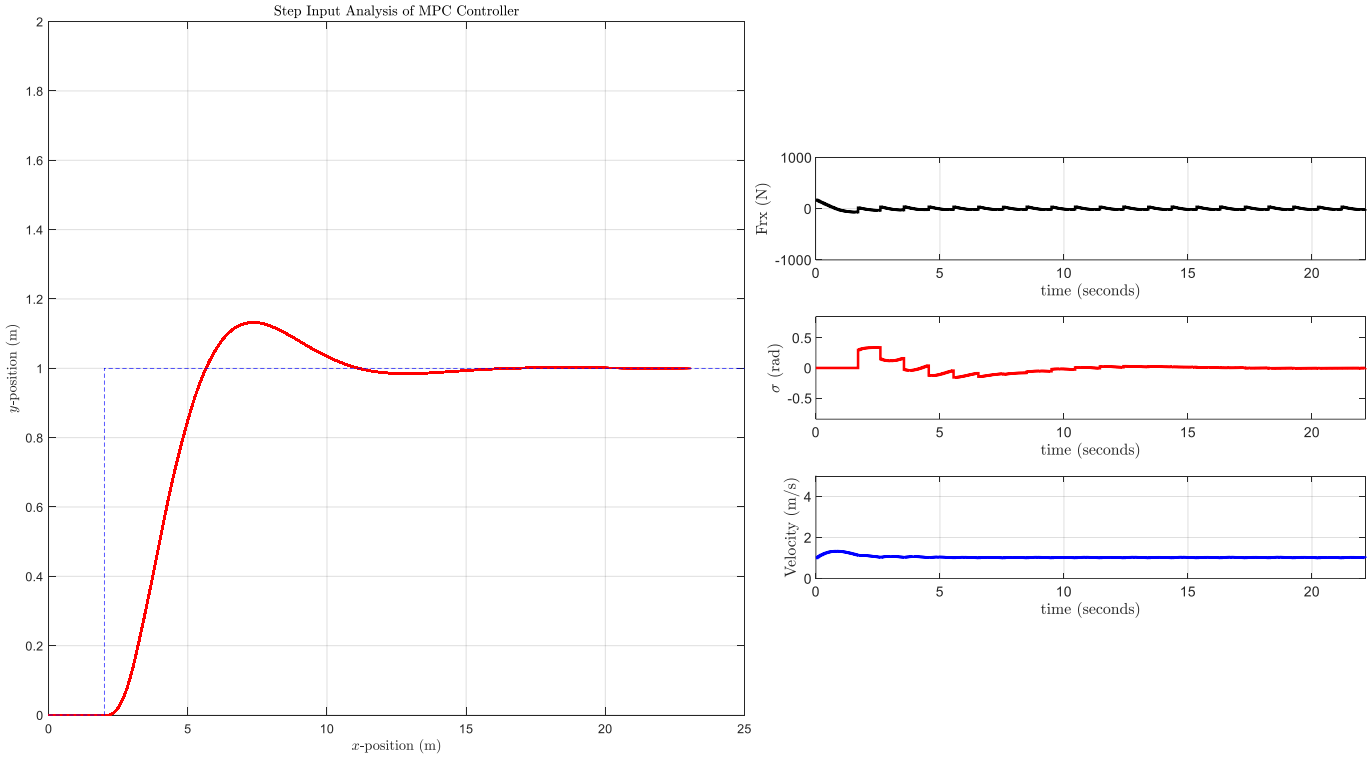


Figure 5.2 Step Input Stability Analysis MPC Controller

5.4.3 Track Events Simulation, Development and Testing

The underlying code structure was kept in line with the process outlined in the flowcharts as it was influenced by code that was proven to be functional [10]. However, significant iteration of certain elements was required before the simulation accurately reflected that of the car round the track. Namely, the determination of whether the cones and reference position need updating and the ideal number of cones to predict ahead of the vehicle.

Tuning of the weighting matrices the controller also posed challenges. Several simulations were carried out before the vehicle was able to complete the track while also producing stable and control output that did not fluctuate excessively. This was done in a systematic way by considering the relative sizes of the states in the model and the control inputs.

The results of a one lap simulation of a known track are displayed in Figure 5.3 with the accompanying control actions and velocity of the vehicle. These show the controller can navigate the car around a track effectively. The simulation was developed so that multiple laps could be tested, and number of cones predicted forward altered to ensure that optimal control is achieved.

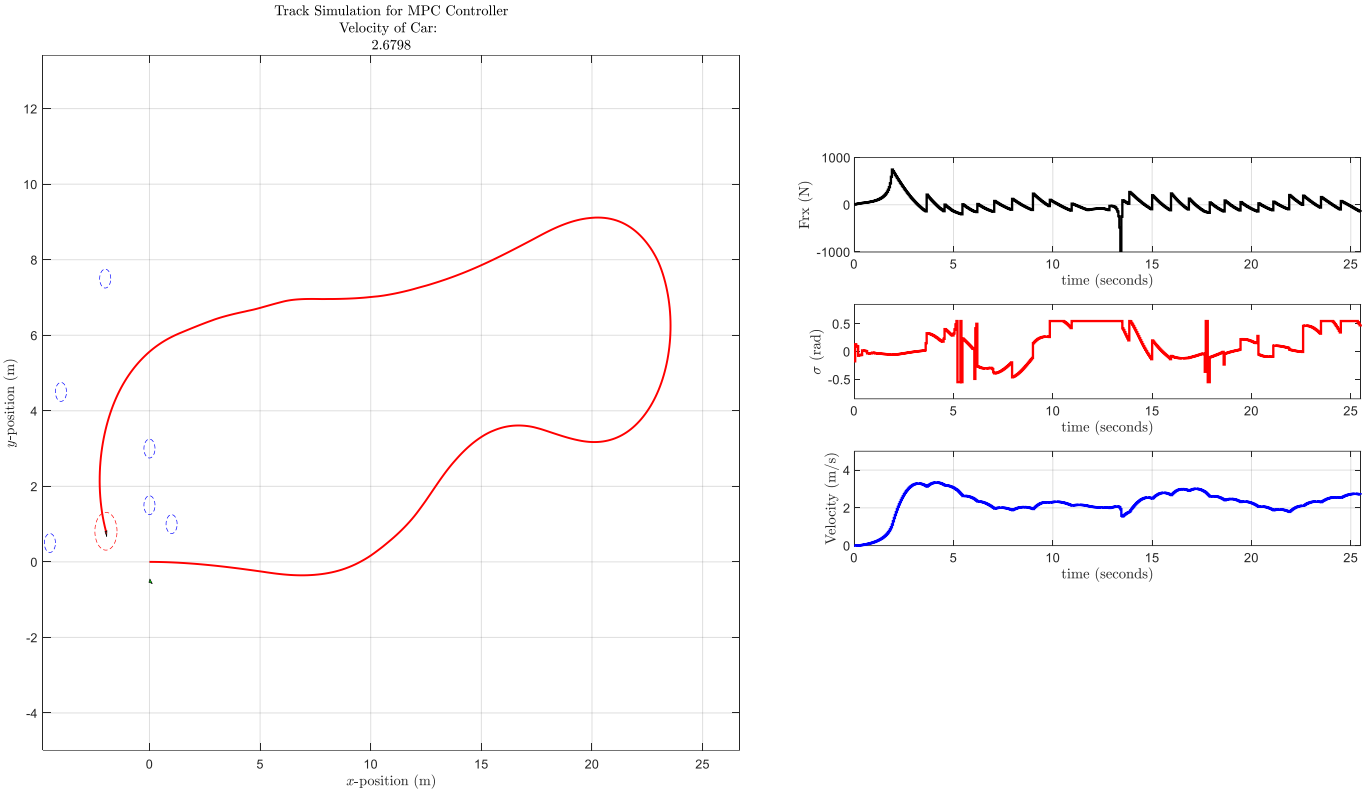


Figure 5.3 Track Event Simulation MPC Controller

6 Implementation Proposal

6.1 Usability on Car

The primary issues with the current design are associated with the low speed and initialization of the controller as identified in section 5.4.1. The code implemented on the final car will need to account for these issues either by changing the model used for the car or by having a work around solution that by passes the MPC control at low speeds up to 1 m/s.

Additionally, error handling (SSR-14,15,16,17) will need to be added before the code should be run on the vehicle. The code should catch errors and act accordingly depending on the error type. The table in Appendix P provides a summary of some of these errors and actionable recommendations to overcome them.

6.2 Upgrades

Upgrades to the controller may include better formulation of the track boundaries. Currently the distance from the current position to the reference position is limited by the car going between cones to reach the reference. By approximating the cones to functions it would stop this issue and allow the distance to `be increased. The benefit of this is that the controller can calculate the optimal trajectory for more of the track as the reference posture is not necessarily the optimal posture.

6.3 Performance and Reliability

Although the tuning of the controller within the MATLAB simulation will be able to provide optimal control of the simulated car; the tuning will not carry over to the real vehicle or the realistic simulation. In addition, once the full physical attributes of the car are known, such as tyre coefficients, the controller will need to be re-tuned. Once fully re-tuned only then can the performance of the controller be accurately evaluated.

Assessing the reliability of the controller is difficult within such a controlled simulation. During testing it was clear that without adequate tuning the controller fails to perform the required tasks. To improve reliability the controller should be rigorously tuned by simulating multiple scenarios in the detailed simulation and then on the car. With the addition of error handling the controller will be more reliable; and with the assessment of where errors occurred, the performance improved.

6.4 Time Cost

The main cost consideration with this subsystem is associated with the development time. This factor has been considered throughout the design and was a key criteria during the evaluation of concepts and choice of programming language. An estimate of the days spent developing the system have been provided, Table 6.1, along with estimates of development times for the outstanding work.

Table 6.1 Summary of Time Cost Developing MPC Controller

Work	Time Cost (Days)
Preliminary Research	10
Scoping of Controller including interfaces	5
Concept Generation, Development and Testing	14
Learn CasADi and code implementation of MPC	7
Develop model equations and plan code flowcharts	7
Write code and test for standard inputs	7
Write code and test for simulated track	14
Total	**64**
Convert to C	*3*
Add error handling	*10*
Integrate with ROS	*5*
Tune in Simulator	*3*
Tune on Car	*3*
Sum Additional Extra	***24***

7 Project Review and Conclusions

7.1 Verification

Based on testing the controller appears to fulfil many of the high priority requirements and verifies them based on methods outlined in the SSR. As some of the requirements are for the full controller on the vehicle it is not possible to verify all the requirements at this stage. However, the controller has been designed so that the requirements that are yet to be verified can still be implemented and verified at a later stage.

7.2 Team

The structure of this year's GBDP was different to previous years and as a result there was significantly less group planning required before work was conducted. In addition, there were fewer people working on the mech GBDP, for example there were only two MEng students for the whole car control system. As a result, each member had to organize their subsystems and deadlines individually as there was no project manager. In addition, the requirements of the TBRe-AI kept changing leading to difficulties for some members in figuring out the exact requirements of their sub-systems

Once the interfaces were defined, the work done on the master controller was found to be quite independent from other activities. This meant that there could be a greater focus on creating a good technical piece of work for the TBRe-AI team, which was one of the primary design objectives.

7.3 Concluding Remarks and Future Work

Due to the time constraints of the GBDP, considerable work needs to be done before the controller can be implemented on the car. In addition, as much of the car is still in development, along with detailed simulation packages, it was not possible to achieve much more than what has been achieved by the end of the GBDP. The main value of the work that this report poses, is evidence of a control solution that has the capability of working and meeting the requirements on the physical car. In addition, several new skills have been developed over the course of the project that will prove useful for future projects.

As highlighted in the implementation procedure the next steps to be carried out are the conversion of the code to C and subsequent implementation into ROS for the vehicle. Although, the underlying code and design principles used in MATLAB should carry forward onto the implementation on the car; it is likely that there will need to be adjustments to how the code integrates with the AI pipeline. Once these adjustments are made the car can be tuned first within the realistic simulation environment developed by the team and then finally on the car.

8 REFERENCES

[1] IMechE, “Formula Student,” [Online]. Available: https://www.imeche.org/events/formula-student. [Accessed 12 March 2022].

[2] MathWorks, “Lane Following Using Nonlinear Model Predictive Control,” MathWorks, [Online]. Available: https://uk.mathworks.com/help/mpc/ug/lane-following-using-nonlinear-model-predictive-control.html. [Accessed 12 March 2022].

[3] J. Kazban, “AMZ Driverless: The Full Autonomous Racing System,” arXiv, 2019.

[4] CasADi, “CasADi,” [Online]. Available: https://web.casadi.org/. [Accessed 13 03 2022].

[5] Y. Ding, “Simple Understanding of Kinematic Bicycle Model,” 15 February 2020. [Online]. Available: https://dingyan89.medium.com/simple-understanding-of-kinematic-bicycle-model-81cac6420357. [Accessed 13 March 2022].

[6] J. Kong, M. Pfeiffer, G. Schildbach and F. Borrelli, “Kinematic and Dynamic Vehicle Models for Autonomous Driving,” IEEE Intelligent Vehicles Symposium, 2015.

[7] L. N. a. H. P. E. Bakker, ““Tyre modelling for use in vehicle dynamics studies,”,” SAE, 1987.

[8] Formula Student, *Formula Student - Artifical Intelligence (FS-AI) 2021 Rules,* Formula Student, 2021.

[9] M. Land, “Steering with the head: The visual strategy of a racing driver,” Current Biology, 2001.

[10] M. Mehrez, “MPC and MHE implementation in Matlab using Casadi | Part 1,” 29 January 2019. [Online]. Available: https://www.youtube.com/watch?v=RrnkPrcpyEA&t=2027s&ab_channel=MohamedW.Mehrez. [Accessed 22 April 2022].

[11] A. Liniger, A. Domahidi and M. Morari, “Optimization-based autonomous racing of 1:43 scale RC cars,” 2014.

[12] K-State Polytechnic, “The Unicycle Model,” [Online]. Available: http://faculty.salina.k-state.edu/tim/robotics_sg/Control/kinematics/unicycle.html. [Accessed 2022 03 13].

9 APPENDIX

9.1 Appendix A – Control System Requirements

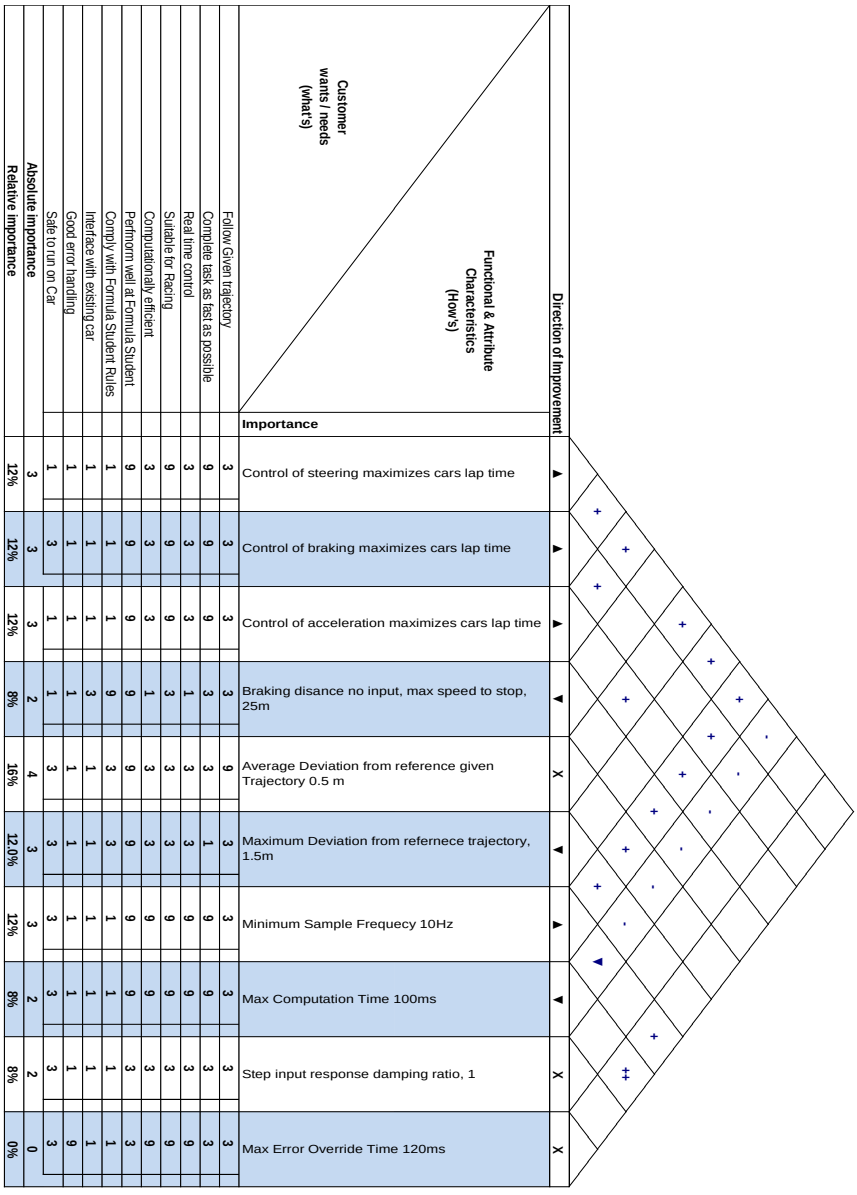
Requirement Code	Requirement	Priority	Verification
Functional			
SR-1	Car braking shall be controlled by system	H	Demonstration
SR-2	Car steering shall be controlled by system	H	Demonstration
SR-3	Car acceleration shall be controlled by system	H	Demonstration
SR-4	Control system shall run in real time	H	Test
Non-Functional/ Performance			
SR-5	Control response time shall be suitable for racing speeds	M	Test
SR-6	Control System should be computationally efficient	M	Test
SR-7	Should perform well at Formula Student	M	Test
SR-8	Written code should use make use of good coding practices	L	Examination
Constraints			
SR-9	Shall comply with Formula Student Rules	H	Examination
SR-10	Should stay within computer memory limits	M	Test
SR-11	Low level controllers shall run on MBED LC1768	M	Examination
SR-12	Master Controller should be an MPC type controller	M	Examination
Interface			
SR-13	The control system shall interface with existing communicating protocols	H	Demonstration
SR-14	The control system shall interface with mechanical systems	H	Demonstration
SR-15	The control system shall interface with car electrical systems	H	Demonstration
SR-16	The control system shall interface with computer operating system	H	Demonstration
SR-17	The control system shall interface with testing simulation software	H	Demonstration
Safety			
SR-18	Code should have good error handling	M	Test
SR-19	Shall be safe to run on physical car	H	Test
SR-20	Should have diagnostic tools for testing	L	Examination

9.2 Appendix B – MPC Controller Requirements

Requirement Code	SR-Requirement Link	Requirement	Verification Method	Priority
Functional				
SSR - 1	[1, 2, 3, 7]	Control of acceleration shall optimize car's ability to traverse track to best possible time around track, TBD.	Test different methods in simulation to determine which achieves the best time around given paths with a speed limit of 20km/h	M
SSR - 2	[1, 2, 3, 7]	Control of braking shall optimize car's ability to traverse track up a on track events to best possible time around track, TBD.	Test different methods and tunings in simulation to determine which achieves the best time around given paths	M
SSR - 3	[1, 2, 3, 7]	Control of steering shall optimize car's ability to traverse track up on track events to best possible time around track, TBD.	Test different methods in simulation to determine which achieves the best time around given paths with a speed limit of 20km/h	M
SSR - 4	[3, 7]	Maximum speed shall be limited to 20km/h on track events.	Ensure that a speed limiter has been implemented for the track-based events.	M
SSR - 5	[3, 7]	Maximum speed shall be unlimited on acceleration events.	Analyze torque request data during simulations and testing to ensure max torque is being applied	M
SSR - 6	[2, 7]	Car to follow given trajectory with maximum error of 1.5m either side.	Test control system in simulation environment with given trajectory to ensure path is followed.	H
SSR - 7	[7, 9]	Control system shall ensure car does not hit cones on track which lie at a minimum of 3m apart, D8.1.1. [1]	Test control system in simulation environment around a track like formula student event to ensure no cones are hit is followed.	H
SSR - 8	[7, 9, 18x`, 19]	Control system shall allow car to come to controlled stop within 25m when no trajectory input is received, D4.3.6. [1]	Create test environment where trajectory signal is cut off and measure distance for car to come to a stop.	H
SSR - 9	[4, 5, 6, 7]	Control system should sample at a minimum of 10Hz to allow for updates every 0.5m at maximum speed.[2]	Set the sample time of the controller above 10Hz	M
SSR - 10	[1, 2, 3, 19]	Control system shall be stable under common inputs (impulse, step, ramp inputs).	Input impulse, step, ramp signals to the controller and measure the output of the controller; ensuring that the controller is stable for all inputs.	H
Non-Functional/ Performance				
SSR - 11	[5, 7]	Control response time should be under 120ms for 90% of computations [2]	Record the computation time for each control move to ensure 90% are below 50ms.	M

SSR - 12	[8]	Code should be written with good coding practices namely commenting, naming conventions, indentation, and simplicity.	Review code before completion.	L
SSR - 13	[7, 9]	Control system shall be usable on all Formula Student autonomous missions, DV2.6.1 [1]	Test control system on simulation environment of each Formula Student event and ensure compliance for each.	H
SSR - 14	[18, 19]	Erroneous output values should be identified.	Simulate erroneous output values to check whether system correctly detects them	M
SSR - 15	[5, 6, 18, 19]	Control system should override to previous output if system takes over 150ms to respond.	Simulate situation that causes system to fail to compute solution within given time and assess whether output is previous output.	M
SSR - 16	[18]	All corrected errors should be logged under error type during testing.	Simulate errors into system during simulation testing and check if system identifies them.	L
SSR - 17	[18]	All logged errors should store information about the location of error during program flow.	Simulate errors into system during simulation testing and check if system correctly identifies location at which they occurred.	L
Constraints				
SSR - 18	[16]	Control system shall be implemented as digital controller.	Control system implemented on cars onboard computer.	H
SSR - 19	[12]	Controller shall be implemented as an MPC style controller.	Controller implemented compared against existing MPC controllers	H
Interface				
SSR - 20	[13, 16, 17]	System shall interface with ROS to retrieve and send data	Run simulation with control system in ROS and verify output is as expected.	H
SSR - 21	[9, 13]	System shall communicate with low-level controllers using CAN Buses, DV1.1.4 [1]	Verify signals sent over CAN bus are correct by analysing signal sent and signal received.	H
SSR - 22	[3]	Control system shall send a positive torque request to the motor max value, TBD.	Analyse signals sent to motor in simulation tests and ensure they are positive and within max value.	H
SSR - 23	[1]	Control system shall send a negative force request to the braking system max value, TBD.	Analyse signals sent to braking system in simulation and ensure they are negative and within max value.	H
SSR - 24	[2]	Control system shall send a steering angle request to the steering system max steering angle, TBD.	Analyse signals sent to braking system in simulation and ensure they are negative and within max value.	H
Safety				
SSR - 25	[1, 9, 19]	Shall be able to trigger EBS signal, DV 1.4 [1]	Simulate and experiment on physical car that an emergency brake is initialized during operation from within the control program.	H

9.3 Appendix C – House of Quality Table MPC Controller



‡	Strong positive correlation
+	Positive correlation
'	Negative correlation
◄	Strong negative correlation
▼	Objective to minimise
▲	Objective to maximise
X	Objective to hit target
9	Strong relationship
3	Moderate relationship
1	Weak relationship

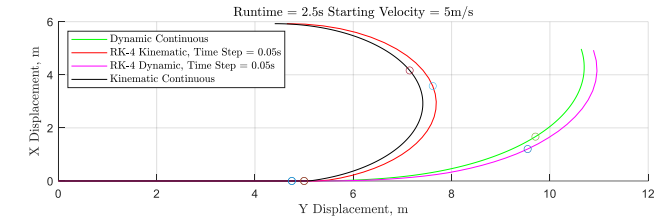
9.4 Appendix D – Pugh Matrix Vehicle Model

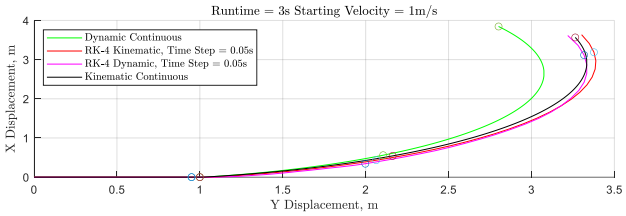
		Concepts		
Criteria	**Rating (1-10)**	**AMZ Combined Kinematic Bicycle Model**	**Kinematic Bicycle Model**	**Dynamic Bicycle Model**
Deviation from Reference Trajectory	**8**		-	s
Control of Steering, maximise lap time	**9**		-	-
Control of Acceleration, maximise lap time	**9**		-	-
Control of Braking, maximise lap time	**9**		-	-
Computation Time	**8**		+	+
Implementation Time	**3**		+	+
Usability in Formula Student Events	**7**		s	s
Interoperability with AI pipeline	**5**		s	s
Stability	**6**		+	s

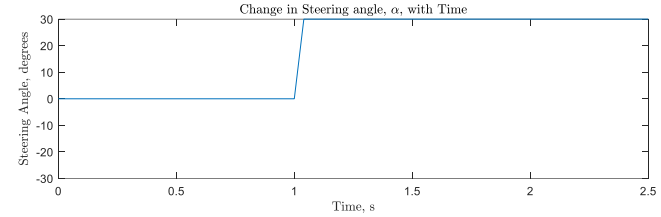
Sum of Positives	**0**	**4**	**3**	**2**
Sum of Negatives	**0**	**-5**	**-4**	**-3**
Sum of Same	**9**	**0**	**2**	**4**
Total	**9**	**-1**	**1**	**3**
Weighted Sum of Positives		**22**	**17**	**12**
Weighted Sum of Negatives		**-42**	**-35**	**-27**
Total	**0**	**-20**	**-18**	**-16**

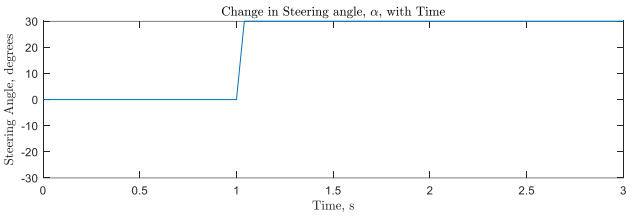
s	**0**	same
+	**1**	better
-	**-1**	worse

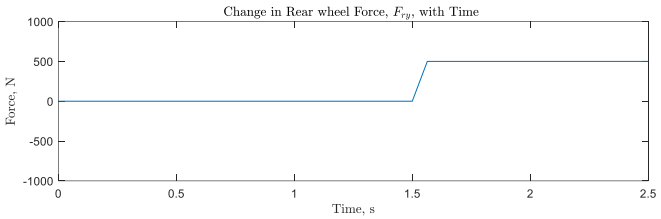
9.5 Appendix E – Vehicle Model Simulation with Varying Inputs

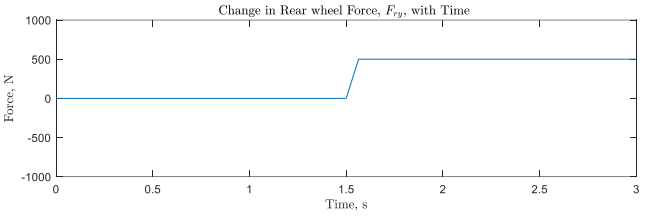


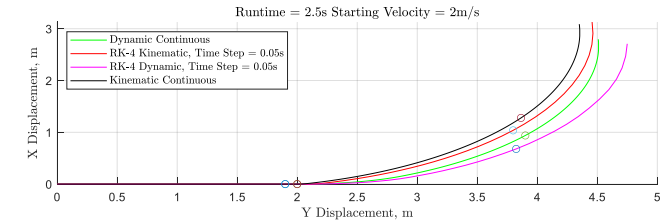


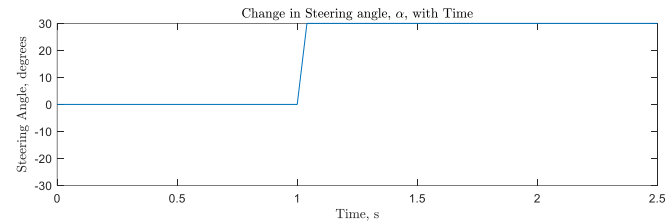


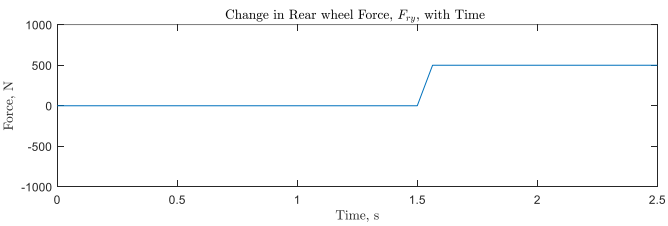




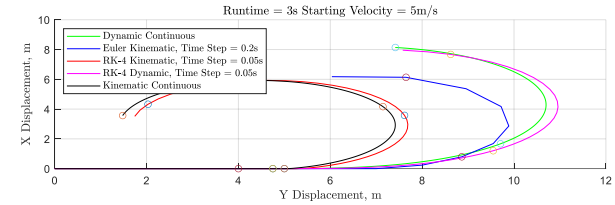


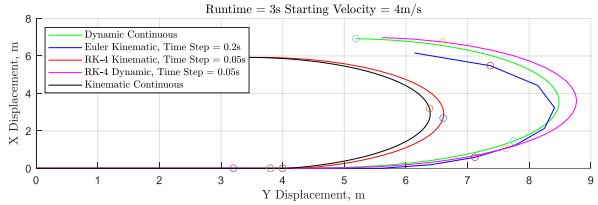


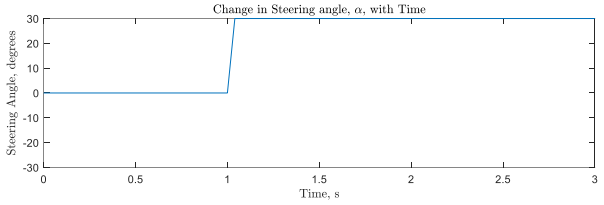


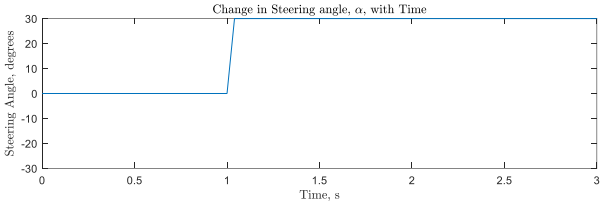


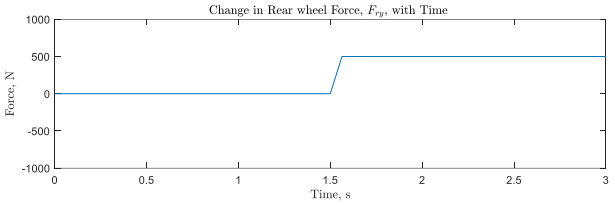
9.6 Appendix F – Vehicle Model Simulation with Varying Discretization

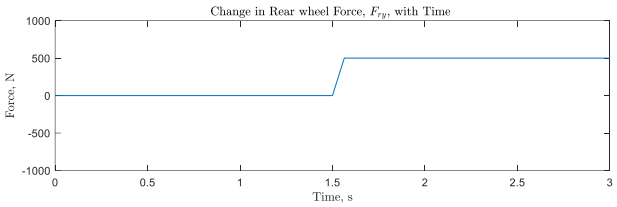


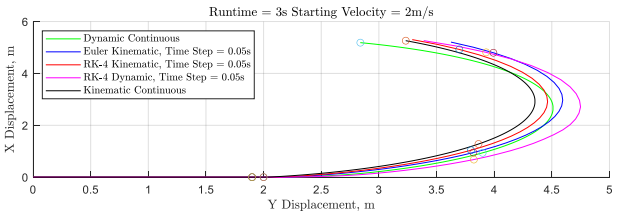


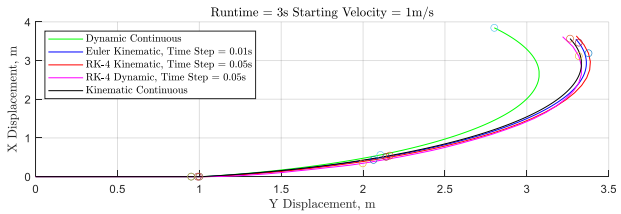


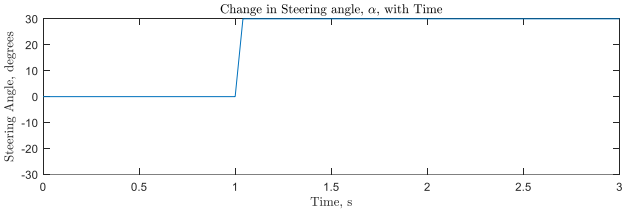


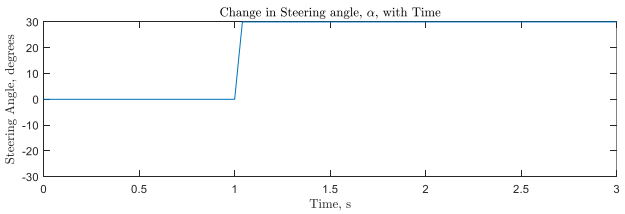


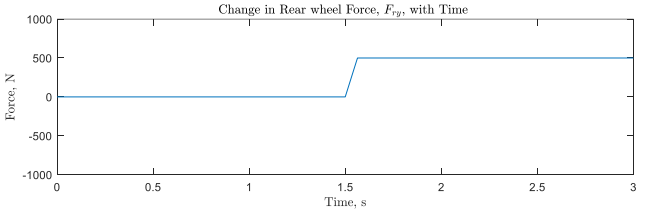


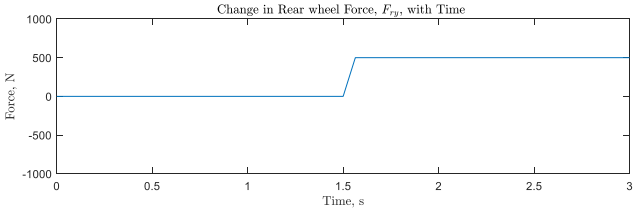












9.7 Appendix G – Pugh Matrix Revaluation of Vehicle Models

		Concepts			
Criteria	**Rating (1-10)**	**AMZ Combined Kinematic Bicycle Model**	**Kinematic Bicycle Model with Adjustable Discretization**	**Combined Kinematic and Dynamic Bicycle model**	**Dynamic Bicycle Model**
Deviation from Reference Trajectory	**8**		-	s	s
Control of Steering, maximise lap time	**9**		-	s	-
Control of Acceleration, maximise lap time	**9**		-	s	-
Control of Braking, maximise lap time	**9**		-	s	-
Computation Time	**8**		+	s	+
Implementation Time	**6**		+	+	+
Usability in Formula Student Events	**7**		s	s	s
Interoperability with AI pipeline	**5**		+	+	s
Stability	**6**		-	s	s

Sum of Positives	**0**	**3**	**2**	**2**
Sum of Negatives	**0**	**5**	**0**	**-3**
Sum of Sames	**9**	**1**	**7**	**4**
Total	**9**	**9**	**9**	**3**
Weighted Sum of Positives		**19**	**19**	**14**
Weighted Sum of Negatives		**-41**	**0**	**-27**
Total	**0**	**-22**	**19**	**-13**

s	**0**	same
+	**1**	better
-	**-1**	worse

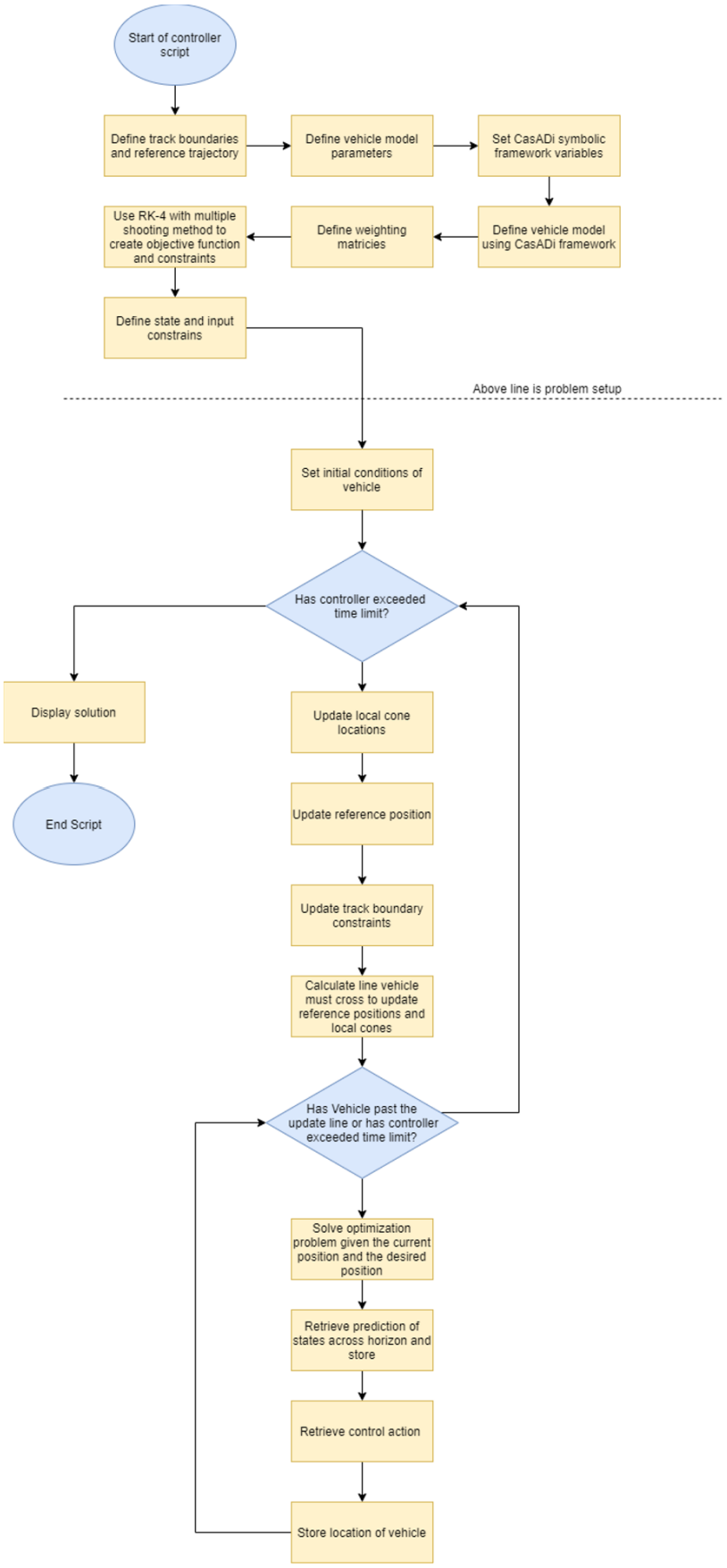
9.8 Appendix H – Pugh Matrix Track Error Calculation

		Concepts				
Criteria	**Rating (1-10)**	**AMZ Contour Formulation**	**Concept 1**	**Concept 2**	**Concept 3**	**Concept 4**
Deviation from Reference Trajectory	**8**		-	s	-	-
Control of Steering, maximise lap time	**9**		s	s	-	s
Control of Acceleration, maximise lap time	**9**		-	-	s	-
Control of Braking, maximise lap time	**9**		-	-	s	-
Computation Time	**8**		+	+	-	+
Implementation Time	**6**		+	+	s	+
Usability in Formula Student Events	**7**		-	s	-	-
Interoperability with AI pipeline	**5**		+	+	+	+
Stability	**6**		s	+	s	s

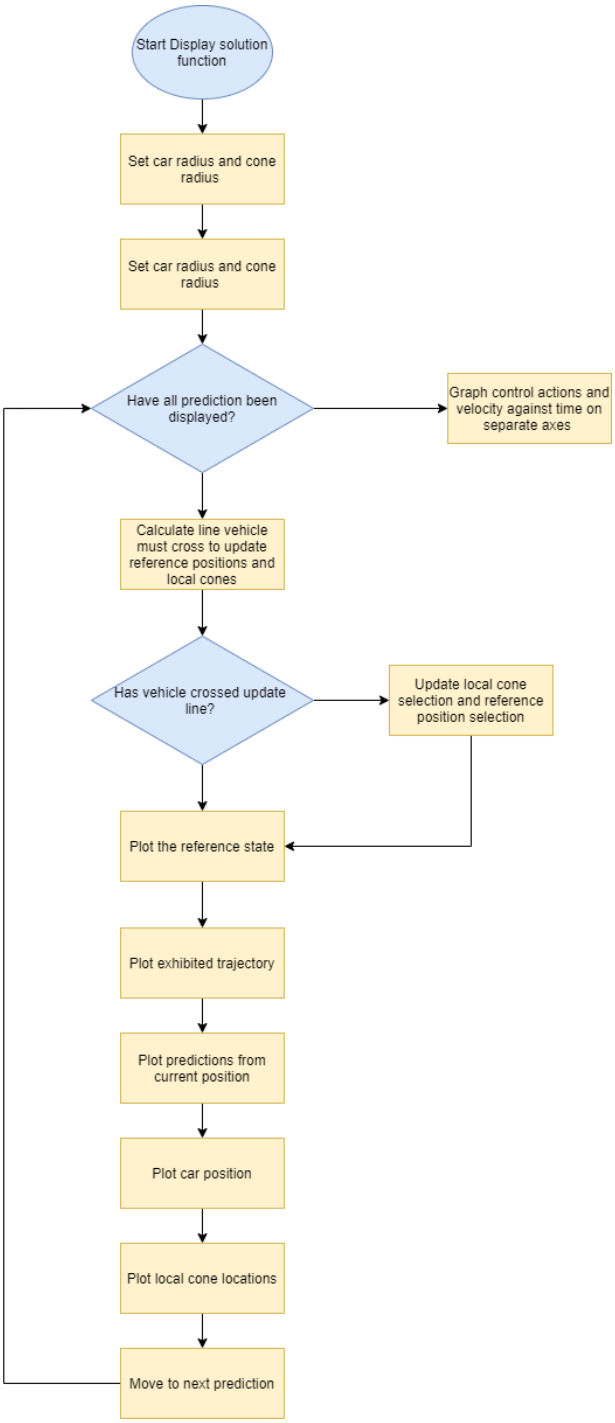
Sum of Positives	**0**	**3**	**4**	**1**	**3**
Sum of Negatives	**0**	**-4**	**-2**	**-4**	**-4**
Sum of Sames	**9**	**2**	**3**	**4**	**2**
Total	**9**	**1**	**5**	**1**	**1**
Weighted Sum of Positives		**19**	**25**	**5**	**19**
Weighted Sum of Negatives		**-34**	**-18**	**-32**	**-33**
Total	**0**	**-15**	**7**	**-27**	**-14**

s	**0**	same
+	**1**	better
-	**-1**	worse

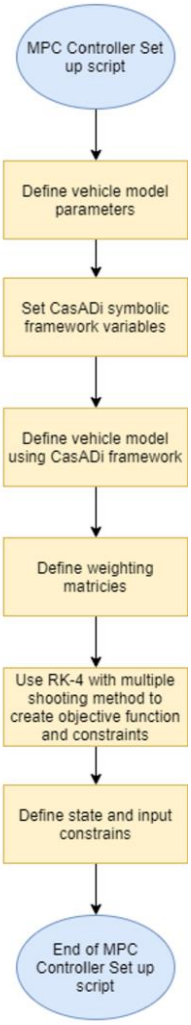
9.9 Appendix I – Flowchart MPC Controller in Test Enviornment



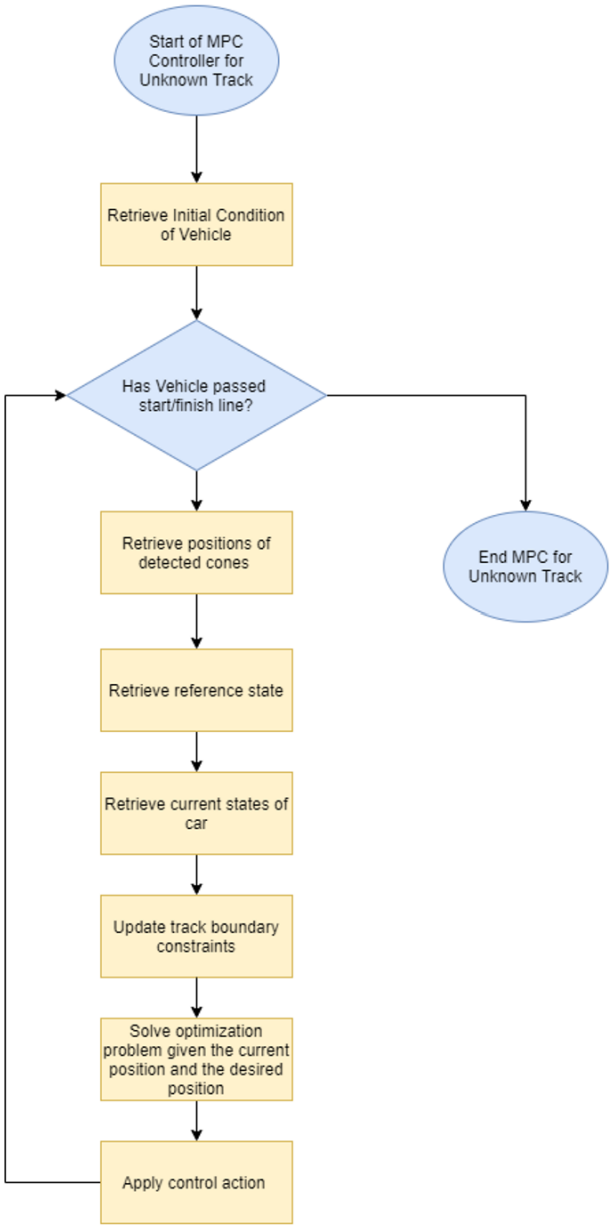
9.10 Appendix J – Flowchart MPC Controller Display Simulation



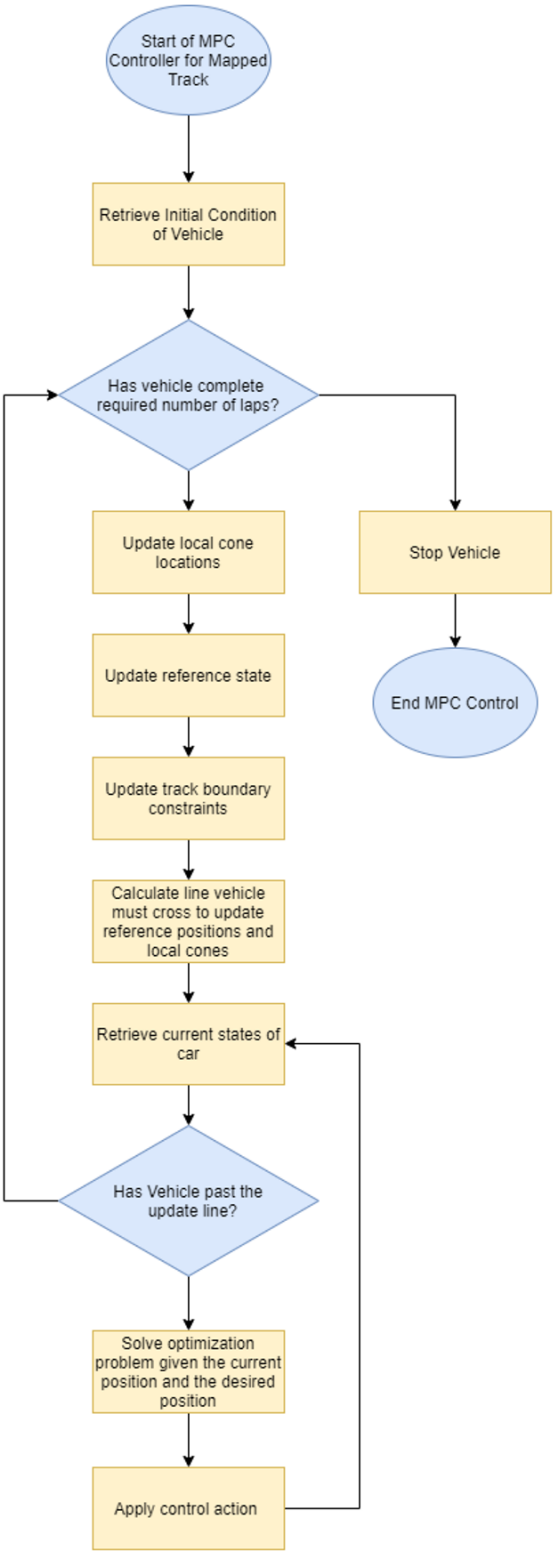
9.11 Appendix K – Flowchart MPC Controller Set-Up



9.12 Appendix L – Flowchart MPC Controller Unknown Track



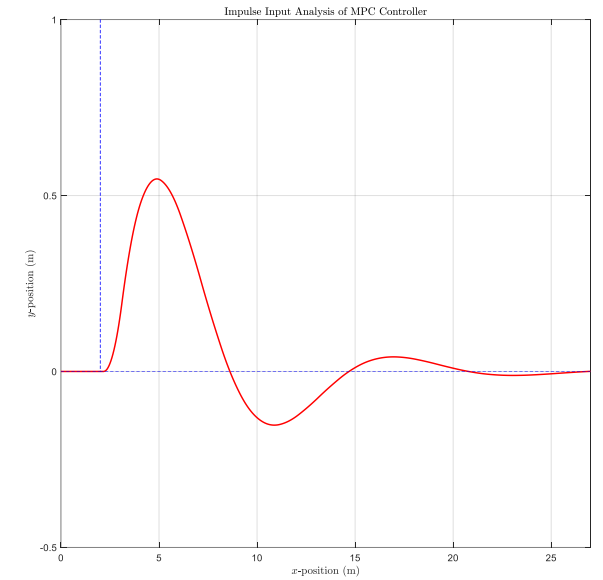
9.13 Appendix M – Flowchart MPC Controller Known Track

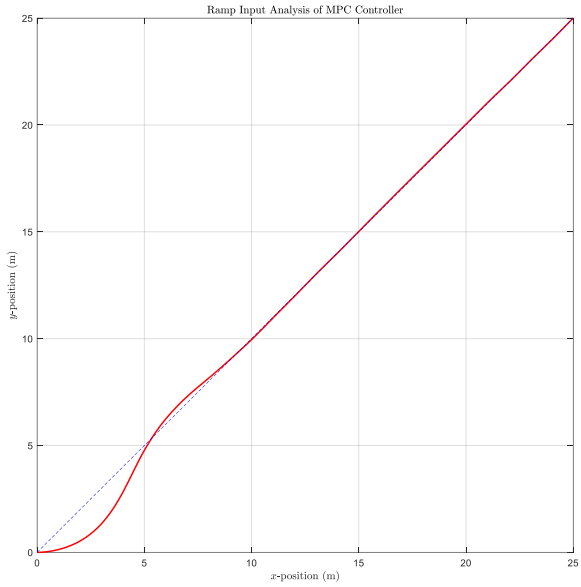


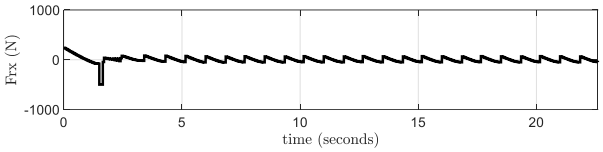
9.14 Appendix N – Implementation Procedure

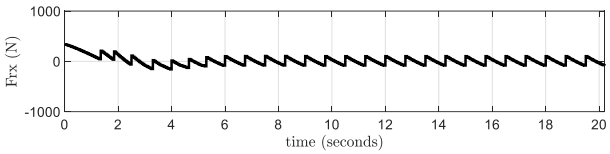
Step	Procedure
1	Vehicle can control to stationary reference state with no track boundaries
2	Vehicle can track a moving reference state with no track boundaries
3	Vehicle can avoid cones and track moving reference
4	Vehicle is stable under impulse, step and ramp inputs
5	Update cones and reference state after vehicle passes given point
6	Vehicle can complete full lap of a given track
7	Vehicle can complete multiple laps of given track
8	Reference states can be created given only reference coordinates
9	Split code into functions and scripts specific for use on car
10	Generate C code for ROS
11	Integrate C code into vehicle

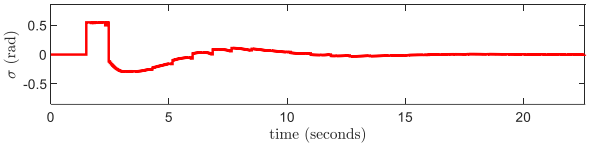
9.15 Appendix O – Stability Analysis of MPC controller

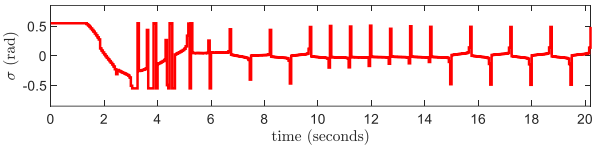


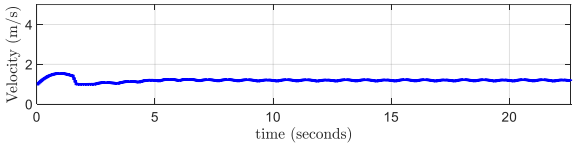


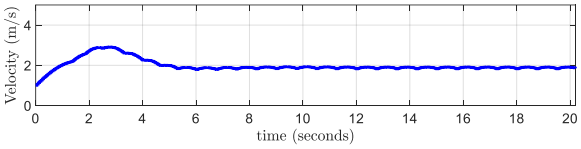












9.16 Appendix P - Error Handling

Failure Mode	Failure Effects	Severity	Failure Causes	Reccomendation Action
Unstable Solution	Car follows incorrect path, goes off track, damages low level controllers	3	Discretization method and model not fully tested before implementation	Detection system to identify and remove instabilities
Too slow to evaluate	Car fails to respond to track at high speeds resulting, goes off track	2	Vehicle model too complicated to solve or discretization time step too small for real-time control	Revert to previous control action if time limit exceeded
Fails to find solution	No control action produced, goes off track	3	Inputs to optimizer not suitable due to lack of testing and modifying input and tuning parameters	Revert to previous control action if time limit exceed
Vehicle fails to avoid cones	Performs poorly in competition, possible damage to vehicle,	2	Track constraint formulation not accurate enough	Reduce Target speed of reference positions
Vehicle fails to follow track	Car unable to complete track and fails competition event	3	Path error formulation not accurate enough to allow vehicle to follow path	Re-tune controller
Controller produces unrealistic control actions	Control actions cannot be used, and vehicle cannot be controlled, car goes off track and may damage low level controllers.	3	Tuning parameters not tuned properly	Re-tune controller
Code fails during operation	Car fails to complete track	3	Error occurs during operation, runtime exceeded, or crash occurs	Revert to previous control action and re-run script