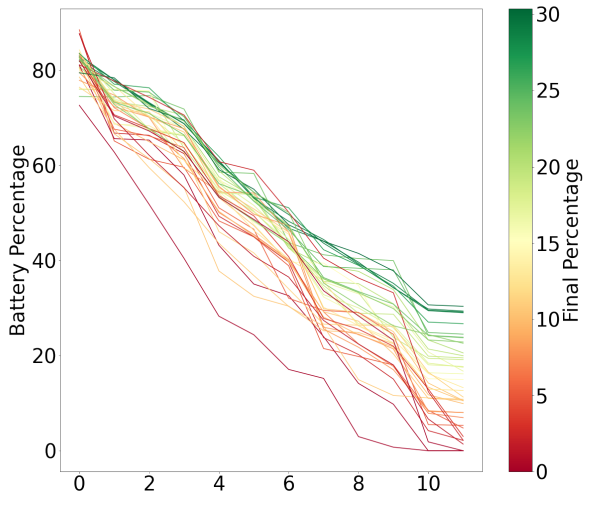
Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Markus Anthony Dørheim Ho-Yen

Driving Strategy Optimisation in DNV GL Fuel Fighter

Master's thesis in Cybernetics and Robotics
Supervisor: Sverre Hendseth
February 2021



Markus Anthony Dørheim Ho-Yen

Driving Strategy Optimisation in DNV GL Fuel Fighter

Master’s thesis in Cybernetics and Robotics
Supervisor: Sverre Hendseth
February 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk
Institutt for teknisk kybernetikk

MASTEROPPGAVE

Kandidatens navn:	Markus Anthony Dørheim Ho-Yen
Fag:	TTK4900 Teknisk kybernetikk, masteroppgave
Oppgavens tittel (norsk):	Kjørestrategioptimalisering i DNV GL Fuel Fighter
Oppgavens tittel (engelsk):	Driving Strategy Optimisation in DNV GL Fuel Fighter

Oppgavens tekst:

Kjørestrategi er en essensiell del av å kjøre energieffektivt. Det er blitt gjort analyser av hvordan man kan gjøre mest mulig energi- og tidseffektivt i en vanlig bil og i en racerbil på bane. Hvordan endrer strategien seg når man skal kjøre en bil bygd for energieffektivitet på bane, men ikke trenger å komme seg raskest mulig fram? Gjennom denne masteroppgaven skal student gjøre rede for ulike kjørestrategier og optimaliseringer, og hvordan disse kan brukes av sjåføren i DNV GL Fuel Fighter for å konkurrere i Shell Eco-Marathon.

Studenten skal:
- Gjøre rede for ulike kjørestrategier og optimaliseringsalgoritmer.
- Lage en matematisk modell av bilen.
- Implementere en eller flere optimaliseringsalgoritmer å finne optimal kjørestrategi, med fleksbilitet i både baneoppsett og bilegenskaper.

Oppgaven gitt:	12. oktober 2020
Besvarelsen leveres innen:	8. mars 2021

Utført ved Institutt for teknisk kybernetikk

Veileder: Sverre Hendeth

Trondheim, 12. oktober 2020

Abstract

This thesis explores different methods to find a driving strategy for DNV GL Fuel Fighter in Shell Eco-Marathon (SEM) competition. The thesis goes some of the basics of path planning, driving strategy and mathematical modeling of vehicle dynamics and its electrical system. It also details some optimisation methods for path planning and reinforcement learning. Reinforcement learning through Q-learning, Rapidly-exploring Random Tree (RRT), and RRT are also implemented. Human reinforcement learning is also implemented through crowd sourcing using a simulator competition. All these methods have shown to be useful to gain some insight in driving strategy and further work is needed to finalise the methods for the SEM competition.

Sammendrag

Denne oppgaven utforsker forskjellige metoder for å finne en kjørestrategi for DNV GL Fuel Fighter i Shell Eco-Marathon (SEM) konkurransen. Oppgaven går gjennom stiplanlegging, kjørestrategi og matematisk modellering av kjøretøy dynamikken og det elektriske systemet. Den går også gjennom noen optimaliseringsmetoder for stiplanlegging og forsterkningslæring. Forsterkningslæring ved hjelp av Q-Læring, og Hurtig-utforskende tilfeldige trær(RRT) og RRT er implementert. Menneskelig forsterkningslæring via nettdugnad er også brukt, ved å holde en konkurranse i en simulator. Alle disse metodene virker til å være nyttig for å få innsikt i kjørestrategi. Mer arbeid gjenstår for å kunne ferdigstille metodene for bruk i SEM konkurransen.

Contents

1 Introduction

Energy efficiency is becoming a more central topic in the autoindustry. As more and more vehicles become electric, the efficiency of the vehicle becomes increasingly more important. Even though Electric Vehicles (EVs), with an efficiency of 94 %, are far more energy efficient than Internal Combustion Engine (ICE) vehicles, with their efficiency of 20 %(U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy 2017), the "refuel" time of the EVs is greater. This means that the benefit of greater energy efficiency is still significant. Travelling a greater distances on each charge means fewer charging stops. The energy density of batteries is also much lower than the energy density of gasoline, thus making the most out of the energy is important.

The fuel consumption that can be reduced by driving more efficently is, on average, up to 10%(Barkenbus 2009). In the last competition, the DNV GL Fuel Fighter car had an energy efficiency of 183 km/kWh, meaning a 10% increase in efficiency could lead to an efficiency of over 200 km/kWh!

This report will explore possible methods of calculating a driving strategy and aims to give a foundation to path planning and driving strategy optimisation for future DNV GL Fuel Fighter members. This means that some background material that is not relevant for the implementation is still kept for reference for future work.

1.1 Problem Formulation

This chapter explains the problem that the driving strategy will solve. The goal of the SEM competition is to drive the most energy efficiently. The vehicle has to complete a predefined number of laps on a racetrack within a certain time. As long as the car completes the number of tracks within the given time, the time does not affect the score, only the energy efficiency. At the end of every lap, the car has to come to a complete stop, before continuing the next lap. This rule is in place to simulate urban city driving, with typical stop-and-go traffic. The maximum speed allowed in the competition is 40 km/h. As there is a human driver in the vehicle controlling it, low-level control and obstacle avoidance is not necessary to incorporate in the driving strategy. There are other competing vehicles driving at the same time, so the strategy will only output an ideal path that the human driver should follow. The driver is in control of the vehicle at all times, and will have to take corrective actions for dynamic changes to the environment.

1.2 Contributions

While working on this thesis, I was also leading the Software group in DNV GL Fuel Fighter. The Software group's mission is to improve on-track performance through off-track software tools. The group consists of the Driving

Strategy members: Benedicte Chen Vestrum and Mithila Packiyanathan, and the Driving Simulator members: Finn Ferdinand Schjøll Sandvand and Mohamed Barry. They have helped greatly with discussions in general, and contributions to the simulator competition mentioned in this report.

2 Background

This background chapter will explain the theory to follow the implementation in described in section 3. The DNV GL Fuel Fighter's system will first be presented to understand what needs to be modeled. *Driving strategy* will introduce trajectory planning, racing lines, and some mathematical calculation of racing lines. *Mathematical Modeling* presents the vehicle dynamics and how to model the electrical system's energy usage. *Planning algorithms* explain some strategies common from autonomous systems, which can be used for DNV GL Fuel Fighter's use case. *Reinforcement learning* is then explained through Markov Decision Process (MDP) and Q-learning. *Related work* summarises what other publicly available research have achieved relating to driving strategies for the SEM competition.

2.1 DNV GL Fuel Fighter's System

DNV GL Fuel Fighter is currently using the same car that competed in 2019. The telemetry recorded during the race attempts last year is limited. We only have GPS position and velocity, and power consumption. This only provides a rough estimate of the dynamics, as the GPS sensor data is not very precise. This year the car will have new electronics, gear, wheels and powertrain. It will have a Permanent Synchronous Motor (PMSM) instead of a brushed DC-motor. Given that the car is going to undergo all these changes, the vehicle dynamics will not be the same as when it competed in 2019. The main strategy for DNV GL Fuel Fighter up to this point has been to accelerate quickly, before disconnecting the motor from the wheel, and coasting for as long as possible. Thus the vehicle is designed to coast. See fig. 1 for an image of the old drive train.

This year the gear has a different design. Instead of disconnecting the motor with a linear actuator, the motor is connected to the wheel through a one-way ratchet system. This means that no regeneration of energy is possible. The motor is a PMSM from Alva Industries[1], made mainly for drones. The light weight and high efficiency are properties well suited for the needs of DNV GL Fuel Fighter. The cars autonomous capabilities are under development, but these sensors and actuators will not be equipped during the race. They may, however, be equipped during testing which can gain useful insights in how to improve the driving.

[1]`https://alvaindustries.com/`

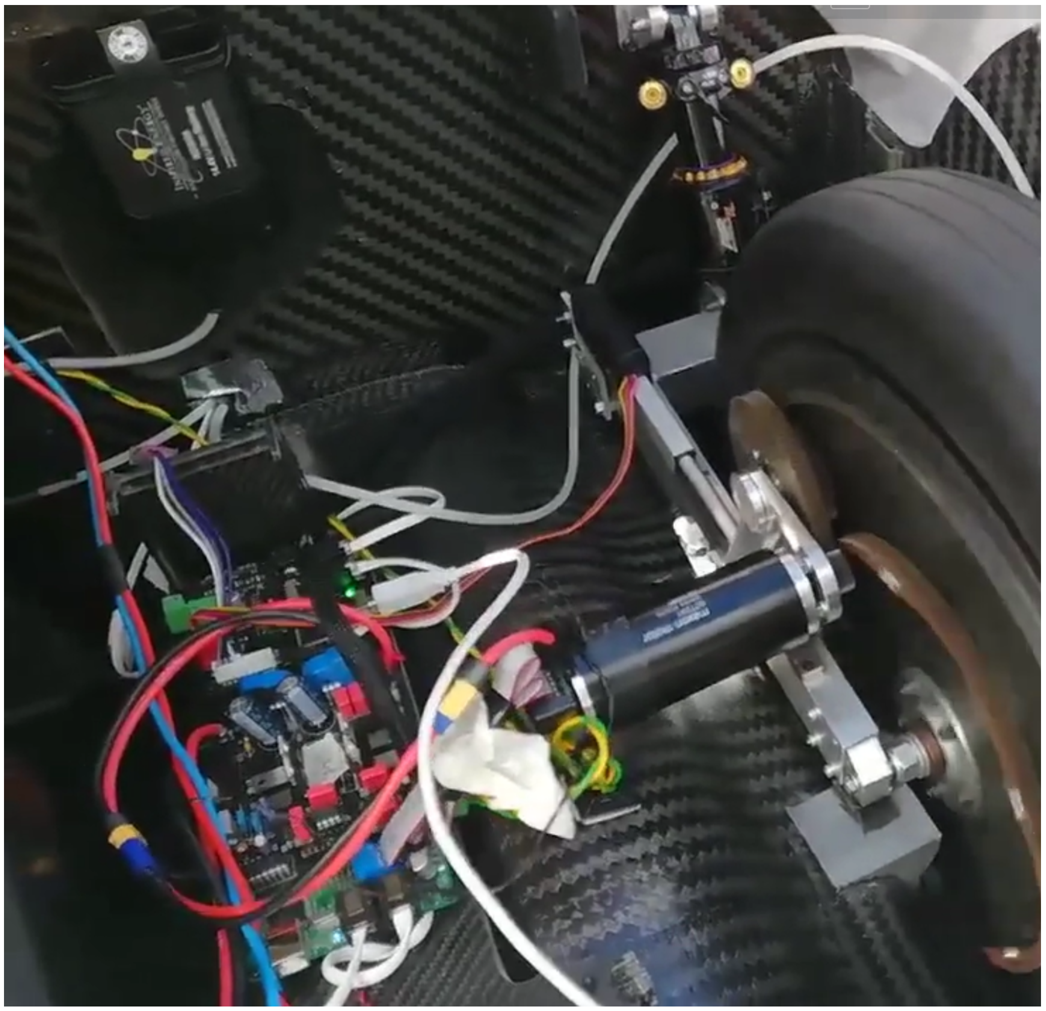


Figure 1: The old drive train system where the motor disconnects with help of a linear actuator.

2.2 Driving Strategy

The driving strategy can be found through trajectory planning, so this chapter will first describe this. Then it will proceed to explaining the racing line, and how to calculate the geometry of the racing line. This racing line will be used as the path to optimise the velocity and acceleration on. Finally, a remark on regeneration is included.

2.2.1 Trajectory Planning

The term *trajectory planning* refers to the problem of determining both a path and a velocity function. First the path is calculated in the obstacle-free space, then a velocity function is computed that fulfills the differential constraints of the system. It may also be called kinodynamic planning, except that trajectory planning bears a historical connotation of an approach that first plans a path, and then the velocity(LaVelle 2006).

2.2.2 Racing Line

In motor sport, the racing line is the optimal path on a racetrack. There are different types of racing lines that optimise for different parameters. The most important parts of the racing line are in the turns of the racetrack. Different factors that affect the line, e.g. the incoming speed, the angle of the turn, the width of the road and the following section of the race track. Turns before a straight section may sacrifice high speed in the turn, if more time can be spent accelerating out of the turn in the right direction. In this case to be able to have the largest possible action space, the maximum speed throughout the curve is desirable. This means that the geometric racing line is the most optimal. The geometric racing line provides the largest diameter in the turn, which then means that the largest speed can be kept(Paradigm Shift Driver Development 2017). This is shown in the following chapters. The radii from the circles are then connected with tangent lines, which is also described towards the end of section 2.2.2.

Maximum speed in a bend
The Critical speed equation gives the maximum speed a vehicle can travel through a bend. This is given in eq. (1)(Brach 1997).

$$v = \sqrt{r\mu g} \tag{1}$$

The derivation for this is given below. The acceleration is given by the maximum friction the wheel can have in the lateral direction. This is $F_r = m\mu g$, which is divided by the mass to get the acceleration.

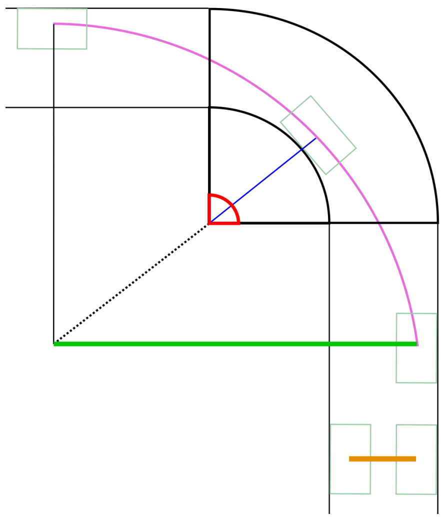


Figure 2: The figure corresponding to eq. (9). W is marked in orange, r is blue, θ is red, $R_{\max}$ is green. The box is the outline of the car.

$$a = \frac{v^2}{r} \tag{2}$$

$$\mu g = \frac{v^2}{r} \tag{3}$$

$$v^2 = r\mu g \tag{4}$$

$$v = \sqrt{r\mu g} \tag{5}$$

For the critical speed equation the maximum radius is needed. The simplest way to do so, would be to take the outer radius minus half the vehicle width. However, as mentioned earlier the geometric racing line increases the radius to more than the bend itself. This larger radius, $R_{\max}$ is derived in (Neades 2007).

$$R_{\max} = W + r + (R_{\max} - r)\cos(\theta/2) \tag{6}$$

$$R_{\max} = W + r + R_{\max}\cos(\theta/2) - r\cos(\theta/2) \tag{7}$$

$$R_{\max}(1 - \cos(\theta/2)) = W + r(1 - \cos(\theta/2)) \tag{8}$$

$$R_{\max} = r + \frac{W}{1 - \cos(\theta/2)} \tag{9}$$

where W is the width of the road minus the width of the car, r is the inner radius of the turn plus half the vehicle width and θ is the angle of the change in direction. See fig. 2.

Putting it together with the critical speed equation, the equation for max speed in a bend is:

$$v = \sqrt{\left(r + \frac{W}{1 - \cos(\theta/2)}\right)\mu g} \tag{10}$$

Tangent Calculation
Once the radii of the bends are determined, the circles are connected together with tangent lines. In the following the calculation tangent for two consequtive circles, with center point (a, b) and (c, d) with radii r_0 and r_1 respectively, will be described. To calculate the tangent line, the intersection point is first computed, given by the following formula if the intersecting point is not between the two circles.

$$x_p = \frac{cr_0 - ar_1}{r_0 - r_1} \tag{11a}$$

$$y_p = \frac{dr_0 - br_1}{r_0 - r_1} \tag{11b}$$

If the intersection point is between the circles, the intersection point can be found by:

$$x_p = \frac{cr_0 + ar_1}{r_0 + r_1} \tag{12a}$$

$$y_p = \frac{dr_0 + br_1}{r_0 + r_1} \tag{12b}$$

Tangent point for the larger circle, r_0 is given by:

$$x_{t_{1,2}} = \frac{r_0^2(x_p - a) \pm r_0(y_p - b)\sqrt{(x_p - a)^2 + (y_p - b)^2 - r_0^2}}{(x_p - a)^2 + (y_p - b)^2} + a \tag{13a}$$

$$y_{t_{1,2}} = \frac{r_0^2(y_p - b) \mp r_0(x_p - a)\sqrt{(x_p - a)^2 + (y_p - b)^2 - r_0^2}}{(x_p - a)^2 + (y_p - b)^2} + b \tag{13b}$$

The tangent point for the smaller circle, r_1 is given by:

$$x_{t_{3,4}} = \frac{r_1^2(x_p - c) \pm r_1(y_p - d)\sqrt{(x_p - c)^2 + (y_p - d)^2 - r_1^2}}{(x_p - c)^2 + (y_p - d)^2} + c \tag{14a}$$

$$y_{t_{3,4}} = \frac{r_1^2(y_p - d) \mp r_1(y_p - d)\sqrt{(x_p - c)^2 + (x_p - d)^2 - r_1^2}}{(x_p - c)^2 + (y_p - d)^2} + d \tag{14b}$$

The tangent line equations are then given by:

$$y = \frac{(x - x_{t_1})(y_p - y_{t_1})}{x_p - x_{t_1}} + y_{t_1} \tag{15a}$$

$$y = \frac{(x - x_{t_3})(y_p - y_{t_3})}{x_p - x_{t_3}} + y_{t_3} \tag{15b}$$

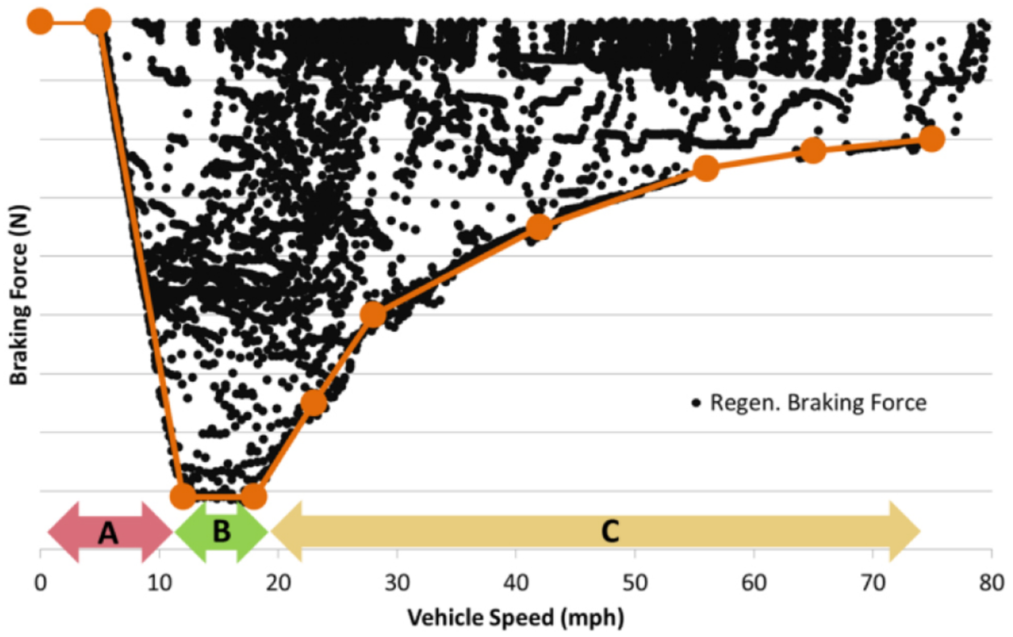


Figure 3: The envelope of regeneration force of a normal non-competitive car.

2.2.3 Regenerative braking

The current gear design does not have the possibility for regenerative braking. However, future cars from DNV GL Fuel Fighter may have that functionality, so it is worth exploring. In city driving, due to the stop-and-go traffic, the energy regain from regenerative braking can be as much as 34%(U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy 2017). This is similar to the competition, as a complete stop has to be done at the end of every lap.

Figure 3 is a typical envelope for regenerative braking force vs speed(Rask, Santini, and Lohse-Busch 2013). The profile in area A is the transition from generative braking to normal friction braking to fulfill the normal brake pedal feel. At lower speeds, the generative force is not as high, so the friction based brake is used to compensate for it to behave as one would expect from a normal brake pedal. This may not be needed in the car design for the SEM competition, as the focus on energy efficiency beats comfort. Area B is the maximum regenrative braking force area, and is not limited by the battery or drive train, which Area C is limited by. (Rask, Santini, and Lohse-Busch 2013) also mentions that rear-wheel braking may be more unstable, but this is more likely at higher speeds, above the competition limit of 40 km/h/24 mph. As the maximum speed is so low, it makes sense to use the regenerative force to make the complete stops in the race.

2.3 Mathematical Modelling

The mathematical modeling need is two-fold. It needs to take into account both the vehicle dynamics when driving on a race track, and the internal electrical system, to be able to judge the energy efficiency. The vehicle dynamics provides the constraints for the optimisation problem. Having an accurate model of the electrical system, enables creating a more correct estimate of the cost/reward for the given actions.

2.3.1 Vehicle Dynamics

A car is a non-holonomic system, meaning it has differential constraints that cannot be completely integrated. The state variables of the car are the position of the midpoint of the rear axle, x and y, the velocity of the vehicle w, the heading θ, and front wheel angle, ζ. The state equation for these states are:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{w} \\ \dot{\zeta} \end{bmatrix} = \begin{bmatrix} w\cos\zeta\cos\theta \\ w\cos\zeta\sin\theta \\ w\sin\zeta \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} u_1 + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u_2 \tag{16}$$

where u_1 is the accelerator, and u_2 is the steering force. This model can be simplified by considering the velocity as a control.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\zeta} \end{bmatrix} = \begin{bmatrix} \cos\zeta\cos\theta \\ \cos\zeta\sin\theta \\ \sin\zeta \\ 0 \end{bmatrix} w + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u_2 \tag{17}$$

Further simplification can done, by setting $v = w\cos\zeta$ and $\omega = w\sin\zeta$:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega \tag{18}$$

However the constraints on v and ω are no longer independant, so the admissable control domain is no longer convex. (Laumond, Sekhavat, and Lamiraux 1998)

Continuing from eq. (17), some bounds on the input can be set. These bounds are for the limit the maximum and minimum input to the car. Dubins car restricts the speed to only forward movement, with a maximum speed of 1. There is also set in place a max steering angle, which creates a minimum turning radius.(Dubins 1957) The Reed-Shepp Car, which builds on Dubins car, permits both forward and backward movement, up to a maximum speed of one.(Reeds and Shepp 1957)

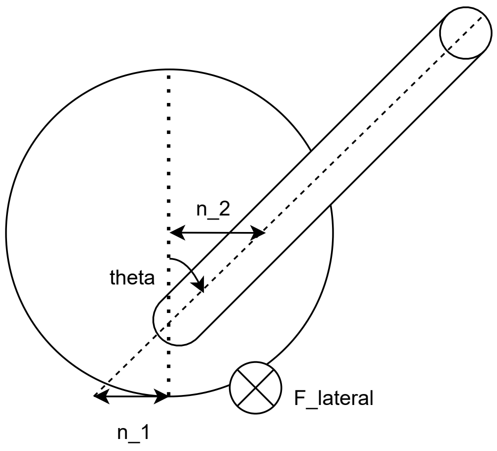


Figure 4: An overview of the parameters used in eq. (19). The dimensions and scales are exaggerated for clarity. The large circle is the wheel, and the diagonal pole is the connection to the rest of the car.

Self-centering
The self-centering property of the steering is missing from the previous models. This property is dependant on the caster angle and/or trail of the vehicle. Euromotor Colleges Automotive Engineering II course(EuroMotor n.d.) mentions a formula for the centrifugal castor force, F_{castor}:

$$F_{\text{castor}} = F_{\text{lateral}}(n_1 + n_2)\cos\theta_{\text{castor}} \tag{19}$$

where F_{lateral} is the lateral force, n_1 is the castor offset, n_2 is the wheel castor offset and θ_{castor} is the caster angle. See fig. 4 for overview. This means that the lateral force needs to be calculated, if it were to be included in the model.

Air Resistance
The formula for air resistance, F_A is given by

$$F_A = \frac{1}{2}\rho C_d A_F v^2 \tag{20}$$

where $\rho = 1.225\,\text{kgm}^{-3}$ is the density of air, $C_d = 0.15\,\text{m}^2$ is the aerodynamic drag coefficient, $A = 0.755\,\text{m}^2$ is the frontal area, and v is the velocity(Kalm 2007). The numbers are from DNV GL Fuel Fighter's own simulations and wind tunnel tests.

Rolling Resistance

$$F_{\text{rolling}} = C_{\text{rr}} F_N \tag{21}$$

where $C_{rr} = 0.0015$ and F_N is the normal force of the vehicle(Transportation Research Board 2006).

The normal force is given by:

$$F_N = m_{\text{car}} g + F_D \tag{22}$$

where g is the gravitational acceleration and F_D is the downforce. DNV GL Fuel Fighter's car aims to be downforce neutral, as both positive and negative downforce increases drag. The downforce formula is very similar to the air resistance formula, except different area and coefficient.

$$F_D = \frac{1}{2}\rho C_l A_A v^2 \tag{23}$$

where $C_l \approx 0$ is the lift coefficient, and A_A is the area from above, the bird's-eye view(Kalm 2007).

2.3.2 Electrical System

Permanent Synchronous Motor (PMSM)
The efficiency of a PMSM can be expressed as a function of the speed, N, in rotations per minute (rpm) and the torque T:

$$\eta(N, T) = \frac{\frac{2\pi N}{60} T}{(2\pi N/60)T + P_{\text{inverter}}(N, T) + P_{\text{motor}}(N, T)} \tag{24}$$

where P_{inverter} and P_{motor} are the losses of the inverter and PMSM. P_{motor} can be found in the data sheet for the motors. However Alva Industries have created an efficiency map specifically for DNV GL Fuel Fighter's use case, seen in fig. 5. The test environment is 100 kHz switching frequency, 40 °C stator temperatur, at 48 V, max 10 A. N and T are functions of time t, so the energy consumption of the drive system W is

$$W = \int \frac{\frac{2\pi N}{60} T(t)[1 - \eta\{N(t), T(t)\}]}{3600\eta\{N(t), T(t)\}} \tag{25}$$

Efficiency maps are not mathematical functions, so it has to be replaced with a lookup table.

$$W = \sum_{n=0}^{T_0/T_s} \frac{\{2\pi N/60\} T(n)[1 - \eta\{N(n), T(n)\}]}{3600\eta\{N(n), T(n)\}} \tag{26}$$

where T_0 is the operation time, and T_s is the sampling time of the operation commands.(Sato and Itoh 2015)

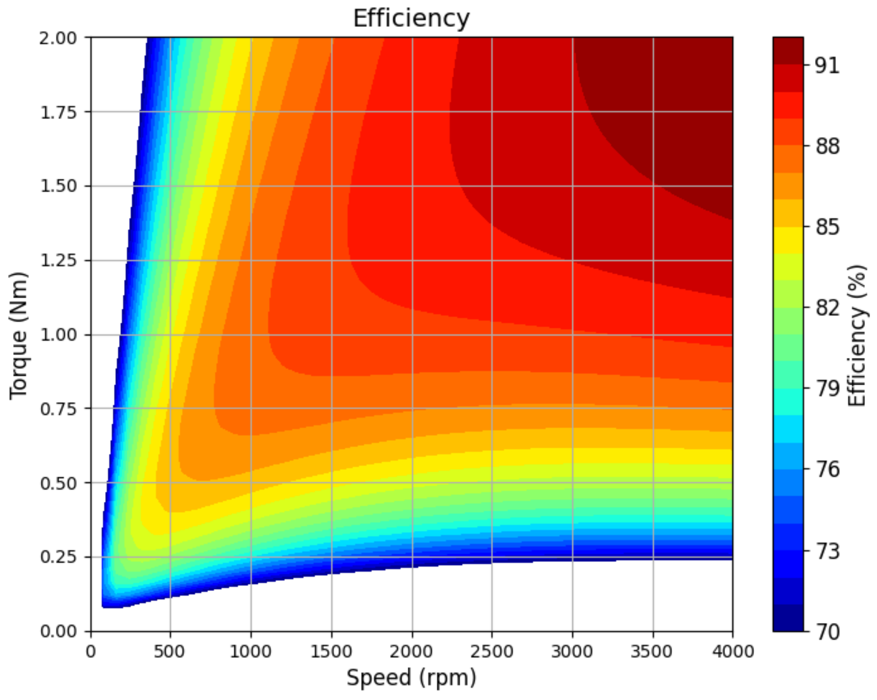


Figure 5: Efficiency map for the X100IR Permanent Synchronous Motor (PMSM) by Alva Industries.

2.4 Planning Algorithms

The planning can be done in either discrete or continuous space. Discrete planning can also be considered as graph traversal. Some of the methods used for this are blind methods like Breadth-first or Depth-first searches. The methods are blind(also called uninformed), because they don't take into account where the goal is. They search the graph until come accross the goal, and then they report success. Informed search algorithms on the other hand, takes the goal into account, often in the form of using a heuristic function. The heuristic function is a measure of whether the search is getting closer or further away from the goal. The algorithm uses this information to decide which nodes in the graph it will search first. Some informed search algorithms are A*, and D* Lite. This chapter will go through dynamic programming, rapidly-exploring tree planning methods and collision detection.

2.4.1 Dynamic Programming

Dynamic programming is a general method for optimisation that stores partial results to save on computation. It will first be explained in the discrete case before expanding to the continuous space.

A cost-to-go value is calculated for every node and stored in a table. The cost-to-go value is the minimal cost from the goal and to the current node being calculated and is given as

$$\text{cost\_to\_go}(n) = \begin{cases} 0 & \text{if goal}(n) \\ \min_{<n,m>\in A}(\text{cost}(<n,m>) + \text{cost\_to\_go}(m)) & \text{otherwise} \end{cases} \tag{27}$$

By having this cost-to-go function in a table, the optimal policy can easily be found by going through nodes with the lowest cost-to-go value.

However, the search graph has to be finite and small enough to the the data in a table. The goal should also not change very often.(LaVelle 2006)

Dynamic Programming for Continuous Spaces

Dynamic programming of continuous spaces can be done by using interpolation. When using interpolation, G_k^* is calculated and stored for a sample of points, and all the values in between are interpolated. It is important that some of the sample points are at the boundaries, and that the dispersion is low. Low dispersion means that the samples are spread out across the entire space. By having points on the boundary, you make sure that all points in the search space are between sample points.

In the simple one-dimensional case, the interpolation formula is

$$G_{k+1}^*(x) \approx \alpha G_{k+1}^*(s_i) + (1-\alpha)G_{k+1}^*(s_{i+1}) \tag{28}$$

where α is calulated as:

$$\alpha = 1 - \frac{x - s_i}{r} \tag{29}$$

The 2D-case can be written as:

$$\begin{aligned} & \qquad (30) \\ G^*_{k+1}(x) \approx & \alpha_1 \alpha_2 G^*_{k+1}(s(i_1, i_2)) \qquad (31) \\ & + \alpha_1(1-\alpha_2) G^*_{k+1}(s(i_1, i_2+1)) \qquad (32) \\ & + (1-\alpha_1)\alpha_2 G^*_{k+1}(s(i_1+1, i_2)) \qquad (33) \\ & + (1-\alpha_1)(1-\alpha_2) G^*_{k+1}(s(i_1+1, i_2+1)) \qquad (34) \end{aligned}$$

where $\alpha_{1,2}$ are similarly defined as in the one-dimensional case. As the number of interpolation neighbours grow, it is beneficial to represent them in simplexes. This poses the challenge of finding which simplex the point x is in. This means that it can only been done practically for dimensions up to six.(LaVelle 2006)

2.4.2 Rapidly-exploring Tree Planning methods

For large and/or continuous search spaces, it is impractical to search through every single point. By probing the space, either deterministically or probabilisticly, large parts of the search area can be explored, but without the great computational cost.

Basic Rapidly-exploring Tree Method
The basic structure of the rapidly-exploring planning methods are given in algorithm 1. The `Sample` and `Extend` are what differs within the family of the methods. In the case of Rapidly-exploring Dense Tree (RDT) the sample sequence is deterministic, whilst it is random in Rapidly-exploring Random Tree (RRT). (Karaman and Frazzoli 2010).

The algorithm starts by adding x_{init} as the root of the tree or graph, before incrementally adding more points to the graph. The `Sample` function decides which point will be attempted to be added next. For the RRT it samples independant, indentically distributed points from the search space. It is *attempted* to be added, because it may fail to add the point to the tree. This attempt is done in the `Extend` function. If it fails to add the point, most likely due to an obstacle, it will try to extend the tree towards the point as far as it can. The `Extend` function will be described in more detail later in this section. Gradually the tree will have more and more points and edges and cover more and more of the search space.

To create a path planner between two points, the `Sample` function can be modified to include the goal point at regular intervals. When the goal point is selected for the next extension, the algorithm will attempt to connect the goal point to the tree. The algorithm will terminate when the goal is reached, or the max number of iterations have been made.

As mentioned earlier, `Extend` function attempts to connect the new point to the tree or graph. The pseudocode for the function can be found in algorithm 2. `Nearest`(G, x_{rand}) finds the closest point or edge from the graph G, to the point x_{rand}. If an edge is closest to the x_{rand} then a new vertex is created on the edge, and returned. Finding the nearest edge is computationally heavy. More on how to do this will be discussed later.

`Steer` creates a new point, x_{new}, on the line between x_{nearest} and x_{rand}. It is normal to define a max distance that the line can go, from the nearest point towards the new point.

`ObstaceFree` checks that the line between the between from x_{nearest} to x_{new} If the line is not in any obstacles, then the line and x_{new} is added to the graph. Checking that the line is not in an obstacle is not trivial. Collision dectection will be discusseed later on, in section 2.4.3.

`Near` finds the set of all vertices within the closed ball of radius r_n, centered at x, which is given as expressed in eq. (35). γ is a constant.

$$r_n = \min\left\{\left(\frac{\gamma \log n}{\zeta_d n}\right)^{\frac{1}{d}}, n\right\} \tag{35}$$

Algorithm 1: The basic structure of the pseudocode for rapidly-exploring planning methods

```
   Input  : N, $x_{\text{init}}$
   Output: G
1  $i = 0$;
2  V = $x_{init}$;
3  G = (V, E);
4  while $i < N$ do
5      $i = i + 1$;
6      $x_{rand}$ = Sample($i$);
7      G = Extend($G$, $x_{rand}$);
```

Algorithm 2: The `Extend` method for Rapidly-exploring Random Tree (RRT)

```
1  Function Extend($G$, $x_{rand}$):
2      $x_{\text{nearest}}$ = Nearest($G$, $x_{rand}$);
3      $x_{\text{new}}$ = Steer($x_{nearest}$, $x_{rand}$);
4      if ObstacleFree($x_{nearest}$, $x_{new}$) then
5          $G$.add_vertex($x_{\text{new}}$);
6          $G$.add_edge(($x_{\text{nearest}}, x_{\text{new}}$));
7      return G
```

Algorithm 3: The `Extend` method for Rapidly-exploring Random Tree (RRT)*

```
1  Function Extend($G$, $x_{rand}$):
2      $x_{\text{nearest}}$ = Nearest($G$, $x_{rand}$);
3      $x_{\text{new}}$ = Steer($x_{nearest}$,$x_{rand}$);
4      if ObstacleFree($x_{nearest}$, $x_{new}$) then
5          $G$.add_vertex($x_{\text{new}}$);
6          $x_{\text{min}} = x_{\text{nearest}}$;
7          $X_{\text{near}}$ = Near($G, x_{new}$);
8          for $x_{near} \in X_{near}$ do
9              if ObstacleFree($x_{near}$, $x_{new}$) then
10                 c' = Cost($x_{near}$) + c(Line($x_{near}, x_{new}$));
11                 if $c' <$ Cost($x_{new}$) then
12                     $x_{\text{min}} = x_{\text{near}}$
13         $G$.add_edge(($x_{\text{min}}, x_{\text{new}}$));
14         for $x_{near} \in X_{near} \notin \{x_{min}\}$ do
15             if ObstacleFree($x_{near}$, $x_{new}$)and Cost($x_{near}$) >
                Cost($x_{new}$) + $c($Line($x_{new}, x_{new}$)$)$ then
16                 $x_{\text{parent}}$ = Parent($x_{near}$);
17                 $G$.remove_edge(($x_{\text{parent}}, x_{\text{near}}$));
18                 $G$.add_edge(($x_{\text{new}}, x_{\text{near}}$));
19     return G
```

RRT*

RRT* is based on RRT, but rewires the tree if it finds a more optimal path to the point. As it is based on RRT, the base structure is the same(see algorithm 1), but the `Extend` function differs. The pseudocode for that function can be seen in algorithm 3. It is a fair bit more complex than the normal RRT. This is because it has to not only find the nearest point, but also the group of points that are near it. Then it searches through these points to find the minimum cost edge and adds this to the graph. The next step is to go through all the other near points, and check to see if the cost is lower if it goes through the newly added point instead. If so, it will remove the old edge connection, and establish a new one with the new edge.(Karaman and Frazzoli 2010)

RRT*-Smart

RRT* is *asymptotically* optimal, meaning that it may not find a solution in a finite time frame, and converges slowly. (Nasir et al. 2013) proposes RRT*-Smart which is a variation that builds on RRT*. The main idea of the algorithm is to shorten the path, and perform intelligent sampling to be more efficient. The pseudocode for the algorithm can be seen in algorithm 4. Initially it starts of in the same manner as RRT*, sampling the space randomly, and rewires the tree when it finds shorter paths. It differs when it has found a path to the goal node for the first time. Then it begins the shortening(or path optimisation) process. It starts at the goal node, and tries to connect the parents parent node(the grandparent node) directly to the goal node. If this is succesful(meaning no obstacles in the direct path between them), it continues the search to try to connect directly to the great grandparent node. The path optimisatoin is based on the concept of triangular inequality. Triangular inequality states that for any triangle, the sum of the two lengths must be greater than or equal to the length of the remaining side.(Khamsi and Kirk 2001) If the path optimisation fails, the node that was not able to be skipped is marked as a beacon node. This beacon node is used for the intelligent sampling. The intelligent sampling is biased to sample more points in a radius from beacons along the path. Beacons are often created near obstacles, as skipping them would lead to obstacle collision. By increasing the sampling point around these beacons, it increases the likelihood of finding the optimal path around the obstacle.

RRT under differential constraints

In RRT under differential constraints the `Steer` function is replaced with a local planner instead. Instead of calculating a state trajectory, an action trajectory is calculated instead. As these differential constraints may cause the local planner to fail, an additional constraint has to be applied. The constraint prevents the planner of trying the same action, from the same point multiple times. Smoothness and continuity of the actions can be enforced by adding new phase variables. The local planner may also return a point in close proximity instead. This helps the algorithm achieve completeness, which means that it correctly identifies whether a solution exists, and returns the solution in finite time if it exists.(LaVelle 2006) The local planner uses a system simulator to find out if the action trajectory is viable. The system simulator calculate:

$$x(t) = x(t_0) + \int_{t_0}^{t} f(x(t'), u(t'))\mathrm{d}t' \tag{36}$$

Algorithm 4: The RRT*-Smart Algorithm

```
 1 $i = 0$;
 2 V = $x_{init}$;
 3 G = (V, E);
 4 $z_{\text{beacons}}$ = None;
 5 while $i < N$ do
 6     $i = i + 1$;
 7     $x_{rand}$ = Sample($i$, $z_{beacons}$);
 8     IfObstacleFree($x_{nearest}$, $x_{new}$) $G$.add_vertex($x_{\text{new}}$);
 9     $x_{\min} = x_{\text{nearest}}$;
10     $X_{\text{near}}$ = Near($G, x_{new}$);
11     for $x_{near} \in X_{near}$ do
12         if ObstacleFree($x_{near}$, $x_{new}$) then
13             c' = Cost($x_{near}$) + c(Line($x_{near}, x_{new}$));
14             if $c' <$ Cost($x_{new}$) then
15                 $x_{\min} = x_{\text{near}}$
16     $G$.add_edge(($x_{\min}, x_{\text{new}}$));
17     for $x_{near} \in X_{near} \notin \{x_{min}\}$ do
18         if ObstacleFree($x_{near}$, $x_{new}$)and Cost($x_{near}$) > Cost($x_{new}$)
             + c(Line($x_{new}, x_{new}$)) then
19             $x_{\text{parent}}$ = Parent($x_{near}$);
20             $G$.remove_edge(($x_{\text{parent}}, x_{\text{near}}$));
21             $G$.add_edge(($x_{\text{new}}, x_{\text{near}}$));
22     if $\exists x_{goal} \in G$ then
23         G, $c_{\text{new direct}}$ = PathOptimisation($G$, $x_{init}, x_{goal}$);
24         if $c_{new\ direct} < c_{old\ direct}$ then
25             $z_{\text{beacons}}$ = PathOptimisation($G$, $x_{init}, x_{goal}$);
```

For linear systems, closed form solutions can be used, but for most systems, a numerical method must be used. To reduce the search space in the action space, a set of actions can be chosen to represent the action space. Dubins car model from section 2.3.1 provides such a reduced action space. The Dubins curves can be charaterised as

$$(L_\alpha\, R_\beta\, L_\gamma,)(R_\alpha\, L_\beta\, R_\gamma), (L_\alpha\, S_d\, L_\gamma), (L_\alpha\, S_d\, R_\gamma), (R_\alpha\, S_d\, L_\gamma), (R_\alpha\, S_d\, R_\gamma), \quad (37)$$

where L, R and S are Left, Right, Straight and $\alpha, \gamma \in [0, 2\pi], \beta \in (\pi, 2\pi)$ and $d \geq 0$, which specify the length of each action. For Reed-shepps car there are 46 different curve options, which will not be shown here.

Finding Nearest Neighbour Efficienctly

To find the nearest point can be solved in two ways, either exactly or approximately. This chapter will explain an approximate method, as it is both easier to construct and computationally efficient. Each line is inserted with intermediate points, such that there is no distance further than Δq between points on the same line. The lines can be ignored, and nearest neighbour is then done on the vertices and intermediate points. The points can be put into a k-d-tree which can be considered as a multi-dimensional generalisation of binary search trees.

The pseudo code for the algoritm for the k-d-search three can be seen in algorithm 5. For a given dimension, d, the algorithm takes the median of the points along that dimension. This median is used to split the points in two, above or below the median. These two groups of points are then split along the next dimension recurively. The recursion is terminated when there are no points left. The tree is then built as a binary tree by having each child correspond to the higher or lower than the medium.

When finding the closest point in the graph, the tree can be searched to like a binary search tree to find the leaf node that is closest.

Algorithm 5: The `K-D-Tree` method for dividing up the search space to find nearest neighbours.

```
Function K-D-Tree(points, d):
    if Length(points) == 0 then
        return None;
    $p_{\text{median}}$ = Median(points, d);
    for $p \in points$ do
        if $p < p_{median}$ then
            $p_{\text{lower}}$.append(p);
        else
            $p_{\text{higher}}$.append(p);
    $d_{\text{next}} = (d+1) \mod N_{\text{dimensions}}$;
    $p_{\text{median}}.p_{\text{left}}$ = K-D-Tree($p_{lower}$,$d_{next}$);
    $p_{\text{median}}.p_{\text{right}}$ = K-D-Tree($p_{higher}$,$d_{next}$);
    return $p_{\text{median}}$;
```

2.4.3 Collision Detection

Collision detection is the computational problem of detecting the intersection of two or more objects. The majority of the computation time in planning algorithms are spent on collision detection. So doing this efficiently can reduce the computational time of planning algorithms significantly. The more complex the shape of the objects are, the more computationally expensive it is to detect collisions. For this reason, most collision detection systems use a two-phase approach, consisting of a broad phase, and a narrow phase. The broad phase uses simple shapes that cover the entire object under one. The shapes can also cover the time dimension as well, if the objects are moving. The collision detection check if any of these simple shapes overlap with other simple shaped objects. For objects far apart from each other, these will be eliminated in this phase.

The narrow phase analyses the objects that were reported to be colliding in the broad phase. Here slower, and more accurate detection algorithms are used to determine if there truly is a collision between the objects. (Mirtich 1997)

2.5 Reinforcement Learning

Reinforcement Learning will first be explained by describing MDP, which models the environment for Q-learning.

2.5.1 Markov Decision Process

A Markov Chain has the Markov property. The markov property is only being dependent on the current state and action, and not any previous states, to know all the possible outcomes. This property greatly simplifies the problem of taking the correct action, as there is no need to calculate and store previous states of the system. A Markov Decision Process is an extension of the Markov Chain, augmented to include actions and rewards. It consists of:

- A set of states, S
- A set of actions, A
- A function that specifies the dynamics, written as $P(s'|s,a) : S \times S \times A \rightarrow [0,1]$.
- A reward function $R(s,a,s') : S \times A \times S \rightarrow \mathbb{R}$ (Poole and Mackworth 2010).

2.5.2 Policy Mapping

A policy is a mapping from states to actions. It decides what action to take in the given state. A policy should be defined over the entire state-space, specifying what to do in any situation. If the policy chooses the same action, each time the specific state is visited, for all states it is called a stationary policy. A stochastic policy chooses an action from the same probability distribution for each state. An optimal policy yields the maximum possible expected reward that can be achieved, starting from that state. (Watkins 1989)

2.5.3 Q-learning

Q-learning is a method to create a policy. Q-learning uses a Q-table to decide what actions to take. The Q-table is as big as the state space times the action space. For each possible state space and possible actions from that space, there is a Q-value that represents the current and future reward for taking that specific action at that state. To find the Q-values, it needs to be learned by repeatedly exploring the environment, and updating the Q-value based on the reward received. The equation to update the Q-value, also called the Q-function is given in eq. (38)(Wikipedia contributors 2021).

$$Q_{\text{new}}(s_t, a_t) = Q_{\text{old}}(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q_{\text{old}}(s_t, a_t)) \tag{38}$$

where $Q(s,a)$ is the Q-value for the state s and action a. α is the learning rate, affecting how much the Q-value can change in each iteration. γ is the discount factor for future rewards. This adjusts how much future rewards are taken into account. The subscript t and $t+1$ indicates the current and next time step state and action.

In Q-learning there are two modes the agent can be in: exploration and exploitation. When exploring, the agent takes random actions to increase the explored area. When exploiting, the Q-table in consulted, and the action with the maximum Q-value is taken. The rate of which two modes the agent is in, is ϵ, and is called the Epsilon-Greedy algorithm. ϵ is between 0 and 1, where 1 is only exploration, and 0 is only exploitation. A normal tactic, called Epsilon-decreasing Strategy, is to have a decaying ϵ. This mean that ϵ initial value is 1, and decreases as more iterations are completed. The agent will be do more and more explotion, thus becoming more greedy over time.

2.5.4 Reward Hacking

Reward hacking is a challenge within artificial intelligence and game theory. It occurs when an agent games the reward system in such a way that it gives rewards for actions that are not intended to give reward for. OpenAI lists reward hacking as one of the five problems discussed in "Concrete Problems in AI Safety"(Amodei et al. 2016). Some examples of reward hacking they give are:

- A buffer overflow in the reward function
- A cleaning robot can
  - disables its vision, so it does not lose rewards for the mess
  - Create mess, to get more reward to clean it.

Some counter-measures OpenAI mentions are:

- Careful Engineering, to avoid buffer overflows.
- Trip Wires, to detect attempts of reward hacking
- Multiple rewards, to make it more difficult to game.

2.6 Related Work

2.6.1 The Worlds Most Fuel Efficient Vehicle - Design and development of Pac-Car II

The Worlds Most Fuel Efficient Vehicle - Design and development of Pac-Car II is a book written by the Pac-Car Team. Pac-Car II is a car that participated in SEM in 2005 and set a world record for the fuel cell prototype class. The book describes many aspects of building the car and it includes a chapter on driving strategy which will be summarised in this section.

The driving strategy chapter is written by Jérôme Bernard(Santin et al. 2007). Bernard have split the model into three submodels:

- Powertrain model
- Vehicle model
- Driving strategy model

The powertrain model included all the components along the car's power path, from fuel tank to wheel. As this car is using a hydrogen fuel cell, the details of this path is not that relevant.

The vehicle model was modeled as a sum of five forces:

- Traction force $F_{traction}$
- Aerodynamic drag $-\frac{1}{2}\rho_{air}A_f C_x V_{car}^2(t)$
- Rolling resistance $-M_{car}g\cos(\alpha_{track}(t))C_r$
- Bearing resistance $F_{bearings}$
- Slope resistance $-M_{car}g\sin(\alpha_{track}(t))$

where:

- M_{car} is vehicle mass
- ρ_{air} is air density
- A_f is frontal Area
- C_x is aerodynamic drag coefficient
- g is gravity Constant
- C_r is rolling resistance coefficient
- α_{track} is track slope

The driving strategy model had not progressed far at the time of writing the book, so there are not many details about it. Bernard expected the driver to have three data inputs to decide the current action in the track:

- Speed
- Track Position
- Time elapsed

The optimisation problem was formulated as minimise:

$$J = \int_0^{t_f} \dot{m}_{H_2}(F_{traction}(u), V_{car}(u))\,\mathrm{d}u \tag{39}$$

subject to:

$$t_f \leq t_f^{max} \tag{40}$$

$$F_{traction} \leq F_{traction}^{max}(V_{car}) \tag{41}$$

where $\dot{m}_{H_2}(F_{traction}(u), V_{car}(u))$ is the change in hydrogen mass as a function of the input action, $F_{traction}$ is the force to the wheels, and $F_{traction}^{max}(V_{car})$ is the maximum force the car can give.

Bernard proposed that the solution space is everything between two scenarios. The first scenario is driving as quickly as possible, completing the race in the shortest time possible. The second scenario is waiting until the last possible moment to start driving but still managing to complete the race within the allocated time. These two scenarios define the boundary and the space in between is sampled to reduce the search space. The dynamic programming algorithm is then used to calculate the optimal path in the solution space.

The result strategy was a mix between acceleration modes: boosting, standby and cruising. They then created a custom accelerator with the three

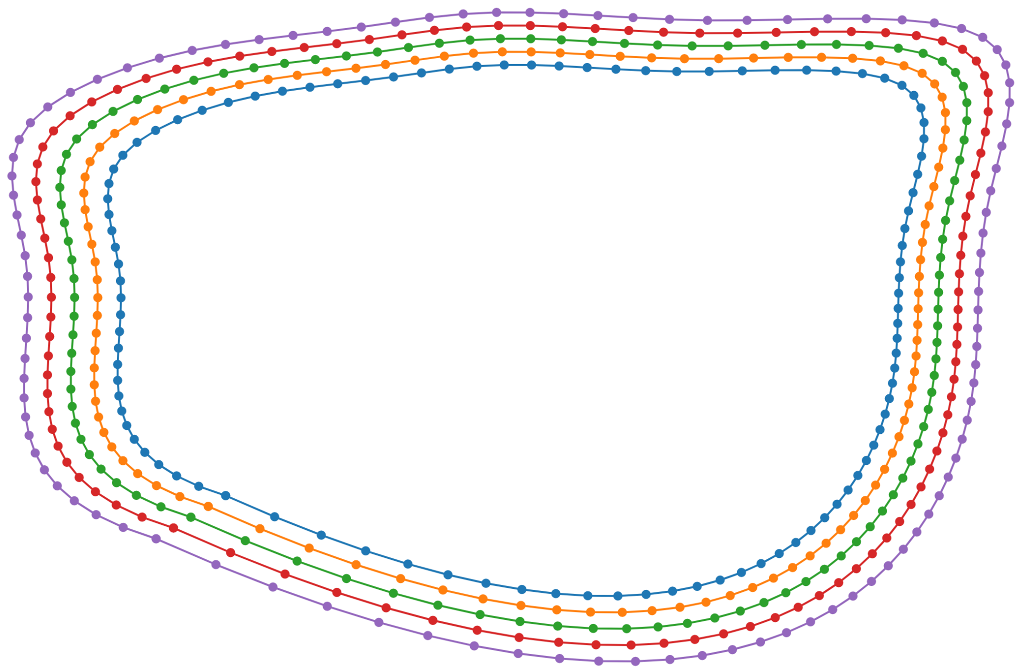


Figure 6: The generated track with five lanes, created by scaling the track within ±5%.

modes. Boosting is giving the maximum acceleration, mostly to get the vehicle up to speed. Standby is no motor power is given, and the vehicle is just rolling. Cruise is using a PI-controller to maintain the speed. Cruise was engaged when the speed fell below 9 m/s.

3 Implementation & Results

3.1 Procedural Race Track Generation and Racing Line Calculation

In order to simplify the calculations, the path is computed first, then calculating the path velocity. The path we created aimed to be similar to the racing line mention in section 2.2.2. As this needed to be done for any race track, the track was procedurally generated. Some points were randomly generated within some thresholds, and a convex hull was created to surround these points using `ConvexHull` from `scipy.spatial`. This convex hull was smoothed out with `splprep` and `splev` from the `scipy.interpolate` library. The connection where the endpoints of the spline meet are not always smooth, so sometimes a few attempts need to be made to be able to get a nice result. When this track is generated, copies are created that scale ±5% to create multiple tracks. See fig. 6 for the result.

The objective function used to calculate the racing line was the weighted

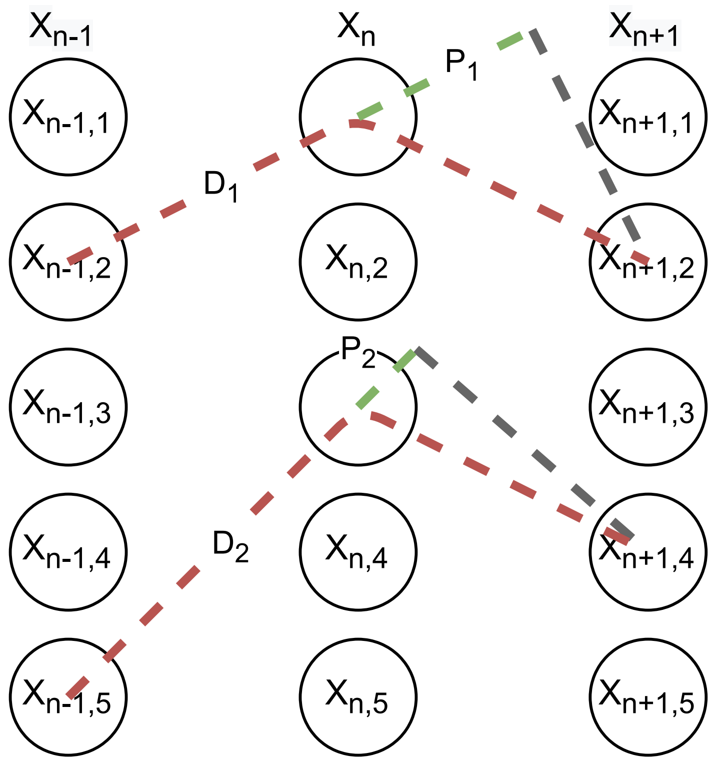


Figure 7: Objective function for raceline calculation. The green line(P_i) is subtracted from the red line(D_i). Lower cost is better, so $i = 1$ is preffered of the two that are displayed.

difference between the total length and the dot-product of the piece-wise line segments. See fig. 7 for illustration.

$$f_{\text{objective}}(x_{n-1,i}, x_{n,j}, x_{n+1,k}) = w_d D - w_p P \tag{42}$$

$$= w_d(\|x_{n,j} - x_{n-1,i}\| + \|x_{n+1,k} - x_{n,j}\|) \tag{43}$$
$$- w_p(x_{n,j} - x_{n-1,i}) \cdot (x_{n+1,k} - x_{n,j})$$

where i, j, k are the lane numbers, n is the current line segment The objective function is evaluated at each combination of previous, $x_{n-1,i}$, current, $x_{n,i}$, and next lane, $x_{n+1,i}$, for each line segment, x_n. After all these paths have been evaluated, dynamic programming is done to find the minimum path through the lap.

The initial results were not great. The first attempt was with the angle difference, instead of the dot product, seen in fig. 8.

A normalisation was added to balance the total distance, versus the dot product. The normalisation was done by calculating all the values for the total distance and dot product, and dividing them by their respective maximum. This can be seen in fig. 9. The objective function values can be seen

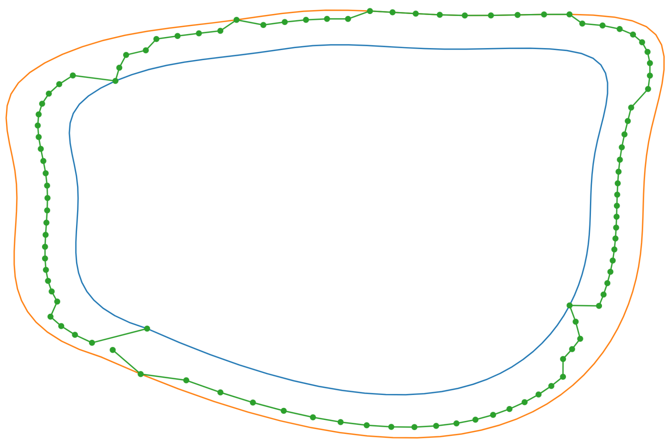


Figure 8: The first path that was generated, using the difference in angle instead of the dot product.

in fig. 10, displayed as the difference from the mean objective function value of that line segment.

Not any of these results are quite as expected or seems to be the most efficient with regards to the objective of largest turning radius and shortest distance. To verify that it at least partially works, the dot product term was removed from the objective function. This resulted in the path seen in fig. 11. From this we can see that the dot product term affects the decision in the upper left and lower right turns of the path. However we would expect it to follow the inner lane for the entire track, which is not what is happening, as it curves out in the upper right and lower left.

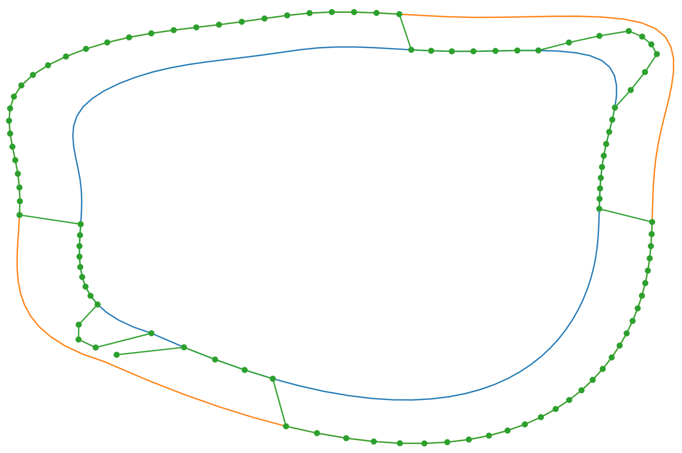


Figure 9: The resulting path when the objective terms were normalised before taking the weighted difference.

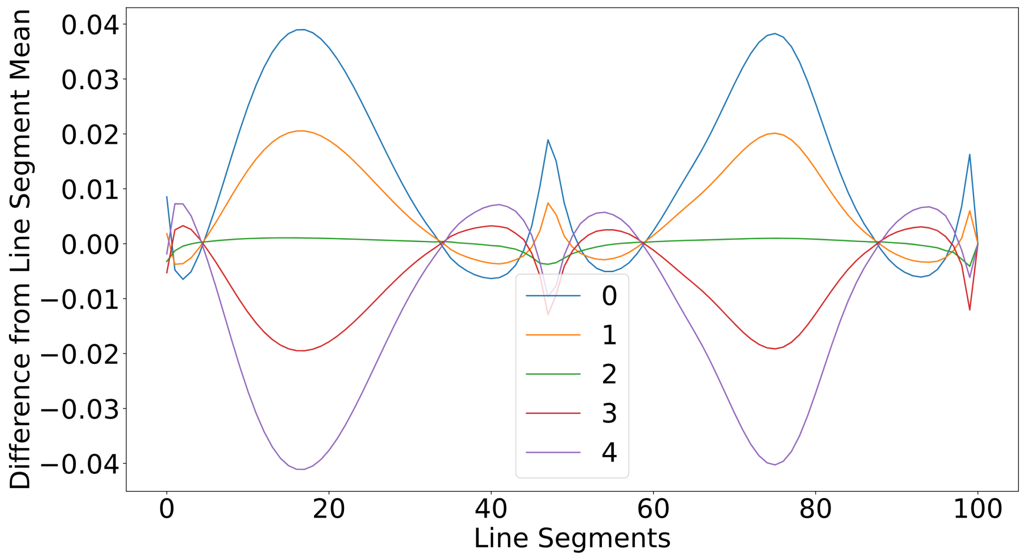


Figure 10: The difference from the mean objective function value of the current line segment. The number in the legend is the lane index.

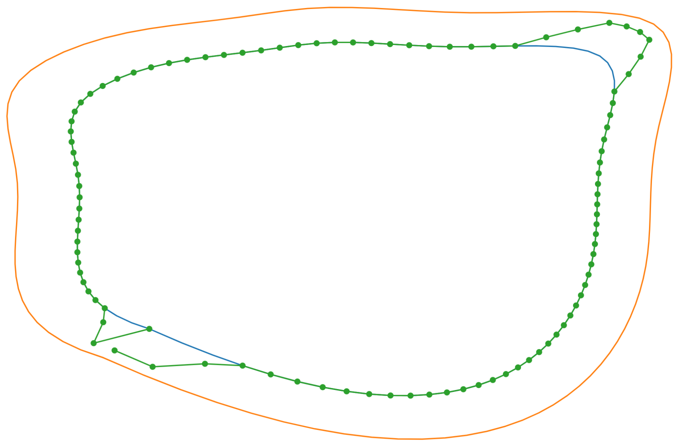


Figure 11: The resulting path with only the total distance term in the objective function.

3.2 Rapidly-exploring Random Tree (RRT) Implementation

To implement RRT, the first step was to create the algorithm that searched the space, without any goal in mind. Using the pseudocode in section 2.4.2, a jupyter notebook[2] was created that explored a 2D canvas. The first iteration did not have the `Steer` function to limit the max distance to draw the line. The nearest neighbour was only calculated on the vertices, and not the lines, causing the tree to cross itself sometimes. The resulting tree can be found in fig. 12a.

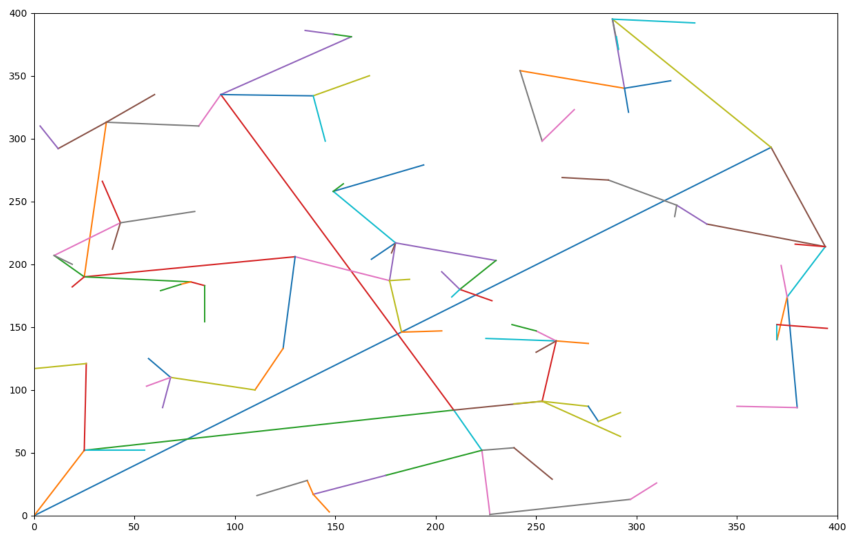
With `Steer` implemented, the graph did not cross itself as much, and it looks more like paths. See fig. 12b. A goal node was then inserted and sampled every ten iterations. This resulted in a path found in 20 iterations, as seen in fig. 13.

Next step is RRT*. The complicated thing here is to find the minimum cost path, without creating loops. Sometimes the algorithm caused loops in the tree, that needed to be prevented before this algorithm can be used. The result of the first implementatons of RRT* can be seen in fig. 14a.

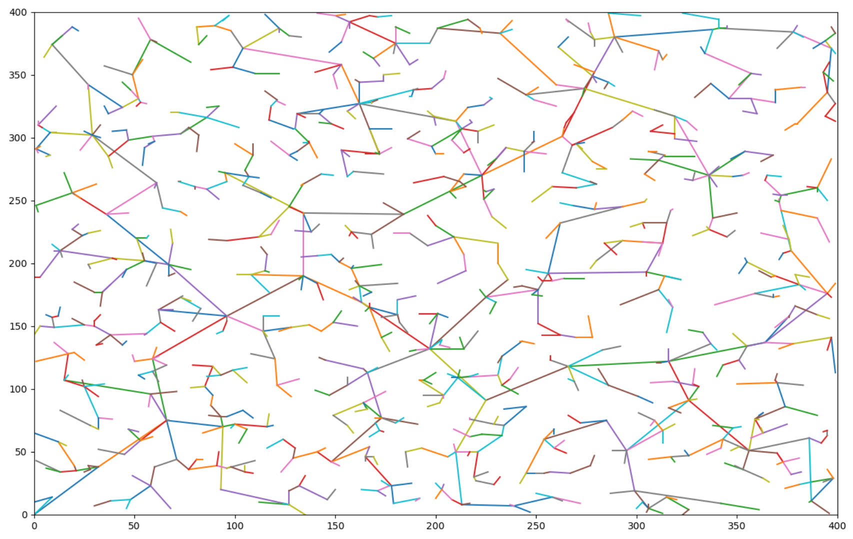
3.2.1 Parametrised track

Instead of representing the track as a set of points on a map, it can be represented as function. The input is the distance along the track as a proportion and the output is the radius at that point. If there are elevation changes along the track, the inclination can also be an output of the function. These are only two track parameters that is needed for the vehicle dynamics discussed in section 2.3.1. For straight sections of the track, the radius can be set to a large value to approximate a straight line.

[2]`https://jupyter.org/`



(a) First Rapidly-exploring Random Tree (RRT) implementation. This is about 100 iterations.



(b) Rapidly-exploring Random Tree (RRT) implementation with `Steer` function. This is 1000 iterations.

Figure 12: Rapidly-exploring Random Tree (RRT) Implementations

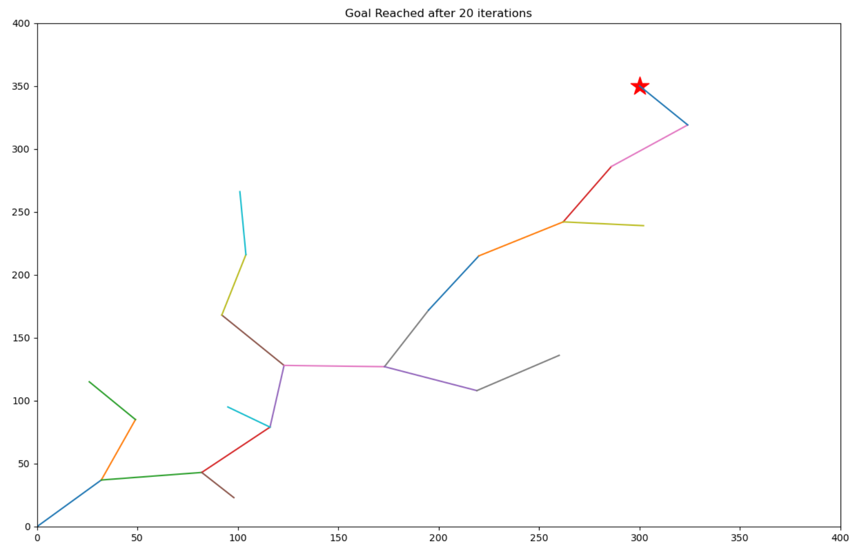


Figure 13: RRT with goal node, marked with a red star

The largest radius where the turning radius is the limit, is calculated below by using the critical speed equation.

$$40\,\text{km/h} = 11.11\,\text{m/s} = v_{\text{max}} = \sqrt{r\mu g} \tag{44}$$

$$v_{\text{max}}^2 = r\mu g \tag{45}$$

$$r = \frac{v_{\text{max}}^2}{\mu g} \tag{46}$$

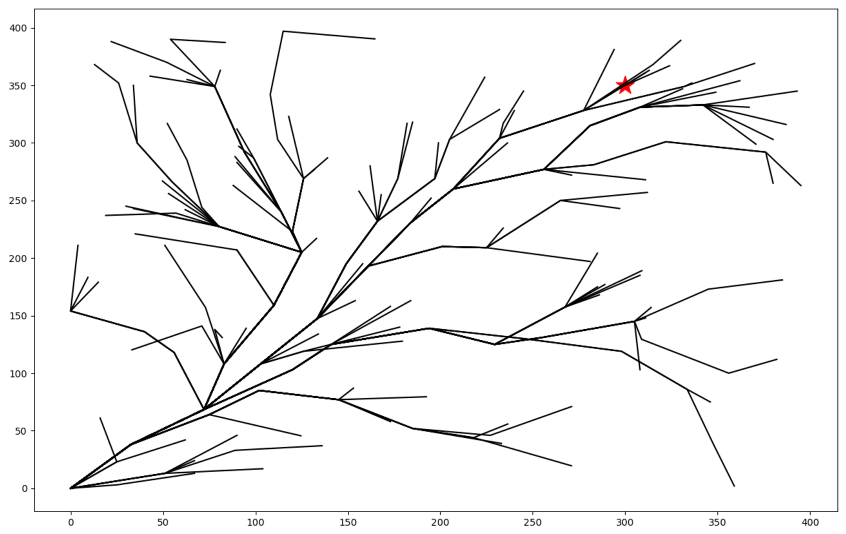
$$r = \frac{(11.11\,\text{m/s})^2}{0.04 \cdot 9.81\,\text{m/s}^2} \tag{47}$$

$$r = 28.313\,\text{m} \tag{48}$$

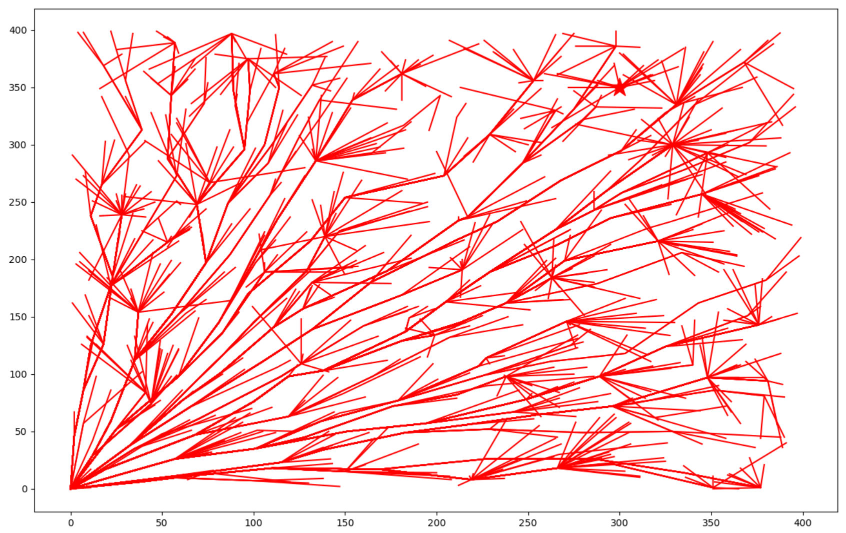
So this means for radii larger than 28.313 m, the maximum possible speed in the bend is larger than the maximum speed allowed in the competition.

3.2.2 RRT* with velocity constraints

Using the parametrisation from section 3.2.1, we can redefine the search space and set a velocity constraint. The goal is set to be 300m along the track, at zero velocity. This corresponds to a goal in position (300, 0) in fig. 15. The blue line is the maximum velocity constraint. The collision detection is not implemented on the lines, only the points, which is why some paths are crossing the velocity constraint. The differential constraints with the system simulator is not implemented, so the optimal path is a straight line to the point. However, this demonstrates how to implement a velocity constraint in the search space.



(a) 100 iterations.



(b) With 1000 iterations.

Figure 14: Rapidly-exploring Random Tree (RRT)* Implementations. Goal is marked with the red star and is in the same location in both plots.

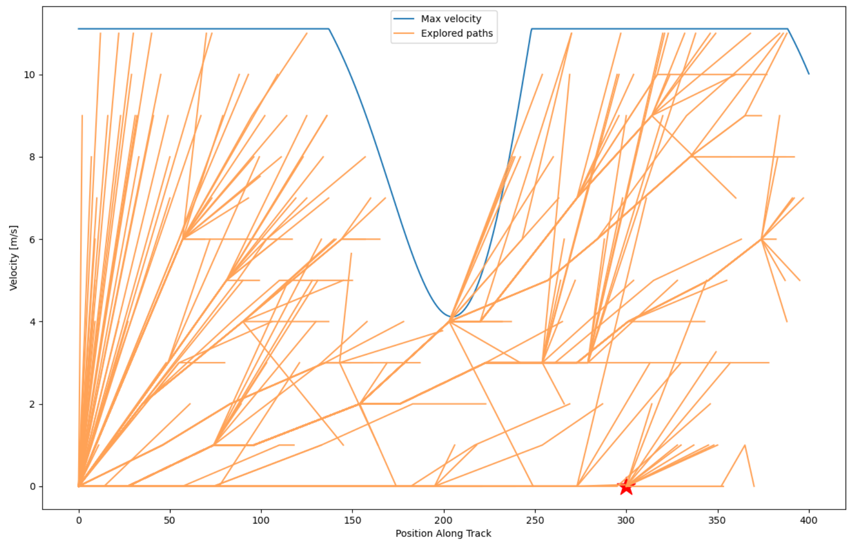


Figure 15: RRT Implementation with velocity constraint, but without system simulator. Goal is marked with the red star.

3.3 Reinforcement Learning

To be able to run reinforcement learning to optimise the driving strategy, we need a reinforcement learning environment. The most simple environment is a one dimensional path, where the speed along the path is the only thing that can change. The reward function is given as

$$R = C_m \texttt{isMoving}(x_1) + C_u|u| + C_g \texttt{isGoal}(x_0) \tag{49}$$

where `isMoving` returns true if velocity is non-zero, and `isGoal` returns true if the position is in the goal area. $C_m = 2$ is the reward for moving, $C_u = -1$ is the cost of action, and $C_g = 1000$ is the reward for reaching the goal. The tutorial from PythonProgramming(PythonProgramming 2019) was followed, as the goal of this was not to find a novel way of using Reinforcement Learning, but rather see if any useful results can be found from using it.

The results from the first run of 25000 iteratons can be seen from in fig. 16. For each iteration, the algorithm only explores 100 steps. This means that is has to move quickly to be able to reach the goal 500 units away. As can be seen in the figure, the area on the right within each plot is not as explored as the ones on the left. So the goal was moved closer to 150 units, to see if the goal reward would be more visible in the Q-values. The Q-table for 2500 iterations and the closer goal can be seen in fig. 17. Here we can see that it reached the goal in more of the iterations. The results from increasing the number of iterations to 25 000 can be seen in fig. 18. Now we can clearly see that there is a dark green prefered path through the state space. Another way to depict the Q-table, is to look that the action with the highest Q-value per state, called the policy map. This can be seen in fig. 20. Here we can see a weak white line between the blue and the red zones, which is the optimal path according to the Q-learning.

As mentioned, the car has to come to a complete stop at every lap. This means that at the goal, the velocity should be low. So the reward function was changed to only give the goal reward if the speed was below 1. The resulting Q-table can be seen in fig. 19. The policy map can be seen in fig. 21.

For all the three attempts, the same values for the hyperparameters were used. A learning rate of $\alpha = 0.1$ and a discount rate of $\gamma = 0.95$. The exploration rate, ϵ was decaying from 1 to 0 over time span of half the iterations. Meaning that in the second half, all the learning came from the updating the Q-values along the maximum value path. All this values are the same as from the original tutorial(PythonProgramming 2019).

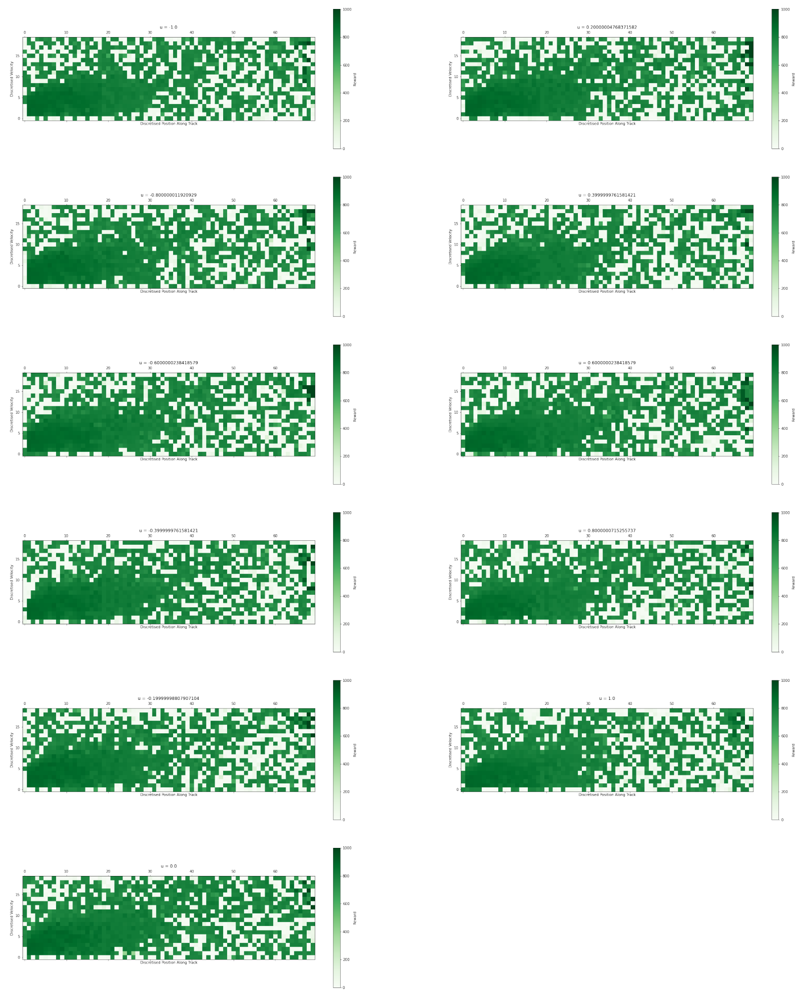


Figure 16: First Q-table with 25000 iterations. The x-axis is the position along the track, the y-axis is the velocity. Each of the subplots are the actions, in ascending column-major order where the first 5 actions are negative acceleration actions. Color is the log of the q-value.

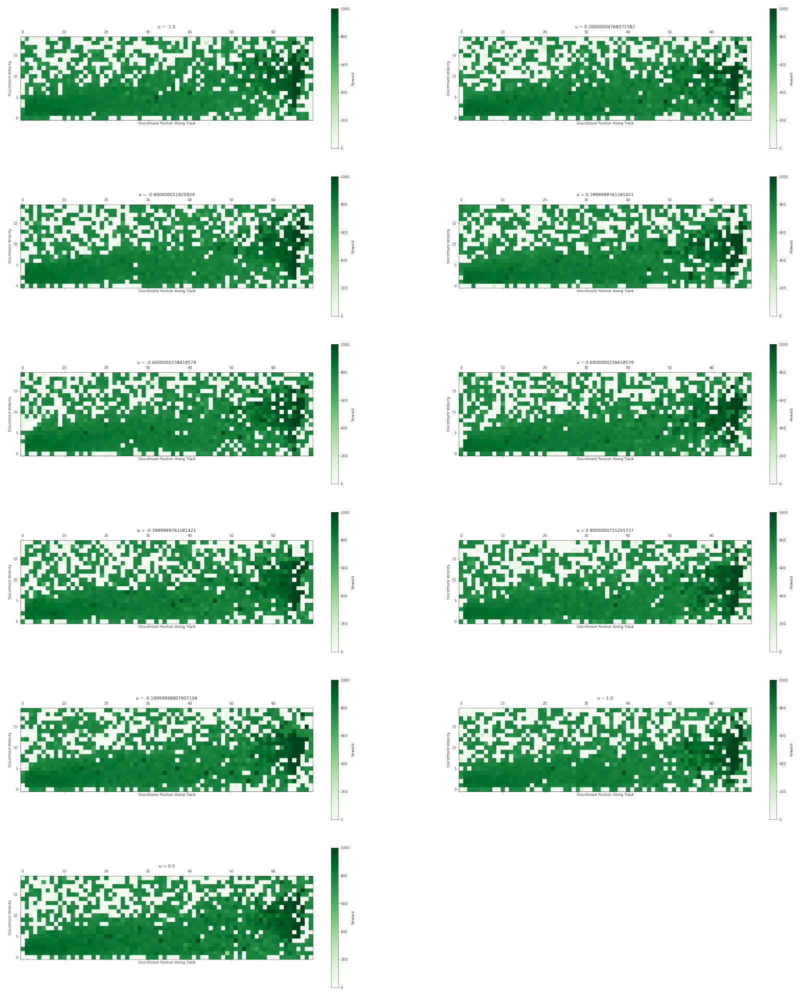


Figure 17: Q-table with 2500 iterations and closer goal. Same format as in fig. 16, except the state space is smaller, because the goal point is closer to the start.

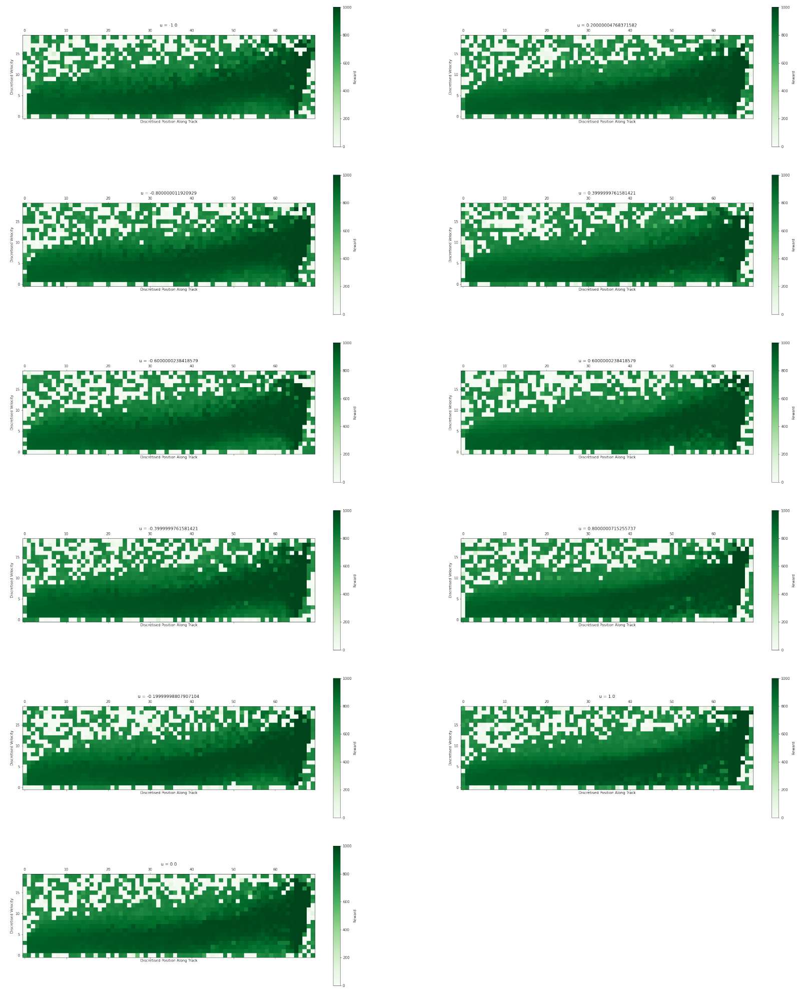


Figure 18: Q-table with *25000* iterations. Same format as in fig. 17.

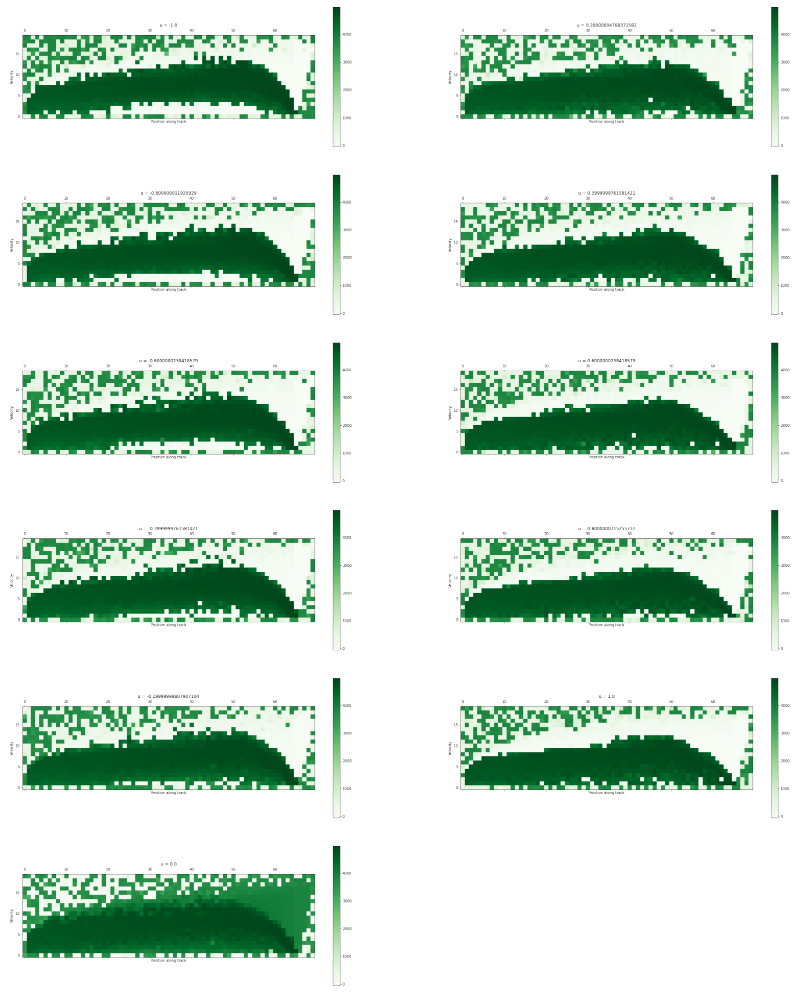


Figure 19: Q-table with 25000 iterations. This time, the goal reward is only given for if the speed of the vehicle is under 1. Same format as in fig. 17.

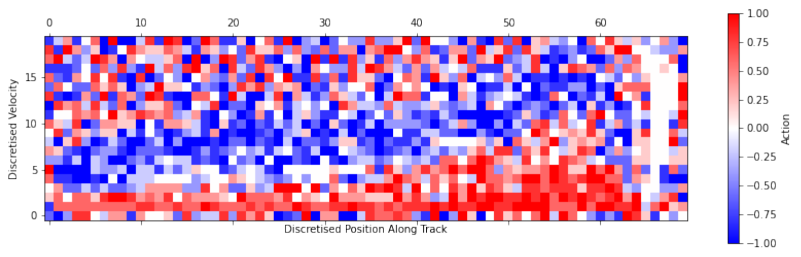


Figure 20: The action with highest reward for each state, from the Q-table in fig. 18

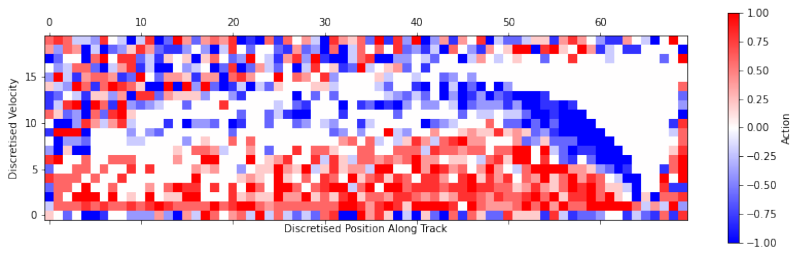


Figure 21: The action with highest reward for each state, from the Q-table in fig. 18

3.4 Crowd Sourcing Through a Simulator Competition

One of DNV GL Fuel Fighter's focus areas is the SEM communication award. The description of this award is:

> Communications and promotional activity are crucial in driving interest in Shell Eco-marathon teams and their activities; and potentially in driving sponsorship opportunities. This award provides the opportunity to reward the most impactful and successful integrated communications campaign - showing the efforts to promote the Team ahead of a potential Shell Eco-marathon competition in 2021. The winner will be the Team that demonstrates the best and most effective communication and promotional activities on their Shell Eco-marathon project.(Shell Eco-Marathon 2021)

To gain attention to what DNV GL Fuel Fighter is doing, and raise awareness of how to drive energy efficient, a simulator competition was held. The simulator was available on a website[3] for anyone to join and try to drive a track in the most energy efficient way. When they completed the race, they had the option to send their score to our highscore list. This competition could also be used to gain insight in energy efficient strategies. We used

[3]`https://metrilytics.herokuapp.com/`

Unity Analytics[4] to retrieve the driving telemetry. We could use this to see if there were any exploits of the simulator, or if it was a good strategy that scored a high efficiency. To capture the driving telemetry, trigger objects were placed with regular intervals that sent a tutorial step event[5] with the car state at that moment. Then we could infer the driving strategy based on this.

As this was only for one specific track, the results from this can probably not be used without further verification. It is however a good way to explore the space of possibilities.

The Unity physics engine handled most of the vehicle dynamics. The battery consumption, or change in charge $\dot{c}$, had to be calculated. This was done according to the formula

$$\dot{c} = c_b b v + c_a a + c_i \tag{50}$$

where $c_b = -0.5\,\%/\mathrm{m}$ is the "consumption" when braking, which is a negative value since it regenerates this energy. $b \in [0, 1]$ is the braking action controlled by the driver v is the velocity of the vehicle. $c_a = 0.02\,\%/\mathrm{s}$ is the acceleration consumption, with $a \in [0, 1]$ being the acceleration action by the driver. $c_i = 0.02\,\%/\mathrm{s}$ is the idle consumption, that is always drawn, to power on-board electronics. The numbers used here are based on trial and error when playing the simulator, to make it challenging but not impossible to complete the race.

The Unity simulator was compiled into a webGL application that is run in a python django project[6]. Django REST Framework (DRF)[7] was used to communicate with the simulator, and receive scores, that are listed on the highscore. To avoid dealing with personal information, anonymous profiles were created with each score. These profile had a random three-word subset of available words as their display name.

As when dealing with aritificial intelligence, reward hacking is a challenge. Initially the Representational State Transfer (REST) Application Programming Interface (API) used to submit the scores, did not have any authentication put into place, meaning anyone could submit scores through the API, without using the simulator. During alpha-testing, this resulted in a hacked highscore list as seen in fig. 22.

To combat this, the browsable API view from DRF was disabled to prevent easily sending requests. The DRF API Key library[8] was added, so that API keys were required to send POST requests. Finally, the POST requests also required a signature in the header to verify that the content had not been tampered with. To achieve the signature a simple tutorial was

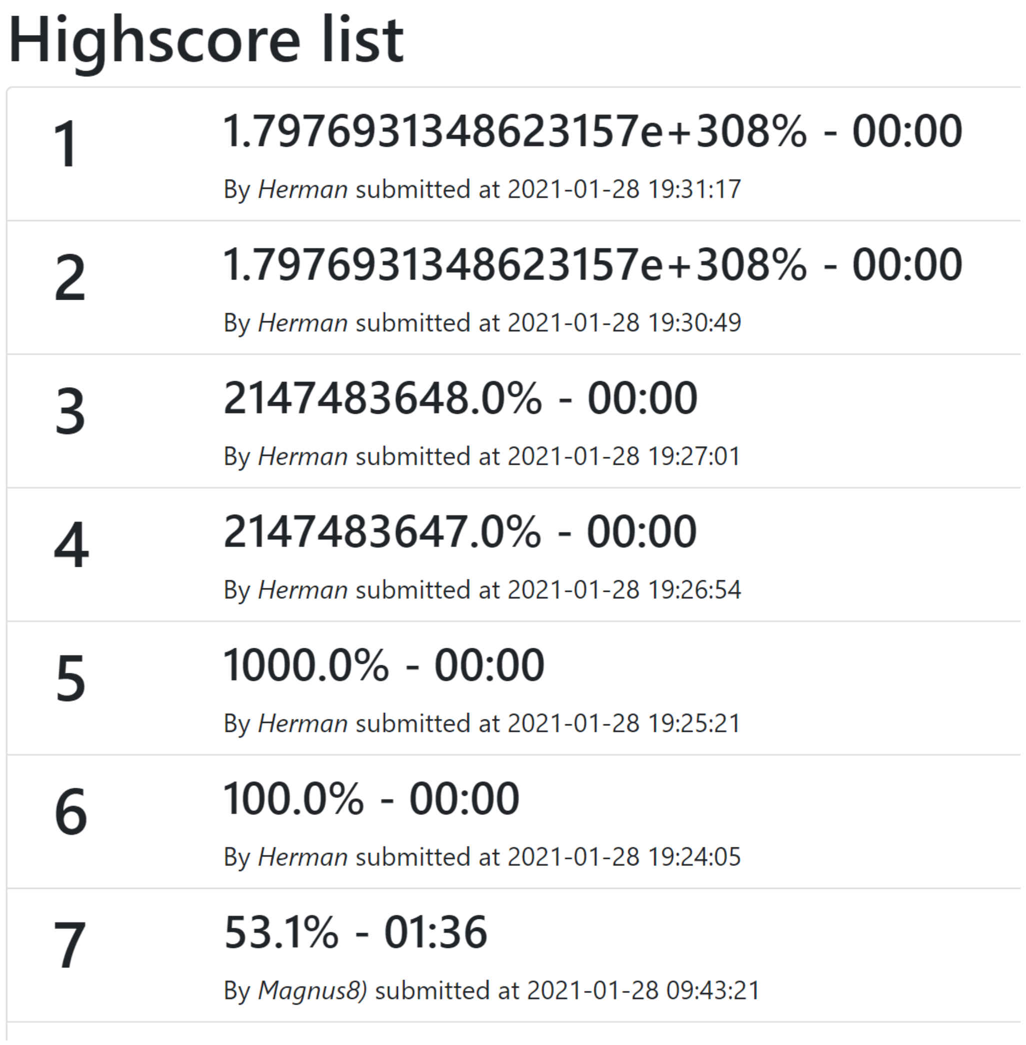
[4]`https://unity3d.com/unity/features/analytics`

[5]`https://docs.unity3d.com/2018.1/Documentation/ScriptReference/Analytics.AnalyticsEvent.TutorialStep.html`

[6]`https://www.djangoproject.com/`

[7]`https://www.django-rest-framework.org/`

[8]`https://florimondmanca.github.io/djangorestframework-api-key/`



Highscore list

1	1.7976931348623157e+308% - 00:00 By *Herman* submitted at 2021-01-28 19:31:17
2	1.7976931348623157e+308% - 00:00 By *Herman* submitted at 2021-01-28 19:30:49
3	2147483648.0% - 00:00 By *Herman* submitted at 2021-01-28 19:27:01
4	2147483647.0% - 00:00 By *Herman* submitted at 2021-01-28 19:26:54
5	1000.0% - 00:00 By *Herman* submitted at 2021-01-28 19:25:21
6	100.0% - 00:00 By *Herman* submitted at 2021-01-28 19:24:05
7	53.1% - 01:36 By *Magnus8)* submitted at 2021-01-28 09:43:21

Figure 22: The reward hacked highscore list. Rank 7 is the first non-hacked entry.

followed(Hosny 2018). This implementation was vulnerable to a timing attack, due to the comparison being a normal python implementation(`==`) that `break`s out of the comparison as soon as it detects an inequality. The `hmac` library has a method that compares two signatures in constant time, called `compare_digest()`, which eliminating the vulnerablity(Python contributors 2020).

As mentioned, Unity Analytics was used to gather the vehicle telemetry. Unity Analytics is a service offered by Unity Technologies which gives great insight into player behaviour and statistics. It is integrated with Unity, and only needs to be enabled to make use of it. There are standard and custom events that can be called in the code, that sends messages to unity analytics, together with any custom parameters set. In the unity dashboard[9] these events are captured, aggrevated and visualised. These statistics can then be exported from the unity dashboard for further data analysis. Originally 50 trigger objects were placed along the track, to have a high measuring frequency. Each lap was defined as a tutorial, so the triggers activated standard unity analytics events like `tutorial_start(id)`, `tutorial_step(id, step)` and `tutorial_complete(id)`. And the tutorial ID corresponded to the lap number. In fig. 23 plots the battery percentage and accerelation inputs for one single user driving one lap. Unfortunately, Unity Analytics have a limited the number of request allowed to 100 requests per hour per user. This means that after one race, the request limit is reached, meaning it's not possible to receive more telemetry for the next hour. A normal user session would probably try at least a couple of times to achieve a better score. The later attempts are more interesting, as the initial attempts are used to get familiar with the simulator and the track. So the number of trigger points was decreased to 6 per lap, placed in strategic locations to have a representative sampling as seen in fig. 24. The reasoning for the placement of the triggers

1. **Prehill**: Placed before the hill to get the start position. Placing it earlier would not gain more insight.
2. **Hill**: Placed in the top of the hill to get a sense of how they chose to climb the hill.
3. **Posthill**: At this point the car is either still rolling, or accelerating after regenerating in the hill.
4. **S-turn**: This point is measuring the position of where the car passes, and at what speed.
5. **Final**: Do people chose to take the turn slowly, take a wider turn?
6. **Finish**: To see what speed they enter the finish line.

Using Unity Analytics also meant that we had to have a proper privacy policy. The policy is inspired by MAKE NTNU's policy[10]. There was also an

[9] `https://dashboard.unity3d.com/`
[10] `https://makentnu.no/`

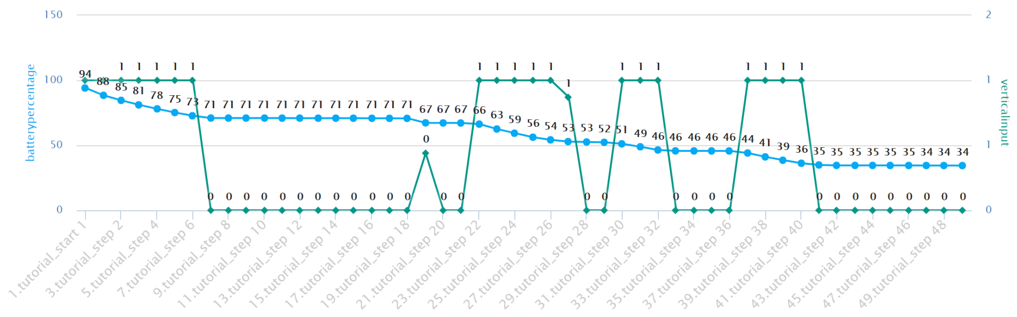


Figure 23: The battery percentage and accerelation inputs for one single user driving one lap.

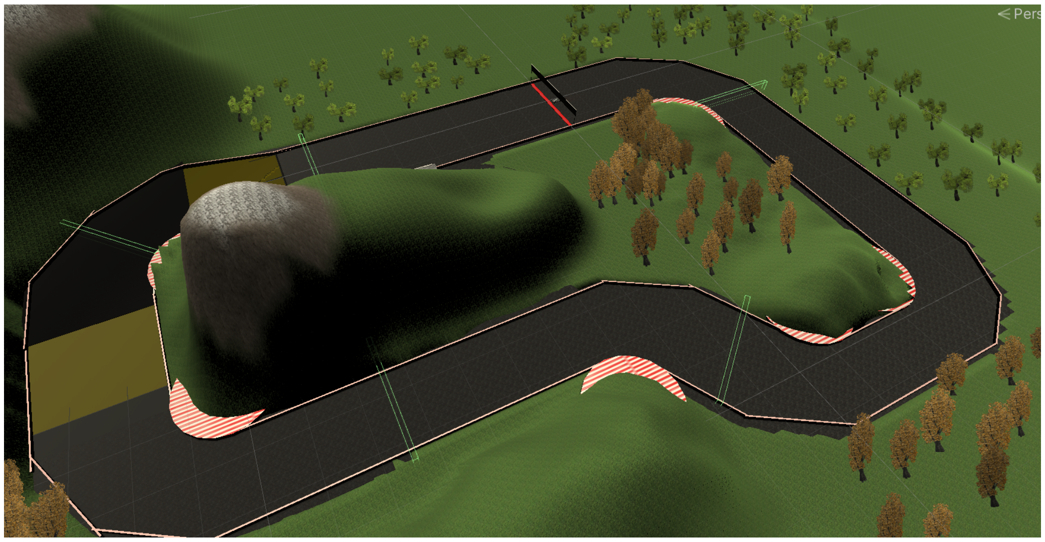


Figure 24: The track with the triggers marked in green. The sixth one is not visible, but it is at the finish line. The orange field in the track are where there is inclination

option to opt-out of data collection available in the simulator. The anonymous profiles does not have any have personal information connected to it, so it can't be used to identify the person.

3.4.1 Results

The results from the beta testing can be seen in figs. 25 to 28. Figure 25 is the position of the vehicles, overlayed with a birds-eye view of the track. The points don't line up with the trigger points because it triggers as soon as any part of the car enters the trigger object, whilst the position reported is the center of the car. As it is difficult to even complete the race, we get an indication that the attempts that did complete must have driven more energy efficiently. We can see that in the distribution of the red vs green lines, the green lines tend to follow the minimum path through the track. This is most visible in the bottom trigger point. This could be that the people who are just exploring, are not focusing on driving efficiently and therefore not on the minimum path.

In fig. 26a we see as expected that the hill takes a lot of energy to get up. It is important to have enough speed to get up the hill, because to start accelerating in the hill to prevent it from rolling down requires even more energy. The hill is also slightly slippery, causing it to be even more challenging to accelerate in the hill. This can explain why some had a higher percentage before the hill, but a lower percentage at the top of the hill, compared to those who completed the run. The best attempt manages to keep the consumption very low after the hill, meaning they use the build up of potential energy efficiently. When plotting battery percentage versus time(in fig. 26b), an interesting pattern appears. The best scores are not the ones that use the longest time. To get a clearer picture of the high efficiency races, the uncompleted races were filtered out in fig. 27b. Completing the race quicker than the available time may be beneficial to avoid the battery draining due to the constant consumption, c_i in eq. (50). The consumption calculation is called 50 times a second, and is scaled with the time in between calls. For a full duration race the idle consumption would be $c = c_i t = 0.02\,\%/\mathrm{s} \cdot 300\,\mathrm{s} = 6\,\%$, which is a significant consumption. Some of the data points did not have the racetime, so this was interpolated for clarity in the plot. This is the reason we see the vertical lines.

In fig. 28 the velocity at the trigger points are plotted. Here we clearly see that the velocity at the hill is lower than in the rest of the track. Figure 28b show only the completed runs. The best runs have the lowest speed at the hill, and regains the speed it had before the hill when it passes the trigger after the hill.

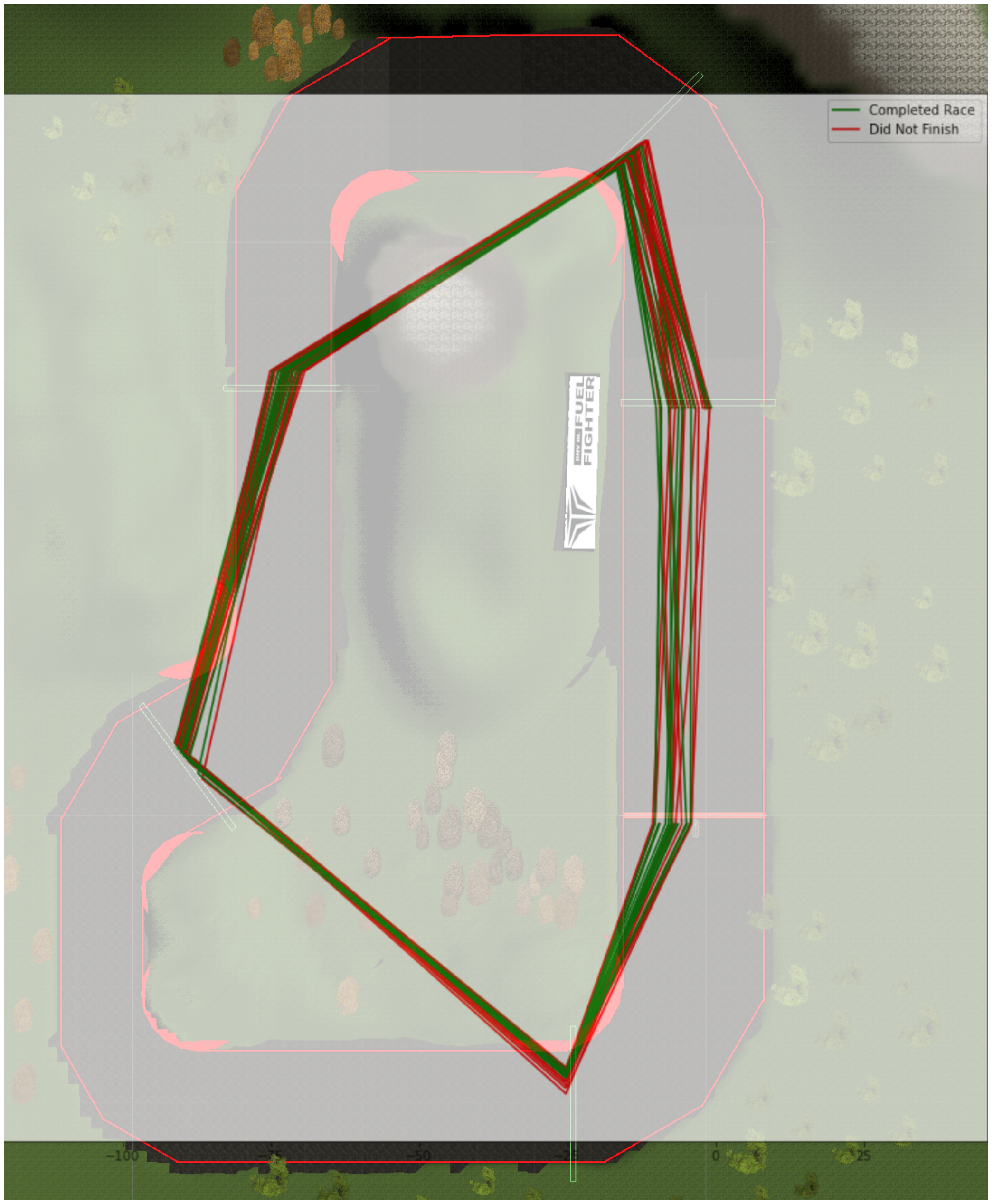
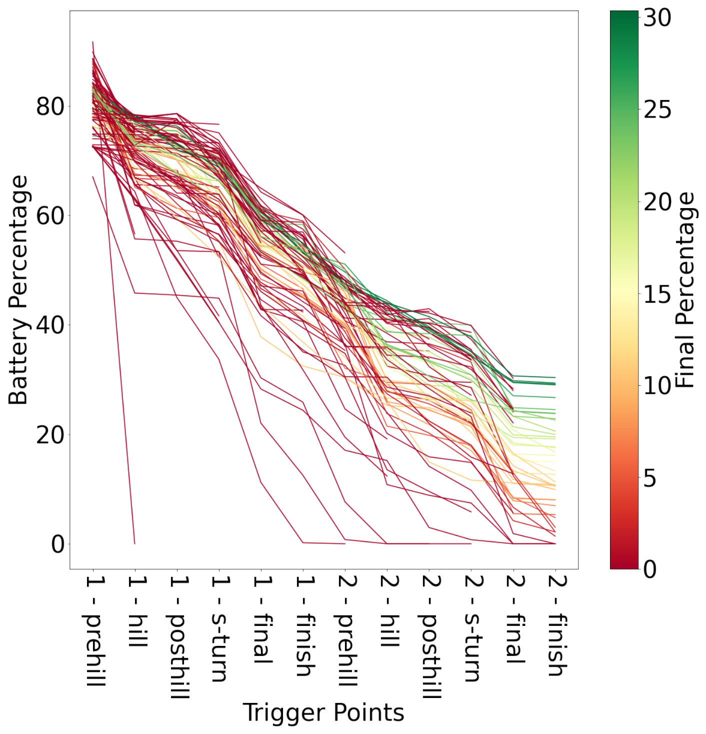
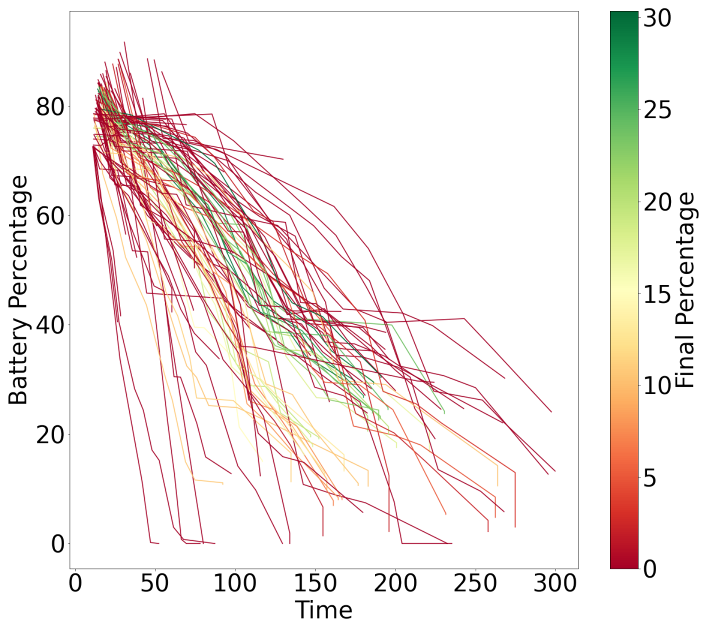


Figure 25: The position of the cars at the trigger points. The green lines are attempts that were able to complete the two laps.

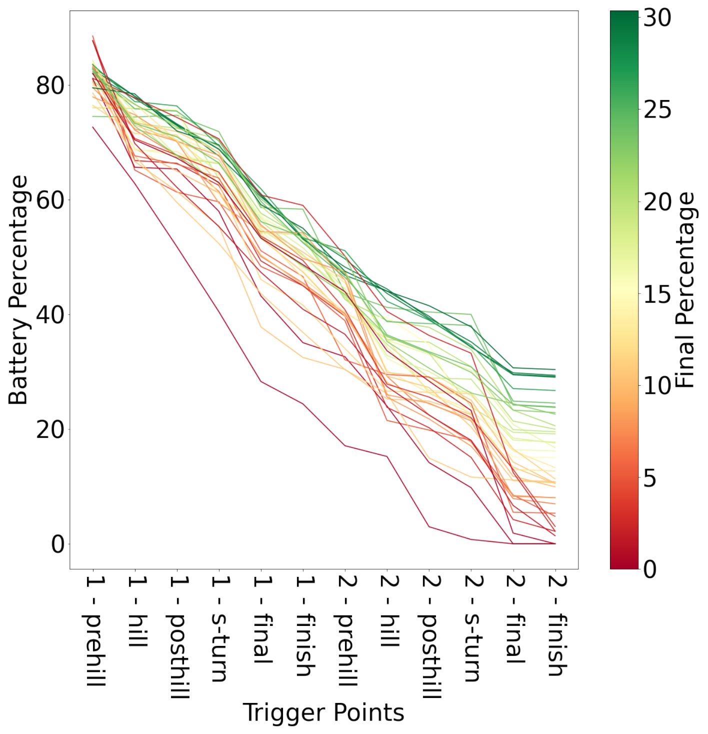


(a) The battery percentage at the trigger points. The color is the battery percentage at race completion, if completed.

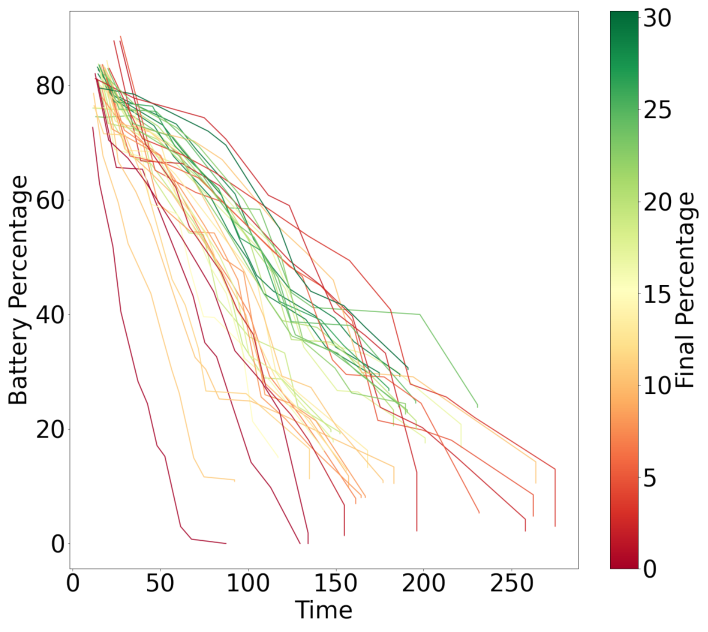


(b) The battery percentage versus time spent. The color is the battery percentage at race completion, if completed.

Figure 26: All races, including uncompleted races.

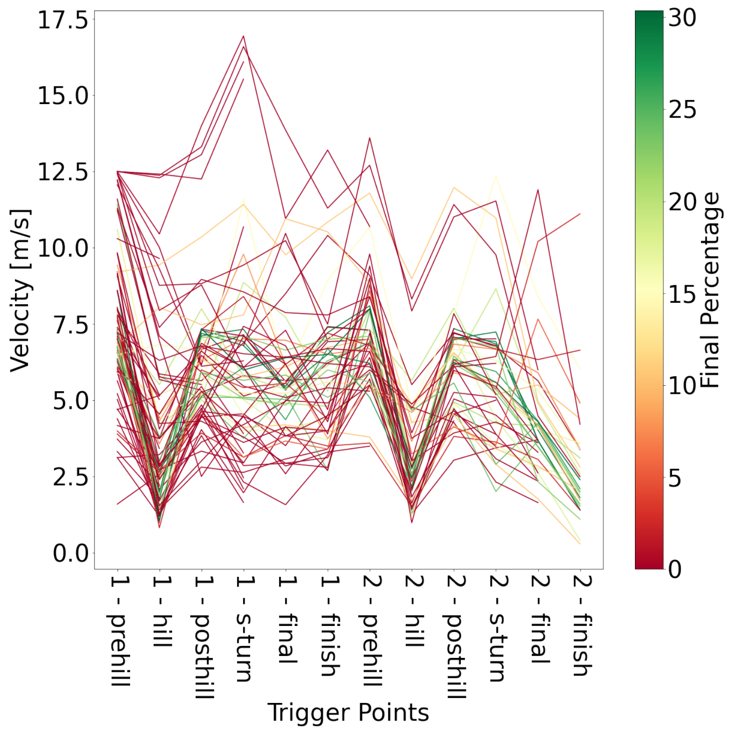


(a) The battery percentage at the trigger points. The color is the battery percentage at race completion.

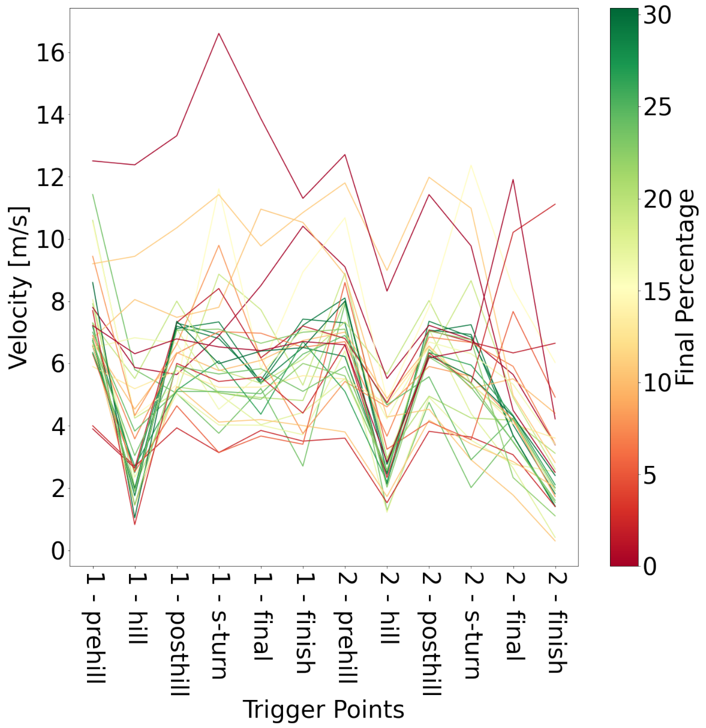


(b) The battery percentage versus time spent. The color is the battery percentage at race completion.

Figure 27: Only completed races.



(a) All races, included uncompleted races.



(b) Only completed races

Figure 28: Velocity versus trigger points

4 Discussion

4.1 Optimisation Methods

The methods I have used in this project are just a small subset of all possible ways of solving this problem. In this subchapter I will discuss the methods I chose to use and how the work can continue for the driving strategy group. All implementations are available for the members on DNV GL Fuel Fighter's GitLab software group[11].

4.1.1 Raceline calculation

The racing line calculation was done in the beginning to ease into the project by solving a discrete optimisation problem with a small search space. Unfortunately, the results were not great without any clear reason to why. This task was down-prioritised as there was other areas that need to get started on and the racing line path could also be drawn manually, and used in the other methods. For less manual work, this task could be further developed. The next step here is to create unit tests to make sure the program functions correctly.

4.1.2 Rapidly Exploring Random Tree

This approach has potential, as it searches a covers the search space quickly and works for continuous spaces. The uncertainty that still remains, is the convergence time of RRT* with the differential constraints. The remaining work here, is to implement the system simulator to satisfy the differential constraints.

4.1.3 Reinforcement learning

Reinforcement learning was implemented using Q-learning, which was a good way to be introduced to reinforcement learning, and setting up the reinforcement learning environment. The next step here is to expand the environment by adding more non-linear effects to make use of the strength of reinforcement learning. The reinforcement agent can also be more complex by moving to Deep Q-networks which are more able to represent complex environments and rewards.

4.1.4 Crowd Sourcing Simulator Competition

Crowd sourcing the strategy calculation by having a simulator competition was entertaining. It was a fun project to learn new technologies, like Django and web security with API keys and signatures. However, developing the simulator and the belonging website took quite some time away from perhaps

[11] `https://gitlab.stud.idi.ntnu.no/fuelfighter/software`

more relevant project work. The viability of this method as a whole can also be questioned. The results from the competition are only for one specific track, within game engine which is not optimised for real world simulation accuracy. There was also fewer participants than expected, meaning that the data is limited. So the conclusions one can draw from this should not be taken to be accurate. However, the fact that the slowest race is not necessarily the most efficient appoach is surprising. This may be because the ratio between the consumption rates are not properly balanced. With real data of idle consumption from the car, this may be better tuned.

The simulator competition achieved multiple objectives:

- Driving Strategy data
- Show who DNV GL Fuel Fighter is and what the team does.
- Get people thinking about how to drive energy efficiently.

Ideally there should have been more trigger points along the track. This would greatly increase the value of the data recieved. It is possible to raise the limit of request per user per hour by sending a support ticket to Unity Technologies, but this was not done, as this limit was detected to late to have time to create the support ticket in time. This should definitely be done before future simulator competitions.

Combining this simulator with reinforcement learning would also be of interest. The lessons learned from the reinforcment algorithm can be viewed in the simulator, and the driver can be trained in the simulator to achieve the same results.

4.2 Usefulness For DNV GL Fuel Fighter

As this was the first Master's thesis within driving strategy for DNV GL Fuel Fighter, there was not much to build on. There are many possible methods and solutions to this problem. This project have explored some of them and gives future members gives the background and knowledge to quickstart the process. There are still many possible directions to expand on this work. As mentioned, one of the objectives that were fulfilled, was to showcase DNV GL Fuel Fighter and what DNV GL Fuel Fighter do. This contributes to our work to focus on the communication award. This competition can be hosted again in the future, with other tracks, or driving paramters.

4.3 My Work Method

Solving this task was challenging as I started with not much background knowledge of the subject. Including multiple approaches to this was both useful and more exciting. If one approached halted, I could continue on other approaches. This may have hurted the project as well, as the approaches are not as complete as they could have been, but that is a trade-off that is worth it. Had I done the thesis again, I would start with trying to adapt the helicopter exercises from *TTK4135 Optimization and Control*. This would give me an introduction to the problem through something I already am familiar with. The book Planning Algorithms from Steven LaVelle was a useful resource for getting the project on track. I received the book recommandation from a meeting with Anastasios Lekkas(Minutes of Meeting in appendix A). I could have greatly benefited from contacting more people at the Department of Engineering Cybernetics.

5 Conclusion

Through this report I have presented varying forms of optimisations and driving strategies for DNV GL Fuel Fighterthrough:

- Dynamic Programming
- RRT*
- Reinforcement Learning
- Crowd sourcing through a simulator competition.

The implementation in this report is not enough to give a concrete driving strategy for the race this summer, but it is achievable with some further work by the driving strategy group in DNV GL Fuel Fighter. RRT* and can be further improved to minimise convergence time and optimality. A mathematical model of the car, vehicle dynamics, and the electrical system has been presented. The simulator competition achieved many of the objectives for DNV GL Fuel Fighter:

- Driving Strategy data
- Show what DNV GL Fuel Fighter does
- Get people thinking about how to drive energy efficiently.

This thesis has focused on the off-track, pre-race phase. Other areas to explore is on the on-track, mid-race phase Model Predictive Control (MPC) algorithm to calculate the corrected optimal path.

6 Bibliography

Amodei, Dario, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. 2016. "Concrete Problems in Ai Safety." `http://arxiv.org/abs/1606.06565`.

Barkenbus, Jack N. 2009. "Eco-Driving: An Overlooked Climate Change Initiative."

Brach, Raymond M. 1997. "An Analytical Assessment of the Critical Speed Formula." In *SAE International Congress and Exposition*. SAE International. `https://doi.org/https://doi.org/10.4271/970957`.

Dubins, L. E. 1957. "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents." *American Journal of Mathematics*, 497–516.

EuroMotor. n.d. "Castor Angle, Castor Trail in Wheel Centre,Castor Offset, Wheel Castor Offset and Lateral Force." Accessed January 2, 2021. `https://www.euromotor.org/mod/resource/view.php?id=21552`.

Hosny, Ahmed. 2018. "API Request Signing in Django." July 17, 2018. `https://medium.com/elements/api-request-signing-in-django-bc9389201871`.

Kalm, Elle. 2007. "Design of an Aerodynamic Green Car." `http://urn.kb.se/resolve?urn=urn:nbn:se:ltu:diva-43779`.

Karaman, Sertac, and Emilio Frazzoli. 2010. "Incremental Sampling-Based Algorithms for Optimal Motion Planning."

Khamsi, Mohamed A., and William A. Kirk. 2001. *An Introduction to Metric Spaces and Fixed Point Theory.*

Laumond, Jean-Paul, S. Sekhavat, and Florent Lamiraux. 1998. "Robot Motion Planning and Control - Guidelines in Nonholonomic Motion Planning for Mobile Robots," 1–53.

LaVelle, Steven. 2006. *Planning Algorithms.* Cambridge University Press.

Mirtich, B. 1997. "Efficient Algorithms for Two-Phase Collision Detection." In.

Nasir, Jauwairia, Fahad Islam, Usman Malik, Yasar Ayaz, Osman Hasan, Mushtaq Khan, and Mannan Saeed Muhammad. 2013. "RRT*-Smart: A Rapid Convergence Implementation of Rrt*."

Neades, Jon. 2007. "Maximum Speeds for Bends Impact Vol. 2 No 1." *Impact Vol. 16 No. 1.*

Paradigm Shift Driver Development. 2017. "THE Racing Line - Four Elements of a Perfect Corner." Youtube. 2017. `https://www.youtube.com/watch?v=N8qBdOs0s1E`.

Poole, David L., and Alan K. Mackworth. 2010. *Artificial Intelligence: Foundations of Computational Agents, 2nd Edition.*

Python contributors. 2020. "Hmac — Keyed-Hashing for Message Authentication." May 27, 2020. `https://github.com/python/cpython/blob/db5aed931f8a617f7b63e773f62db468fe9c5ca1/Doc/library/hmac.rst`.

PythonProgramming. 2019. "Q-Learning Introduction and Q Table - Reinforcement Learning with Python Tutorial." `https://pythonprogramming.`

net/q-learning-reinforcement-learning-python-tutorial/.

Rask, E., D. Santini, and H. Lohse-Busch. 2013. "Analysis of Input Power, Energy Availability, and Efficiency During Deceleration for X-Ev Vehicles." *SAE Int. J. Alt. Power.*, 350–61.

Reeds, J. A., and L. A. Shepp. 1957. "Optimal Paths for a Car That Goes Both Forwards and Backwards." *American Journal of Mathematics*, 497–516.

Santin, J. J., Christopher H. Onder, J. Bernard, D. Isler, P. Kobler, F. Kolb, N. Weidmann, and L. Guzzella. 2007. *The Worlds Most Fuel Efficient Vehicle - Design and Development of Pac-Car Ii.* Verlag der Fachvereine Hochschulverlag AG an der ETH Zurich.

Sato, D., and J. Itoh. 2015. "Evaluation Method of Energy Consumption for Permanent Magnet Synchronous Motor Drive System." In *IECON 2015 - 41st Annual Conference of the Ieee Industrial Electronics Society*, 005267–72. https://doi.org/10.1109/IECON.2015.7392929.

Shell Eco-Marathon. 2021. "2021 Virtual Off-Track Awards." 2021. https://www.makethefuture.shell/en-gb/shell-eco-marathon/2021-programme/virtual-programme-2021/virtual-off-track-awards.

Transportation Research Board. 2006. "Tires and Passenger Vehicle Fuel Economy – Special Report 286." 2006. http://onlinepubs.trb.org/onlinepubs/sr/sr286.pdf.

U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy. 2017. https://www.fueleconomy.gov/feg/atv-ev.shtml.

Watkins, Christopher. 1989. "Learning from Delayed Rewards," January.

Wikipedia contributors. 2021. "Q-Learning — Wikipedia, the Free Encyclopedia." https://en.wikipedia.org/w/index.php?title=Q-learning&oldid=998305166.

A Minutes of Meeting with Anastasios Lekkas

Meeting was held 2020-11-24, on Teams.

Minutes of Meeting

With Anastasios Lekkas(https://www.ntnu.no/ansatte/anastasios.lekkas)

Minimal distance may be enough.

Some students have used MPC for autonomous docking.

Approximate to a convex, important to look at it as the shape, not as a particle. Use MPC approach.

Include energy consumption in the objective function. Numerical methods like this are extremely strong.

Two approaches

Warm-Started Optimized Trajectory Planning for ASVs gives a quick overview of the methods.

Road map method, give waypoints. Connect them with vehicle constraints afterwards

Difficult to formulate energy.

Optmisation control as with vehicle dynamics as constraints

Find yourself into local optmimum

Need a good problem statement. If you cannot approximate energy, go for minimum distance.

optimal control best if good models in the objective function.

Switching systems

1 model for the engine mode, and 1 for the engine free mode. Changed control mode.

Try reinforcement learning. The reinforcement model don't need a model, but a good simulator.

Formulate problem, build reward function. something close to energy consumption. Try first without switch modes.

Other Resources

Steven LaValle - planning algorithms

hybrid systems - andrew teel

At IDI: Frank Lindset computer vision, working on autonomous car

Markus Anthony Dørheim Ho-Yen

NTNU
Norwegian University of
Science and Technology

DNV GL FUEL FIGHTER