Technische Universität München
Department of Electrical Engineering and Information Technology
Assistant Professorship of Hybrid Control Systems

Adaptive path planning for autonomous vehicles

Master's Thesis

Author: Yassine Hamza

Technische Universität München
Department of Electrical Engineering and Information Technology
Assistant Professorship of Hybrid Control Systems

Adaptive path planning for autonomous vehicles

Master's Thesis

Author: Yassine Hamza
Supervisor: M.Sc. Mahmoud Khaled
Advisor: Prof. Dr. Majid Zamani
Submission date: 07.01.2019

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 07.01.2019 Yassine Hamza

Acknowledgments

I would first like to thank my thesis advisor Phd student Mahmoud Khaled of the Assistant Professorship of Hybrid Control Systems. His door to office was always open whenever I ran into a trouble spot or had a question about my research or writing. I would also like to thank Prof. Dr. Majid Zamani who was involved in the validation of this thesis. Without their passionate participation and input, the validation survey could not have been successfully conducted.

Finally, I must express my very profound gratitude to my parents and to my friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Thank you
Yassine Hamza

Abstract

The introduction of self-driving vehicles in the automotive market raises many concerns regarding safety. This is due to the black-box behaviour of the deep learning systems, which are used to control these vehicles. The aim of this master's thesis is to make a contribution towards providing safety guarantees for such systems using formal methods. In order to accomplish this, a deep-learning based system along with the symbolic controller synthesis tool, scots, is integrated into a highway traffic simulation. The deep-learning based system is used for the path planning of a vehicle while scots and a hard-coded safety shield are used to provide safety guarantees by modifying biased outputs. The results shows that using scots and this safety shield for monitoring the deep learning based system was successful. However, this is only a proof of concept. Indeed, many adjustments need to be performed to make this contribution relevant for real life settings.

Contents

1 Introduction

Fully autonomous vehicles are expected to have a significant impact on several markets and industries [11]. The recent findings in the fields such as Artificial Intelligence (AI), measurement systems and sensor technologies made such predictions conceivable [4, 20]. However, deep learning-based designs for self-driving vehicles do not provide safety guarantees. Indeed, the black box behaviour of these systems makes them hard to formally verify and, hence, they can not be certified [19, 7]. In order to make a contribution towards increasing demand for safety guarantees, SCOTS, a tool for the synthesis of symbolic controllers, along with an AI-based design were successfully integrated into a 3D-highway traffic simulator [5].

One feasible solution is to deploy formally verified designs along with the AI-based prototypes. These designs would either monitor the output generated by the faulty prototype (AI design in this case) as it is done in [34], or to be used as a majority-vote system, as in [21]. In this thesis, the monitoring methodology is implemented. Indeed, the purpose of this work is to increase the robustness of autonomous vehicles in the absence of a proper methodology for certifying AI-based systems.

In a previous study, a method using reachability analysis for online verification of an autonomous vehicle was implemented in order provide more safety guarantees [1]. Despite the fact that the results of this study is conducive to the field, the high computational costs of such approach makes it difficult to be implemented in a real-life setting [29]. To overcome this issue, first, a synthesis tool, namely SCOTS, is used to determine the safest path to reach a predetermined target while avoiding obstacles (e.g., other cars). Afterwards, two recurrent neural network are trained to achieve the same requirement. Finally, the AI design is integrated into the simulation for delivering the path, which is monitored by a formal-safety shield. Here, the hard-coded shield decides if the trajectory suggested by the neural networks is safe. If this is not the case, SCOTS's synthesized path is taken instead. The results show that the vehicle drives autonomously without violating any safety requirements using the formal planner alone or the formal planner and the AI design combined.

This thesis is structured as follows. First, the controller synthesis method is introduced. Second, the work process of the symbolic controller synthesis tool SCOTS is discussed. Then, the integration of SCOTS in the highway traffic simulation is explained in details. Then, an introduction to recurrent neural networks as well as their integration in the simulation is presented. Furthermore, the results of the thesis and an overview of the work methodology are presented. Finally, some perspectives regarding potential improvements for future work are discussed.

2 Controller synthesis

In this chapter the controller synthesis method is presented. First, the advantages and the drawback of this method are discussed. Second, the Linear Temporal Logic (LTL) is introduced as a mean of describing formal requirements. Finally, two different approaches to the controller synthesis are introduced.

2.1 Why synthesis?

Constructing correct systems by manually writing control software for a safety-critical application is a difficult task. It requires translating abstract requirements into code. Afterwards, the code will need to be verified against safety requirements. Both steps are repeated in a close loop process in order to fix potential bugs. This translates to an expensive and time-consuming design phase.

To overcome above mentioned issues, an automated controller synthesis method is introduced. Here, given some formal specifications, the control code is generated automatically and the formal verification process is already embedded in the synthesis. Indeed, if the controller synthesis fails, then the specifications, which are encoded in LTL, would need to be refined [24].

However, translating requirements into LTL requires also expertise and intensive training. This is due to the shift in the level of abstraction i.e. from writing code to writing formal specification [8]. An additional drawback is the computational complexity of generating controllers using state-of-the-art LTL synthesis algorithms, which is of a time-complexity of 2-EXP-time [25].

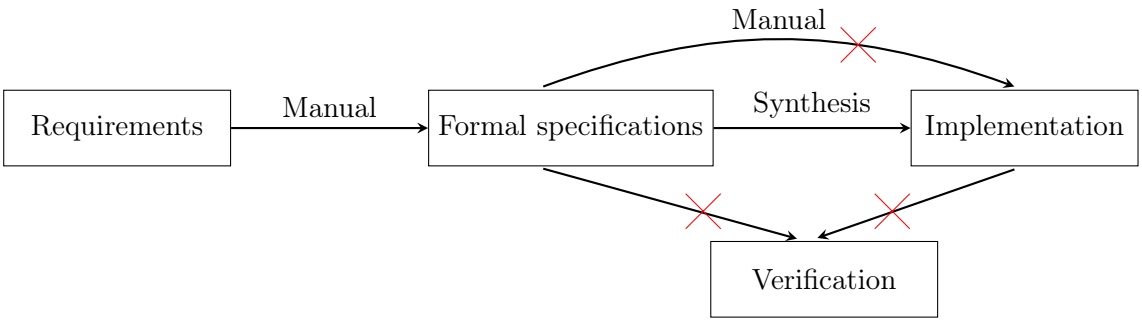


Figure 2.1: Controller synthesis

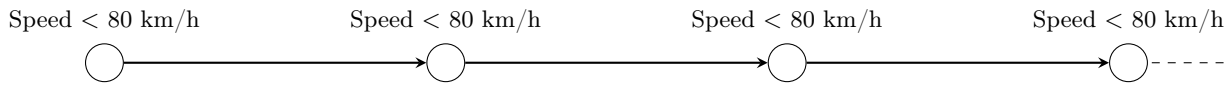
2.2 Linear Temporal Logic (LTL)

Linear Temporal Logic (LTL) is a model for writing formal specification. It is then used for formal verification and checking of embedded systems whose specifications are given as LTL formulae. LTL has logical and temporal operators, which makes it a useful formalism for indicating and confirming properties of reactive systems. Hence, one can translate properties which are written using the natural languages into a formula φ by using the LTL syntax [24]. For a better understanding of the LTL syntax, these logical and temporal operators are listed below:

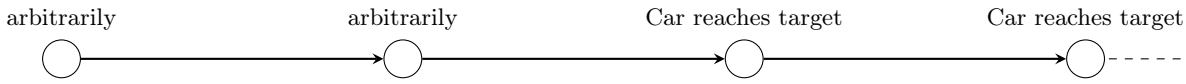
Logical operators:	Temporal operators:
$\wedge$: AND operator	X : NEXT
$\vee$: OR operator	U : UNTIL
$\neg$: NOT operator	G : ALWAYS
$\rightarrow$: IMPLIES operator	F : EVENTUALLY

In order to comprehend the usage of these operators, some examples of self driving requirements specified in LTL are shown below:

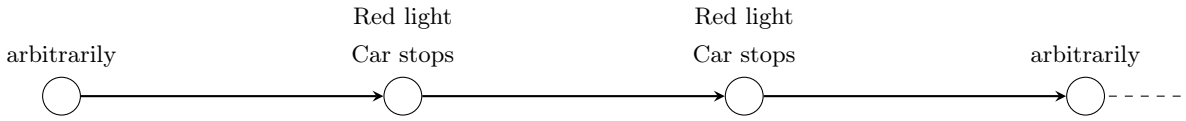
$\varphi = \mathsf{G}$ ($\neg$(Speed $>$ 80 km/h)) (The speed limitation of 80 km/h is never violated)



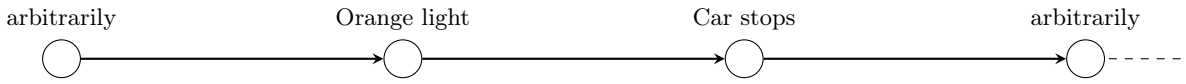
$\varphi = \mathsf{F}\ \mathsf{G}$ (Car reaches target) (Eventually, the car always reaches the target)



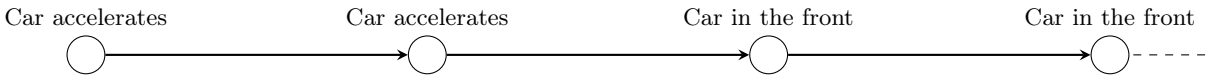
$\varphi = \mathsf{G}$ (Red light $\rightarrow$ Stop) (If "Red light", then th "Car stops" is true)



$\varphi = \mathsf{G}$ (Orange light $\rightarrow$ X (Stop)) (If "Orange light", then the Car will stop)



$\varphi = \mathsf{G}\ \neg$(Front car $\wedge$ Car accelerates) (Never accelerate when "Car in the front")



2.3 Formal synthesis

2.3.1 Overall concept

In order to synthesize a controller, one needs a set of input atomic propositions AP_I, output atomic proposition AP_O, and an LTL formula φ over $AP_I \uplus AP_O$. The purpose is to find a Mealy or a Moore automaton reading AP_I and writing AP_O that satisfies φ i.e. is there a finite state system which ensures that the specifications are not violated?. If such system exists, then the controller is said to be realizable. In order to achieve this, the synthesis is considered as a game with two players. In this game, the inputs are determined by the environment, the output by the system, and the playground is represented by a finite state graph with the possibility of infinite plays. To win this game, the player needs to find a strategy, which plays fulfill φ. Since only one player can make a move at a time, this game is characterized as a turn based game.

Summary:

- Player 1 controls AP_I and player 2 controls AP_O
- Set of states: Q with initial state: q_0
- Transition function: $\delta : \ Q \times \ I \ \times O \ \rightarrow \ Q$
- Player 1 selects i_k, Player 2 selects o_k, $where \ q_{k+1} = \delta(i_k, o_k)$
- Winning condition: objective over $F \ \subseteq \ Q$
- Strategy: $Q \ \times I \ \rightarrow \ O \Rightarrow$ Choose the winning strategy such that all resulting plays fulfill φ
- Reachability game: A certain set of states are eventually reached by each play.
- Safety game: Only states safe states are visited by all plays.
- LTL \Parity game: A certain set of states that are visited infinitely often.
- and more..

2.3.2 GR(1) synthesis

As mentioned in section (2.1), synthesizing the full LTL cannot be considered due the high complexity of the approach. In order to overcome this issue, one approach would be to enforce determinism. This is done by enforcing deterministic Büchi automata for specifying both environment and system and it is called Generalized Reactivity of the Rank 1 or GR(1) [27]. Using this approach, only LTL specifications of the form (2.1) are accepted.

$$(\mathsf{G}\,\mathsf{F}\,\varphi_1 \wedge \ldots \mathsf{G}\,\mathsf{F}\,\varphi_2 \wedge \ldots \mathsf{G}\,\mathsf{F}\,\varphi_n) \rightarrow (\mathsf{G}\,\mathsf{F}\,\psi_1 \wedge \ldots \mathsf{G}\,\mathsf{F}\,\psi_2 \wedge \ldots \mathsf{G}\,\mathsf{F}\,\psi_n) \quad (2.1)$$

The complexity of synthesis is then decreased to a polynomial time symbolic-synthesis problem [23, 3]. For better understanding of the difference between full LTL and GR(1) synthesis, the table shows the divergence between these two approaches

Synthesis approach	Specifications	Obtain the game	Solve game
Full LTL synthesis	LTL Formula	• Get the non-deterministic Büchi Automaton (time complexity: 2^n)[33] • Determinize the Automaton in order to get the game (time complexity: 2^{2^n})[28, 22]	Solve parity game (time complexity: 2^{2^n})[23]
GR(1) synthesis	Specifications of the form (2.1)	Nothing has to be done[23]	Solve GR(1) game (Polynomial time)[23]

3 SCOTS: A Tool for the Synthesis of Symbolic Controllers

3.1 Overall concept

As shown in the previous section, a good approach to avoid the complexity issue is to focus on simpler specifications. Following this methodology, the tool SCOTS is designed to synthesize controllers it accepts only reachability (reach spec) and invariance specifications (reachAvoid spec) [27].

$$\mathsf{F}\ \text{(Reach target)} \tag{3.1}$$

$$\mathsf{F}\text{(Reach target)}\ \wedge\ \mathsf{G}\ \text{(Avoid obstacles)} \tag{3.2}$$

Despite the limitation in terms of LTL specifications, this tool supports the computation of abstractions of nonlinear control systems. SCOTS considers dynamical systems of the form:

$$\dot{\xi}(t)\ \in\ f(\xi(t), u)\ +\ [\![-w, w]\!] \tag{3.3}$$

- Where $f(., u)$ is continuously differentiable for every $u \in U$
- The set U is non-empty and $U \subseteq\ \mathbb{R}$
- The vector $w = [w_1 \dots w_n]\ \in \mathbb{R}^n_+$ is a perturbation bound and $[\![-w, w]\!]$ denotes the hyper-interval $[-w_1, w_1]\ \times \cdots \times [-w_n, w_n]$

Moreover, the solution ξ and the simple system $S_1 := (X_1, U_1, F_1)$, which is constructed for implementing the desired behavior of the closed loop with respect to a sampling time τ, are defined [26].
Furthermore, the transition function F_1 is defined as:

$$F_1(x, u) := \{x' | \xi(0) = x \wedge \xi(\tau) = x'\} \tag{3.4}$$

with respect to the state alphabet $X_1 := \mathbb{R}^n$, state input $U_1 := U$, and sampling time τ

In order to synthesize the controller, the following need to be specified:

- The Ordinary Differential Equation (ODE) that describes the vehicle's dynamics.

$$\dot{x}(t) = f(x(t), u, t) \tag{3.5}$$

- The growth bound

$$r(t) = g(r(t), u, t) \tag{3.6}$$

- An ODE solver with n number of intermediate steps and with sampling time τ

SCOTS constructs the symbolic model S_2 of $S_1 = (X_1, U_1, F_1)$, which is an over-approximation of attainable sets. The growth bound function 3.6 specifies the extent to which initial state-sets of S_1 diverges form the exact solution φ i.e. $r(0) = 0 \rightarrow \xi(0) = \varphi$.

- Specify the input space: input space quantization η (node distance) as well as the lower and upper bound (lb and ub).

$$u \in [lb_1, ub_1] \times \cdots \times [lb_n, ub_n]$$
$$\text{Quantization: } [\eta_1, \ldots, \eta_n]$$

- Specify the state space: state space quantization η (node distance) as well as the lower and upper bound (lb and ub).

$$x \in [lb_1, ub_1] \times \cdots \times [lb_n, ub_n]$$
$$\text{Quantization: } [\eta_1, \ldots, \eta_n]$$

- Create a symbolic set of post variables by the symbolic set of the state space i.e. posterior state space
- Instantiate the symbolic model with the input and state space as well as the posterior state space
- Compute the transition relation F_1 of the symbolic model with respect to the ODE and growth bound
- Setup a fixed point from the symbolic model after computing the transition relation F_1
- For specifying obstacles, SCOTS provides two different options to model them:
  - Polytopes: $\{ y \in \mathbb{R}^n \mid Hx \leq h\}$ parameterized by $H \in \mathbb{R}^{q \times n}$ and $h \in \mathbb{R}^q$
  - Ellipsoids: $\{ y \in \mathbb{R}^n \mid |L(x-y)|_2 \leq 1\}$ parameterized by $H \in \mathbb{R}^{n \times n}$ and $y \in \mathbb{R}^n$

- For specifying the target, polytopes and ellipsoids are also applied.
- In the case of reach (3.1) specification, then only the target space need to be specified. In case of reachAvoid (3.2), then not only the target but also the obstacle space need to specified.

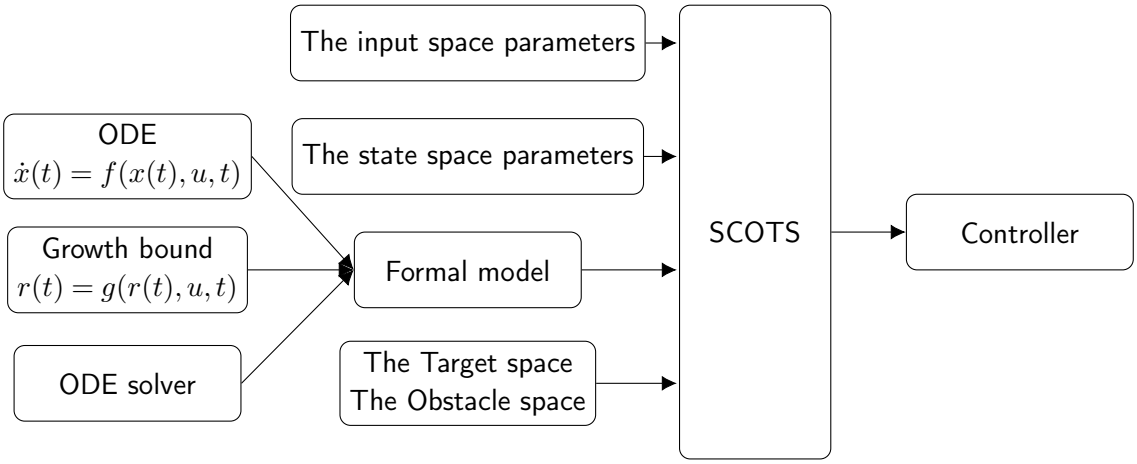


Figure 3.1: Simplified illustration of SCOTS' work process [27]

After specifying all of the above, a controller can be synthesized in the form of a symbolic set. The synthesized controller takes the vehicle's state x as an input and a set of inputs U_1 as outputs. For each $u_1 \in U_1$ and $x_1(t) \in X_1$, we have $\text{safe}(f(x_1(t), u_1, t)) = \text{True}$. The size of U_1 depends not only on the realizability of the problem but also on the growth bound.

3.2 Matlab simulation

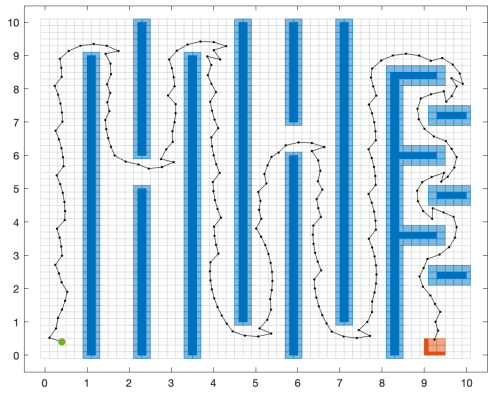


Figure 3.2: SCOTS vehicle example [27]

For a better understanding of the tool's output, a simple example is presented. Figure 3.2 shows a green point at the left side. This is the starting point of the vehicle. The blue objects represents the obstacles to be avoided. As for the orange area, it represents the target that the vehicle needs to reach. Finally, the line represents one safe path for the vehicle to reach the target. However, the ego vehicle is always modeled as a point. Therefore, if the moving object is, for example, a vehicle, then the vehicle's width needs to be taken into account. Unfortunately, SCOTS does not provide the possibility to specify the moving object dimensions. In this case, two workarounds can be considered. The first is to make all obstacles bigger such as collisions can be avoided. The second one is to choose points from the synthesized controller that hold a safety distance between the moving object and all obstacles.

4 Tool integration in the highway traffic simulation

4.1 Udacity traffic simulation

Figure 4.1: Udacity high way simulation [5]

The high way traffic simulation used in this thesis is developed by Udacity as a part of a paid program called nanodegree [5]. The propose of this course is to teach students how to train cars to drive autonomously. The simulation has two modes: the manual and the autonomous one. If the autonomous mode is chosen, then the simulation sends the sensor fusion outputs as well as the ego vehicle location and speed to the control code in a *json* format. The initial project, in which the control the car should be implemented, is provided in [31]. In order to enable the communication between the simulation and the control code, the websocket protocol is used. The websocket protocol enables the communication between a client (the control code in our case), and a remote host (the simulation) with continuous information exchange [12]. The illustration of the websocket protocol usage in this thesis is shown in figure 4.2.

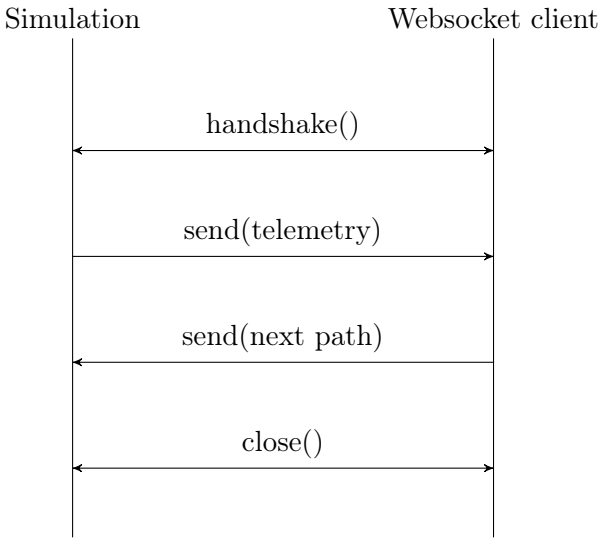


Figure 4.2: Simulation-control code communication [12]

Once the host gets the telemetry data, these outputs need to be processed by two modules: the behaviour planner and the path planner. Depending on the ego vehicle situation in the highway traffic, the behaviour planner decides if the car would stay in the same lane, accelerate or decelerate, turn right or left. The behaviour of the car is not only driven by safety requirements but also by the user preferences. Hence, one conservative approach would be to limit the task to the safety requirements but this is not the case in this thesis. Indeed, the purpose is to reach the target as fast as possible while upholding traffic rules. The state diagram 4.3 illustrates the behavior planner used in thesis. It shows that the car never change lanes unless there is a car blocking in the front. A car is considered as blocking the front if the distance between the ego vehicle and the front car is less than $\Delta s = 17$ i.e. $s_{\text{ego}} + 17 > s_{\text{front car}}$. If this is true, then the car should change lanes and the priority is to turn left (see figure 4.8). If there is a car on the left or there is no left lane, the ego vehicle should turn right (see figure 4.9). If this also cannot be achieved, the ego vehicle should stay in the same lane until there is no blocking obstacles. One should note that the car change lanes once at a time.

Once the next action is known, the path planner, which is illustrated in the state diagram 6.3, determines the next path of the ego vehicle. The target is always specified as $s_{\text{target}} = s_{\text{ego}} + 40$, and as for the d axis, it depends on the chosen behavior. If the ego vehicle turn left, then $d_{\text{target}} = d_{\text{ego}} - 4$, and if the car turn right, then $d_{\text{target}} = d_{\text{ego}} + 4$. For a better understanding, the figure 4.7 illustrates the lanes coordinates. Once the target is determined, both target and obstacle spaces are specified in scots, which synthesizes the path from the ego vehicle to the selected target using the specification reachAvoid.

Now that we have the safe next path in Frenét coordinates, we need to convert it to Cartesian coordinates. However, this is not sufficient to run the car safely. Hence, the spline interpolation is used to assure the continuity of the trajectory by interpolating points between the starting point and the target in x-y coordinates [10]. This would specify the speed as well as the motion smoothness of the ego vehicle.

If this is accomplished, then the simulation gets the next path of the ego vehicle, which is a set of x-y points, from the websocket client. In this simulation, the speed, location, and previous path of the ego are provided. Moreover, sensor fusion outputs provides also the location of the nearby cars and their velocities.

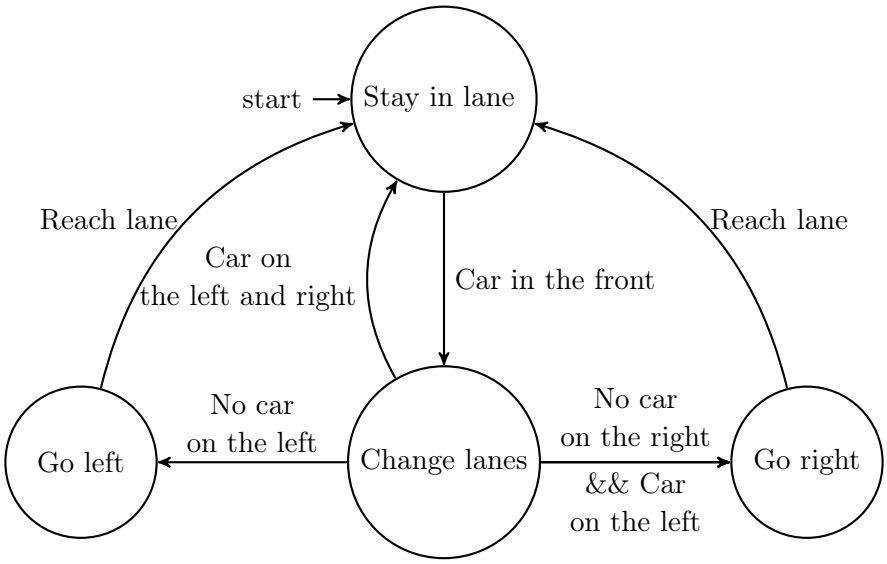


Figure 4.3: Behavior planner

These information are updated in each iteration according to the changing environment. Since the map of the landscape is also provided, then the next safe location of the car can be predicted [5]. The safety requirements in this simulation are the following:

- No collision with other cars
- No speed limit violation ($50mph/80kmh$)
- No acceleration limit violation ($10m/s^2$)
- No maximum jerk violation ($50m/s^3$)
- Stay in the middle of the lane.

The jerk refers to the acceleration change, which has a big impact on the driver and passengers comfort. Therefore, in order to keep a certain comfort, the maximum jerk of $50m/s^3$ should not be exceed.

All these information need to be processed in order to extrapolate a design that would satisfy all of the requirements above. Using scots for path planning, all the requirements were considered. However, the spline interpolation [10], which was taken from [30], contributes to the non-violation of the max jerk and speed limit requirements. Hence, the path and the target are specified by SCOTS but the smoothness of the car's motion as well as the speed limitation and the maximum jerk limitation is determined by the spline interpolation.

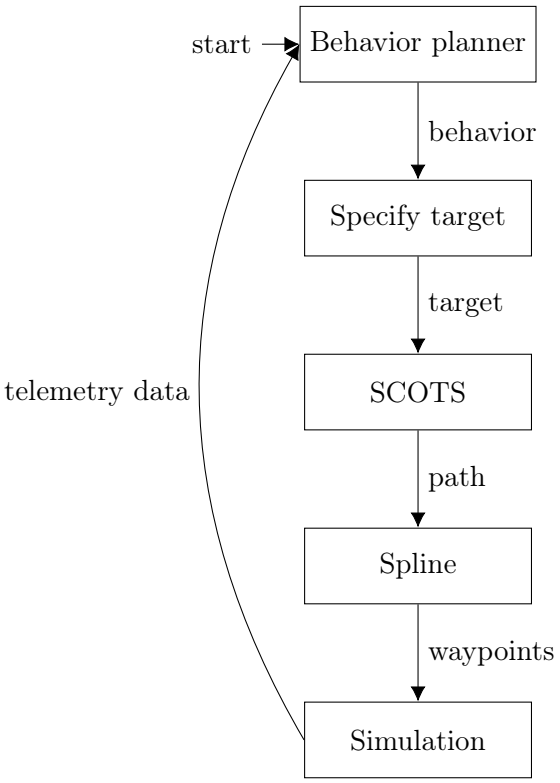


Figure 4.4: Overall process

4.2 Frenét coordinates

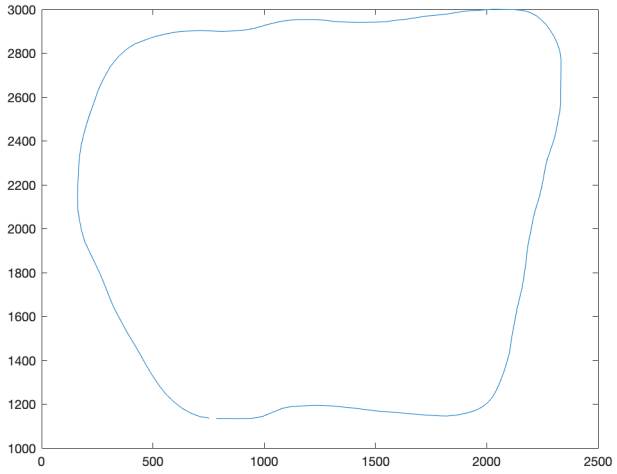


Figure 4.5: Full map

The sensor fusion provides the location of the ego vehicle in two different coordinates: x-y (Cartesian coordinates) and d-s (Frenét coordinates). The figure 4.5 shows the map of the highway road in x-y coordinates. Looking a the shape of the road, one can conclude that locating vehicles relatively to each other using this map is an inconvenient and a complicated task. In order to overcome this issue, the map is transformed to a Frenét coordinates. The transformation is shown in figure 4.6.

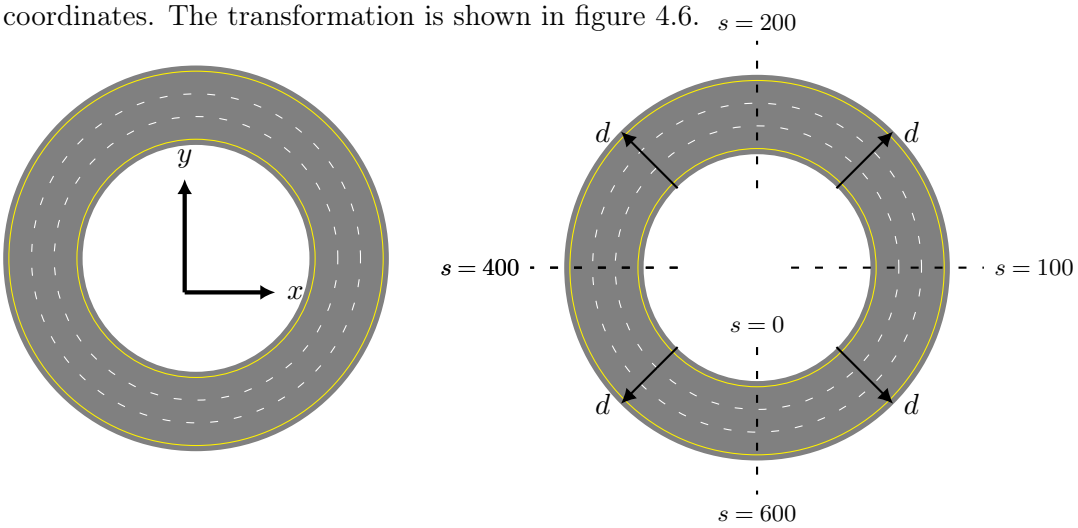


Figure 4.6: Cartesian vs Frenét coordinates [32]

The advantage of Frenét coordinates is that the axis takes the shape of the road i.e. in the of middle the road we have $d = 0$ and each lane l is defined by an interval $d \in$ [lane_start, lane_end]. And as for the s axis it represents the length of the road. As a result of using these coordinates, one can easily detect the presence of a vehicle at a certain lane, which makes events such as "There is a car on the left" easier to define. Moreover, it makes the use of SCOTS for this task possible. Indeed, the use of the Frenét coordinates makes it possible to use a predefined state space. Hence, the use of a small shifting window is used instead of specifying the whole map as the state space. The figure 4.7 illustrates this shifting window in red dashed square. The use of this workaround allows scots to synthesize a controller much faster and subsequently send the way-points in time such as the ego vehicle does not stop or crash. The empirical evidence from this thesis shows that sending new points to the simulation each $0,03$ second without incidents is feasible.

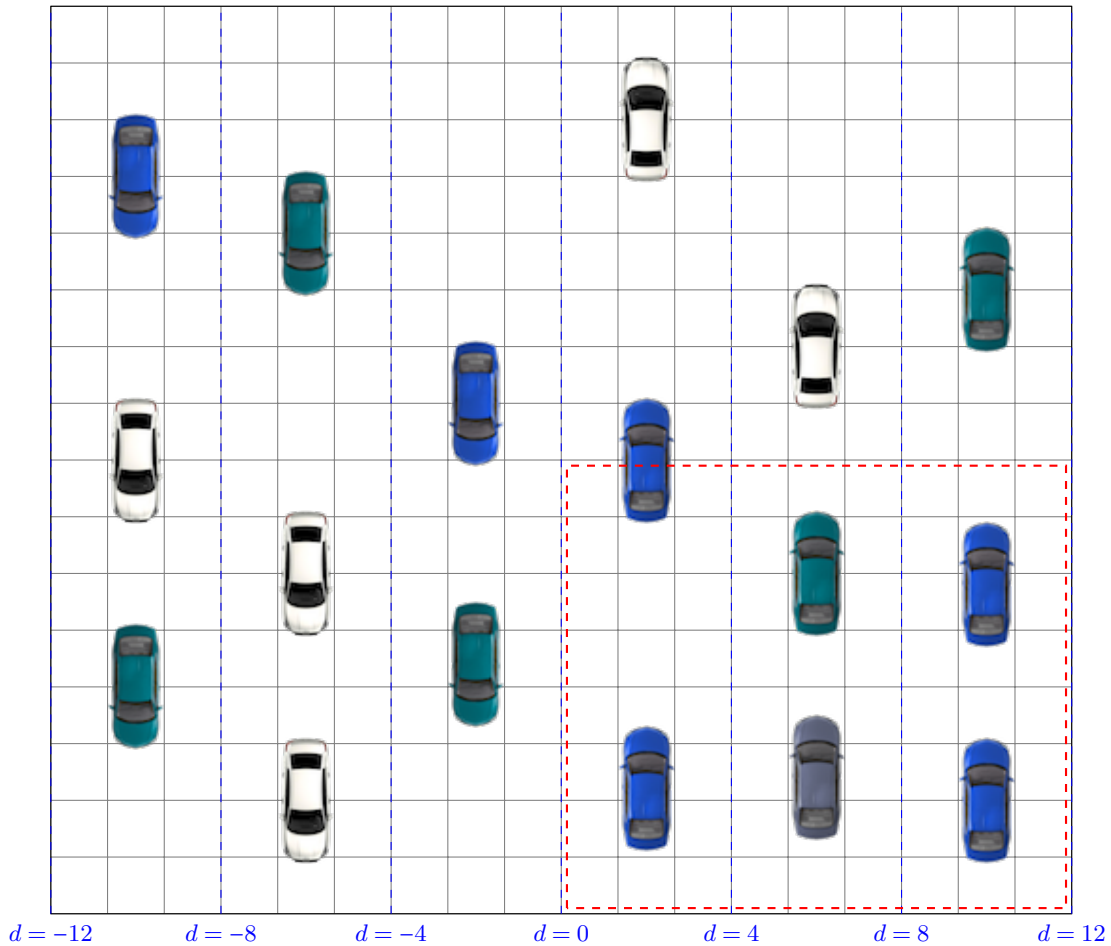


Figure 4.7: Representation of the simulation's highway traffic

4.3 Use cases

Turn left situation:

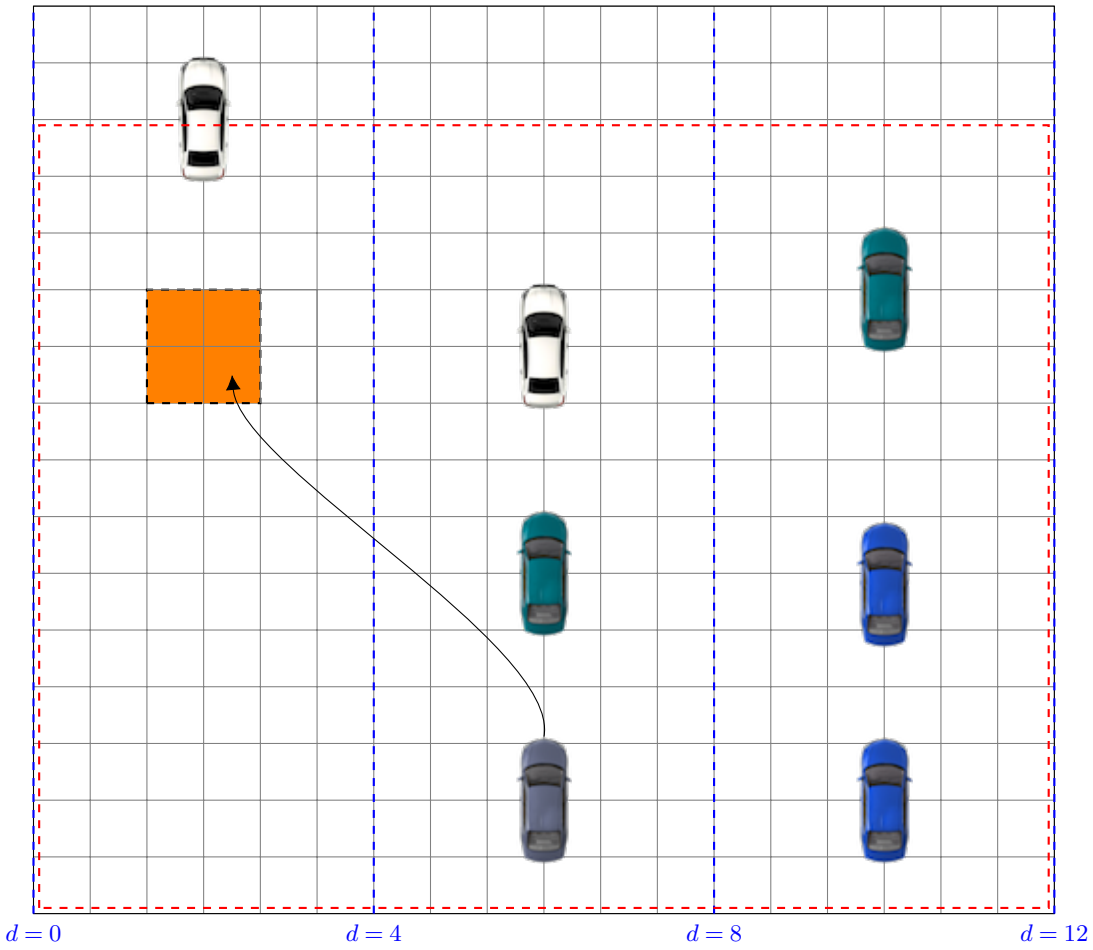


Figure 4.8: Turn left

The figure above shows the case when the ego vehicle (represented by the grey car) turn left because there is another car (represented by the green car in front of the ego vehicle) blocking the front road and there is no other car blocking the left side. The grey grid represents the quantization of the state space and the orange square represents the target space. Scots tool synthesizes the safe path from the ego vehicle to one point that is included in both target space and controller domain.

Turn right situation:

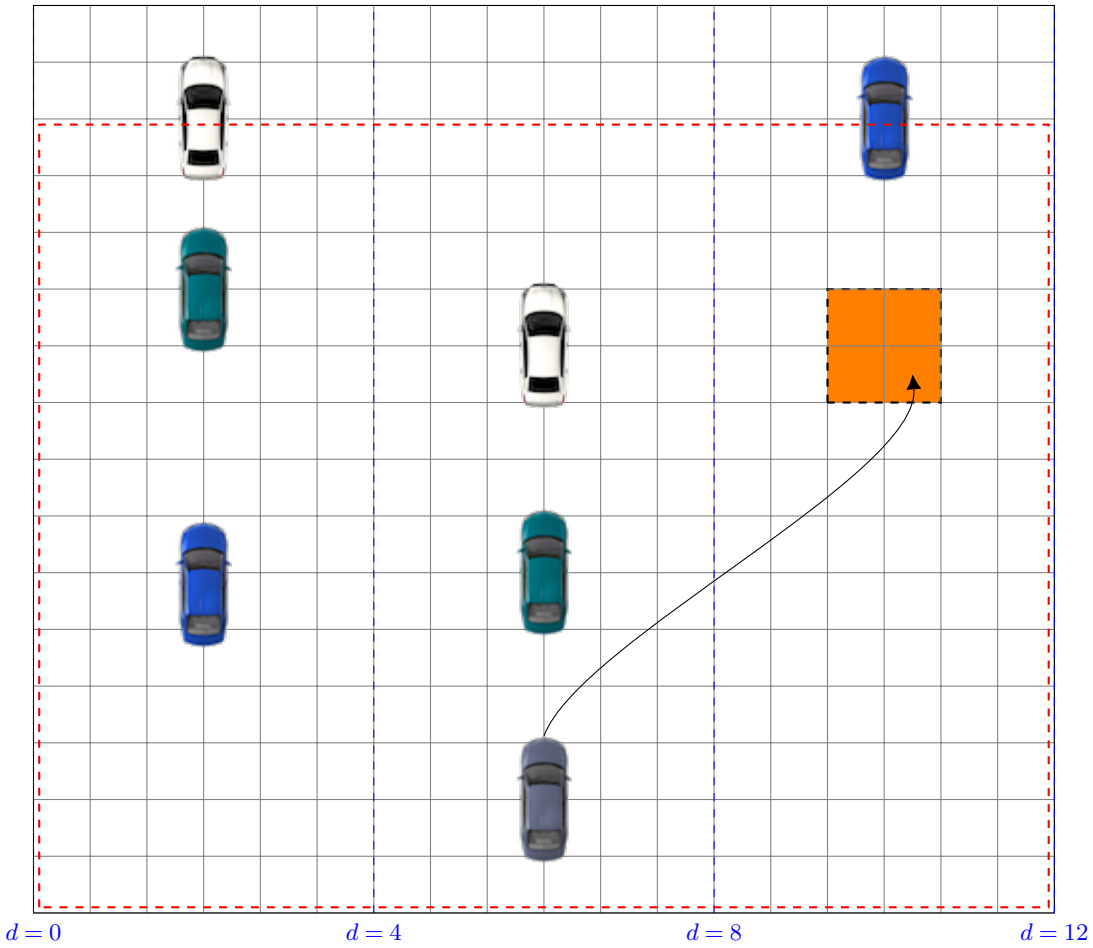


Figure 4.9: Turn right

The figure above shows the case when the ego vehicle (represented by the grey car) turn right because there is another car (represented by the green car in front of the ego vehicle) blocking the front road, a car blocking the left side, and no car is blocking the right side.

Stay in lane situation:

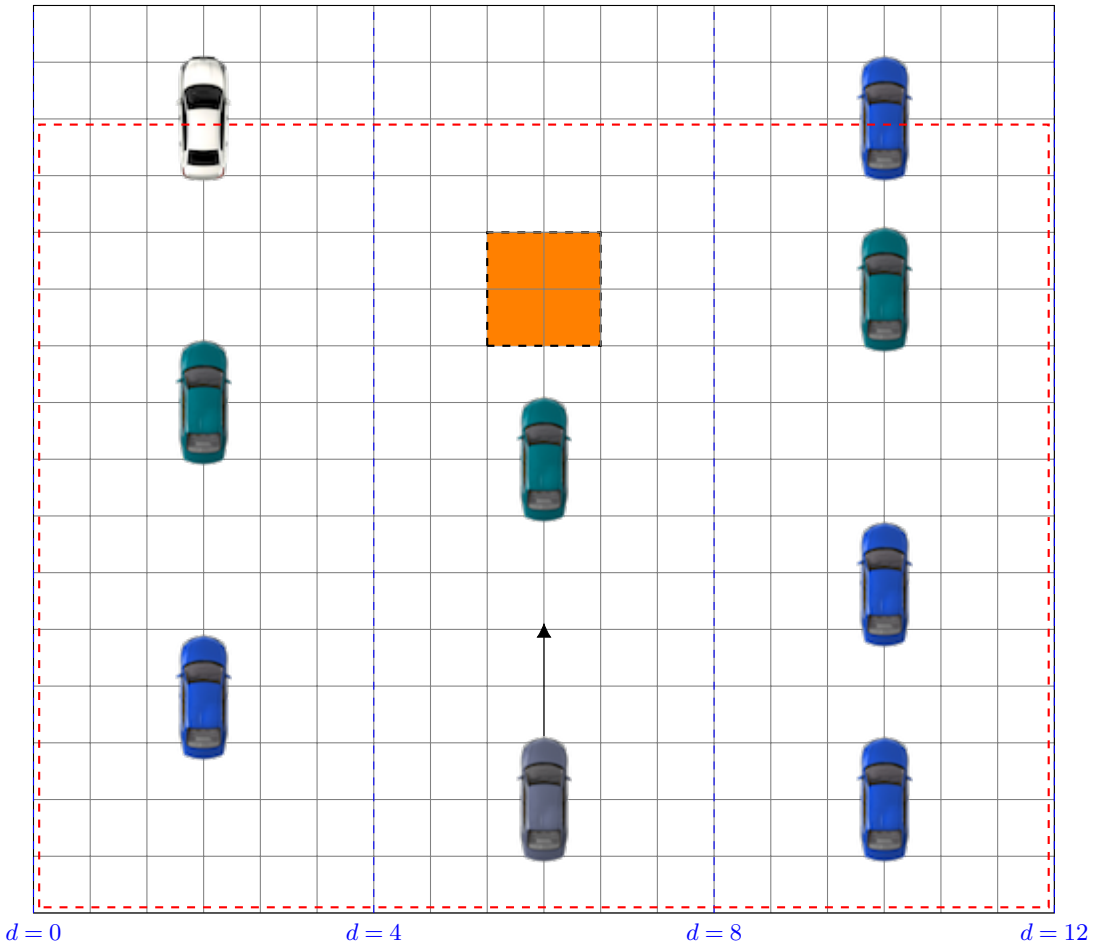


Figure 4.10: Stay in lane

The figure above shows the case when the ego vehicle (represented by the grey car) stays in the same lane while another car (represented by the green car in front of the ego vehicle) is blocking the front road, and the right and left side is blocked.

4.4 Symbolic sets parameters

ODE:

$\dot{x_1} = u_1\ cos(u_2)$

$\dot{x_2} = u_1\ sin(u_2)$

u_1 : velocity ; u_2 : the heading

x_1 : s position ; x_2 : d position

Growth bound:

$r_1 = 0.5$

$r_2 = 0.5$

State space:

lower bound $= [210, 0]$

upper bound $= [316, 12]$

$\eta = [5, 1]$

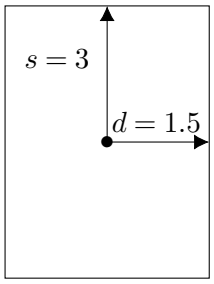
Input space:

lower bound $= [0, -3.14]$

upper bound $= [50, 3.14]$

$\eta = [0.1, 0.4]$

Obstacle space:



The figure below shows a visualization of the moving window, which was introduced in the prior section. This was taken from a simulation run at a random time. In this window, the ego vehicle (shown in yellow and not specified in scots) has 3 other vehicles in the back and one blocking the front road (the cars are shown in blue). In this situation, the ego vehicle has to turn left, therefore the target space, is on the left (shown in orange). One should notice that the d-axis in reversed in this figure and that the ego vehicle is moving from the left to the right.

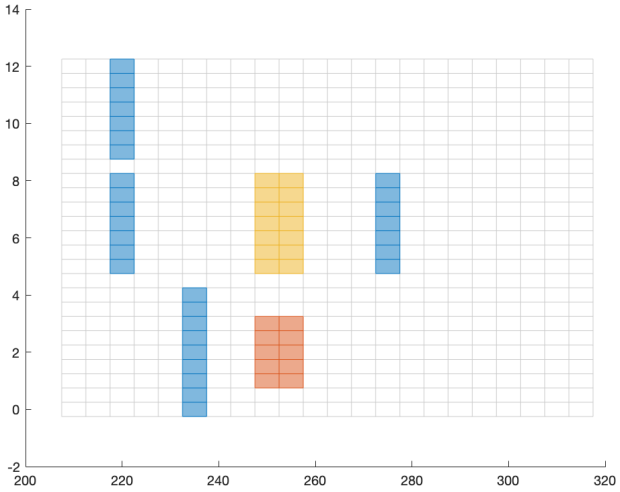


Figure 4.11: Matlab visualization

5 Artificial Intelligence (AI) for path planning

The use of AI-based systems in autonomous driving was encouraged by the fact that great findings has been made in the field of object detection and pattern recognition. Hence, autonomous vehicles have been deployed on the road that uses deep learning to detect lanes and other vehicles as well as to control the steering wheel and the vehicle's speed [16, 4].

However, the black box nature of these systems makes the certification of safety critical application that uses these system not feasible. Hence, to this day, there is no technique that would formally verify these system i.e. no safety guarantee can be obtained. Even worse, if the AI-design fails and the vehicle crashes as a result, there is also no proven method to explain the failure. Moreover, even if the issue can be explained, no guarantee can be provided regarding the recurrence of the failure [19, 7, 18].

In order to make a contribution towards solving these issues, a design is implemented, where the neural network outputs are checked for safety requirements. This is called shielding, which is useful when a faulty design is used in a safety critical application. Hence, the safety shield is the component that enforces safety specifications by modifying every output that is generated by a faulty design and considered as unsafe. For a better understanding, the figure shows how the shielding technique occurs.

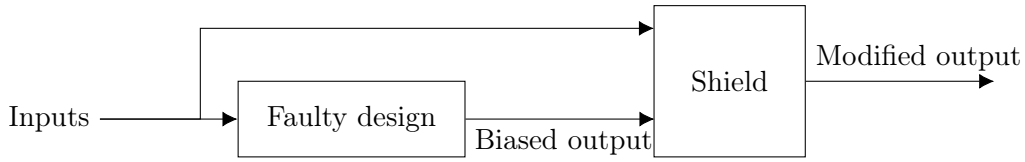


Figure 5.1: Safety Shield

5.1 LSTM cell

First when neural networks were introduced, they did not have the capability to memorize the inputs. For instance, classifying textual information was not feasible since each input word has a dependency with the prior one. In order to solve this issue and make the neural networks memorize sequences, the Long Short-Term Memory (LSTM) cell was first introduced in 1997 and then modified in 2000. The figure below shows the structure of LSTM cell [13].

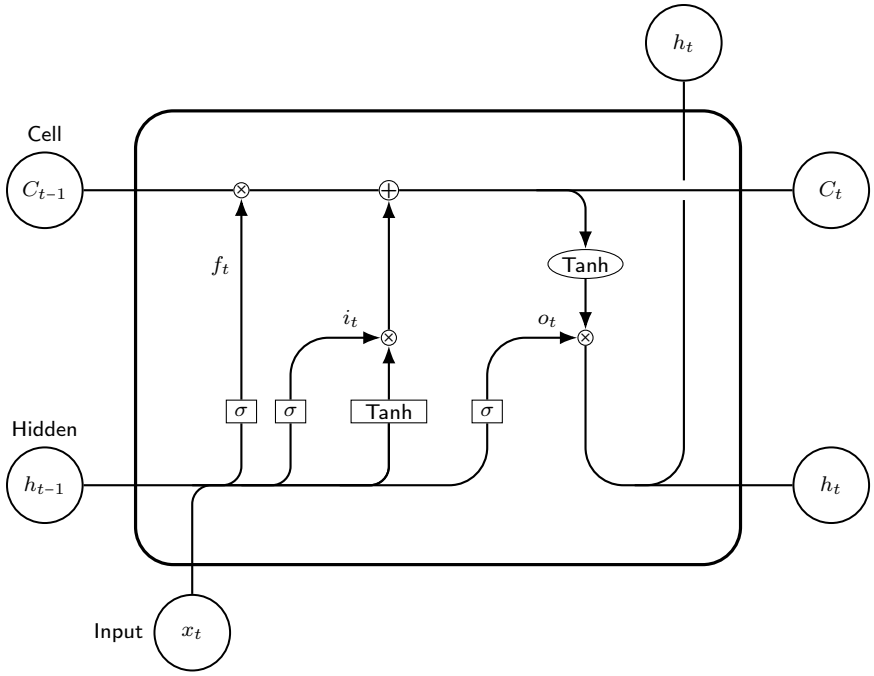


Figure 5.2: LSTM cell [13]

$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$
$i_t = \sigma(W_f[h_{t-1}, x_t] + b_i)$
$\tilde{C}_t = \tanh(W_f[h_{t-1}, x_t] + b_C)$
$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$
$o_t = \sigma(W_f[h_{t-1}, x_t] + b_o)$
$h_t = o_t \times \tanh(C_t)$

$x_t \in \mathbb{R}$: input vector
$f_t \in \mathbb{R}^h$: forget gate's activation vector
$i_t \in \mathbb{R}^h$: input gate's activation vector
$h_t \in \mathbb{R}^h$: hidden state vector
$c_t \in \mathbb{R}^h$: cell state vector
$W_t \in \mathbb{R}^{h \times d}, U_t \in \mathbb{R}^{h \times h}$, and b $\in \mathbb{R}^h$: weight matrices and bias vector

5.2 Recurrent neural networks

There are many varieties when it comes to Recurrent Neural Networks (RNNs). One variety is composed of LSTM layers since any function involving recurrence can be considered a RNN. This type of RNN has better memorization capacity than other types [13]. In this thesis, two RNNs with an identical architecture were trained to predict the next safe path of the ego vehicle in the Frenét coordinates (s and d positions). However, it should be noticed that the s and d refer to the ego vehicle position inside the predefined window illustrated as a dashed red square in the figure 4.7.

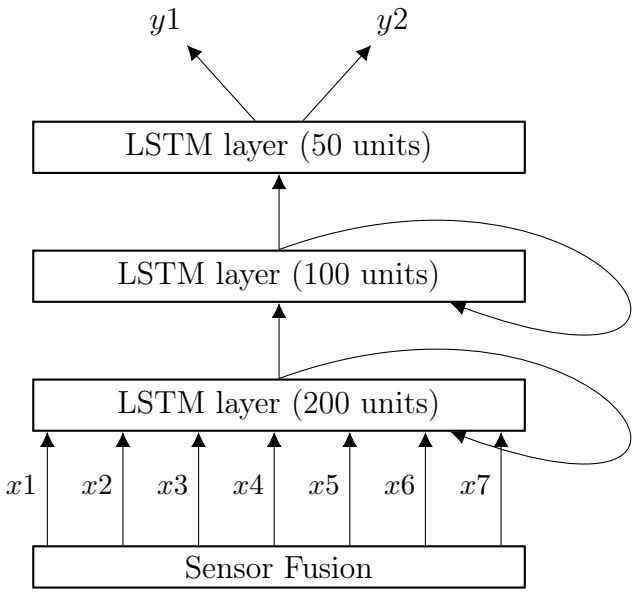


Figure 5.3: RNN architecture

- $x1/x2/x3/x4$: Front/Back/Left/Right Car (0 or 1)
- $x5$: Car speed (0 to 50mph)
- $x6/x7$: Current car s/d position
- $y1$: Δs position
- $y2$: Next d position

5.2.1 Data collection

The data used to train the two RNNs was collected from a simulation run that was previously implemented with SCOTS.The data set is divided into training and test data. Moreover, the log data was collected in a csv format. The inputs of the two RNNs are the following: 4 binary inputs x_1, x_2, x_3, and x_4 that signals the presence of other cars in the front, back, left, and right side as well as the speed x_5 and the position of the ego vehicle x_6. As for the outputs, y_1 and y_2 refers to the Δs position and d position. Hence, the RNN that is meant to predict the s position, is trained to the predict $\Delta s = s_{new} - s_{old}$ instead. This modification provided a better accuracy for the training procedure.

5.2.2 Data visualization

In this section, a data visualization of the data collected is provided. The visualization provides the quality of the data collected. Hence, if the data has low variance, the probability that the trained architecture would suffer from either over-fitting or under-fitting. In other words, if an event appear must often enough during the simulation run, it is likely that other events will be false predicted. This is referred to as over-fitting. On the other hand, if an event does not appear often during the simulation run, it is likely that a false prediction would occur. This is referred to as over-fitting [14].

Therefore, before to begin with the training, the collected data should be visualized. If the quality of the data is not good enough, another simulation run and data collection procedure should be executed.

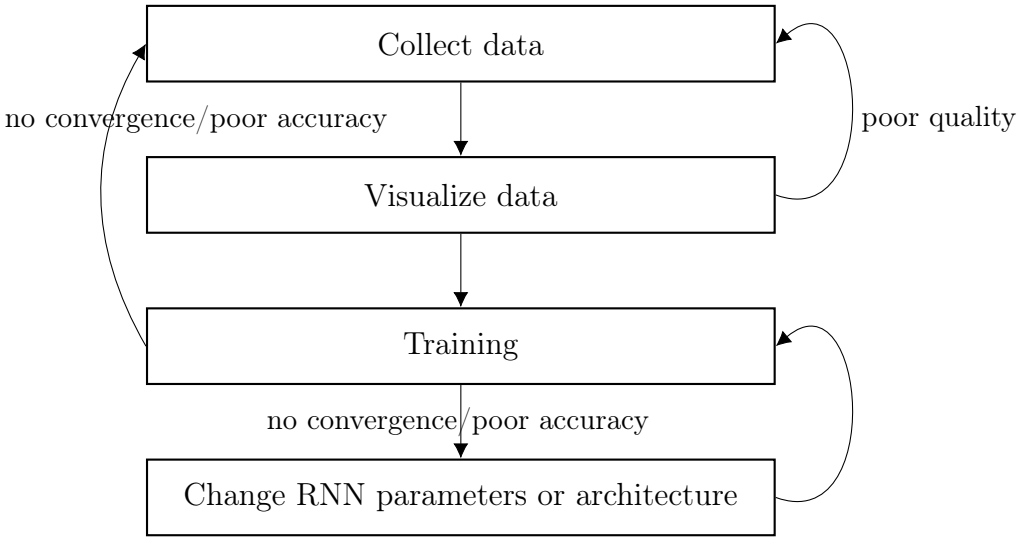


Figure 5.4: Methodology

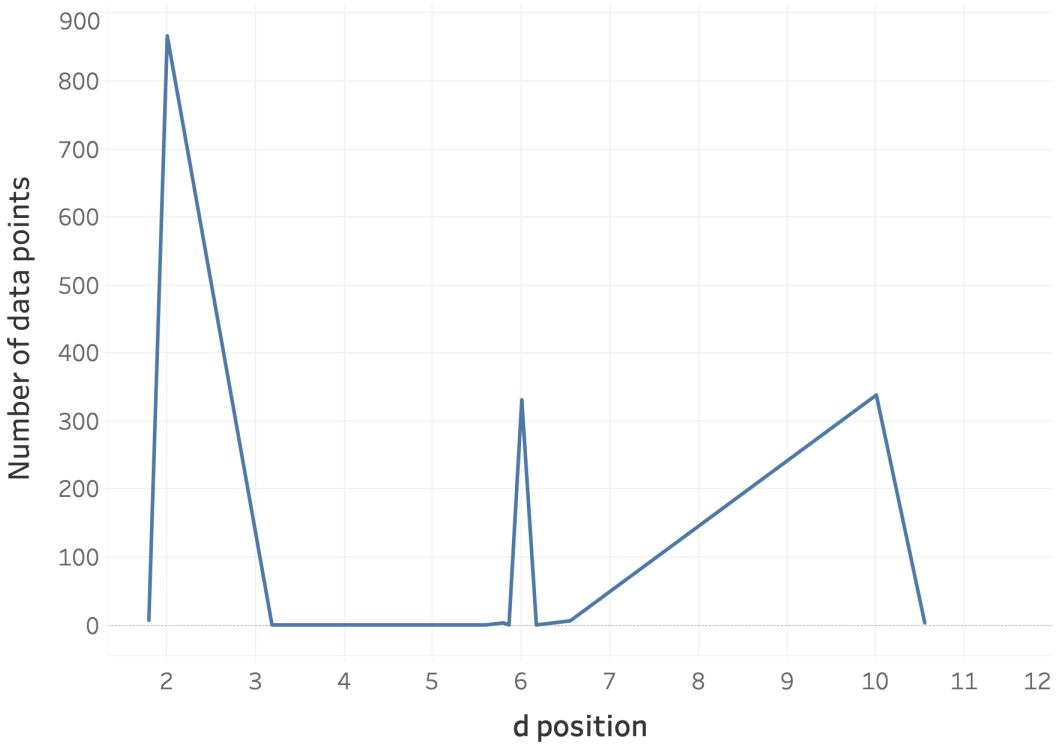


Figure 5.5: Data points of the position d

The figure 5.5 shows the lane position of the ego vehicle during the simulation run that was used to train the RNN. It shows that the ego vehicle has been more on the left side on road than other lanes. This can be explained by the fact that when a car is blocking, the priority is always to turn left as long as there is obstacles.

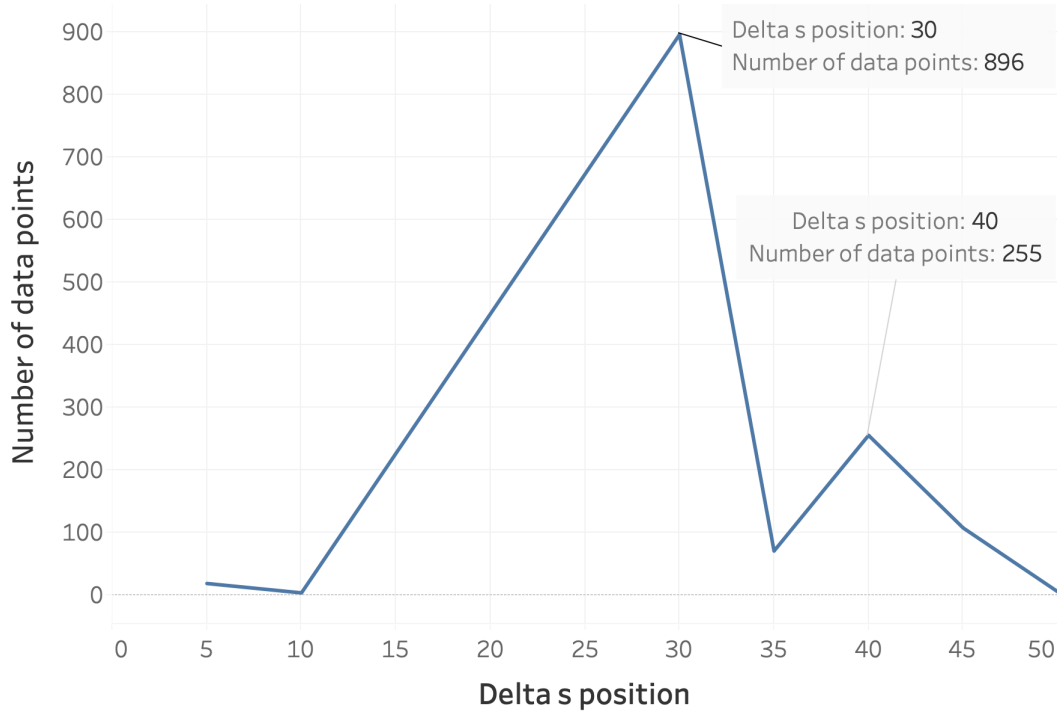


Figure 5.6: Data points of Δs position

The figure 5.6 shows the Δs position ($\Delta s = s_{new} - s_{old}$) of the ego vehicle during the simulation run that was used to train the RNN. It shows that Δs position has low variance, which would lead to a relatively poor accuracy (74% accuracy against 92% for the d position) after training the RNN with this data. This can be explained by the fact that when a car runs with a constant speed (speed limitation of $50mph/80kmh$) unless a car is blocking the front or taking a turn.

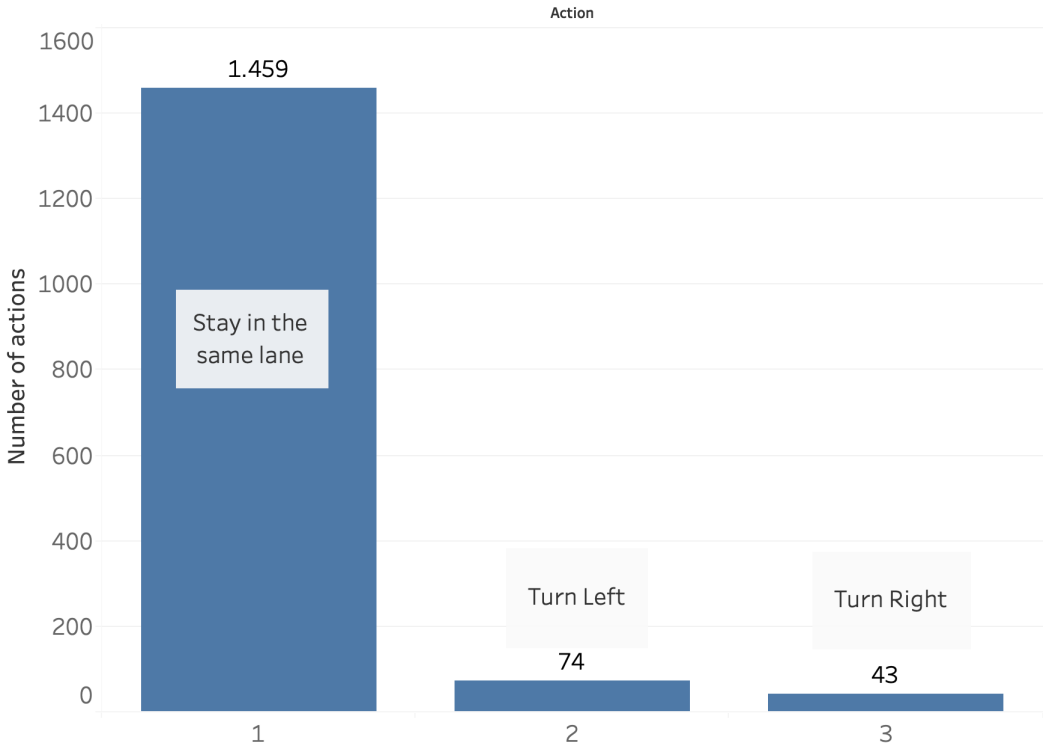


Figure 5.7: Actions taken during the simulation run

The figure 5.7 shows the actions taken by the ego vehicle during the simulation run that was used to train the RNN. The actions are labeled as follows: 1→stay in the same lane, 2→turn left, and 3→turn right. The data shows that, most of the time, the ego vehicle stayed in the same lane, and that it turned left more that it did to the right.

5.2.3 Training

After training both neural networks, the test data shows that predicting the lane position d has 92% accuracy and 74% for the s position. The difference between both accuracy's can be explained by the fact that the data lane position d has more variance than the position s. Although the same architecture was adopted for both networks, two different data sets were collected. This was crucial for the convergence of both training. Moreover, only 1 data point as batch size and number of epochs equal to 20. Unfortunately, no formal explanation can be provided regarding these decisions. The RNNs have LSTM layers that have the dimensions of 200, 100, and 50 in number of units respectively. The number of units in this context refers to the dimension of the inner cells C_t and C_{t-1}, and hidden/output state h_t and h_{t-1}, which are shown in figure 5.2. Moreover, it should be noticed that the ADAM optimizer was used for the training [17].

```
Test score: 0.2981689135233561
Test accuracy: 0.927891156462585

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 1, 200)            166400
_________________________________________________________________
lstm_2 (LSTM)                (None, 1, 100)            120400
_________________________________________________________________
lstm_3 (LSTM)                (None, 50)                30200
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 51
=================================================================
Total params: 317,051
Trainable params: 317,051
Non-trainable params: 0
```

Figure 5.8: Summary of the RNN model (d position)

```
Test score: 74.31016685407789
Test accuracy: 0.7416879795396419

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 1, 200)            166400
_________________________________________________________________
lstm_2 (LSTM)                (None, 1, 100)            120400
_________________________________________________________________
lstm_3 (LSTM)                (None, 50)                30200
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 51
=================================================================
Total params: 317,051
Trainable params: 317,051
Non-trainable params: 0
```

Figure 5.9: Summary of the RNN model (Δs)

5.2.4 Tools and frameworks

In order to train the RNNs, the Keras framework was used in python. This choice is motivated by the fact that the deep learning community use the programming language python to train neural networks. And as for the framework, Keras provides a high level of abstraction in order to define the neural network architecture as well as for the training [9].

Figure 5.10: frugally-deep [15]

Once the training is done, the model should be transferred from the python code to the c++ code to run on the simulation. In order to achieve this, the header-only library frugally-deep is used to run the forward pass on the model. First, the model is saved in h5 format (python code) using model.save('....h5', include_optimizer=False) and then converted to json file using frugally-deep [15].
Consider the following command to convert the model:

```
# python3 keras_export/convert_model.py keras_model.h5 fdeep_model.json
```

Finally, frugally-deep allows the loading of the model using fdeep::load_model(...) and model.predict(...) to run the forward pass on the model.

6 Methodology and results

6.1 Work process

This section highlights the work process of this thesis. This is shown in the flow chart below. The arrows that goes backwards indicates that the process was not done successfully since the prior process needed to be rectified many times such as the next process can be implemented.

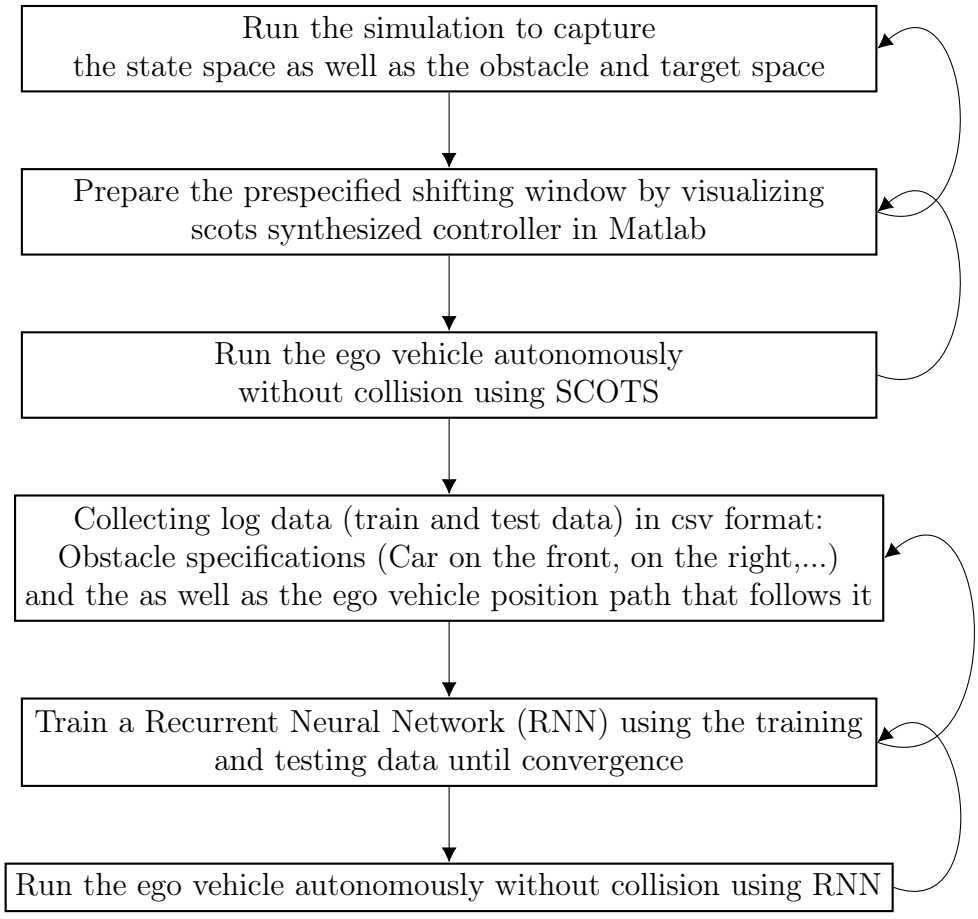


Figure 6.1: Methodology

6.2 Final model

The final implemented model is composed of the formal path planner, the AI design, and the shield that decides which path should be adopted. Hence, the shield takes the AI design output and compare it with the formal path planner output. If the AI output is considered as unsafe, then the path planner's output is taken instead. In this thesis, the safety of the AI output is measured by $\xi_s = s_{predicted} - s_{safe}$ and $\xi_d = d_{predicted} - d_{safe}$, where the $safe$ output is the formal path planner output and the AI output is the *predicted* output. The figure 6.2 illustrates the full model that was modeled in this thesis.

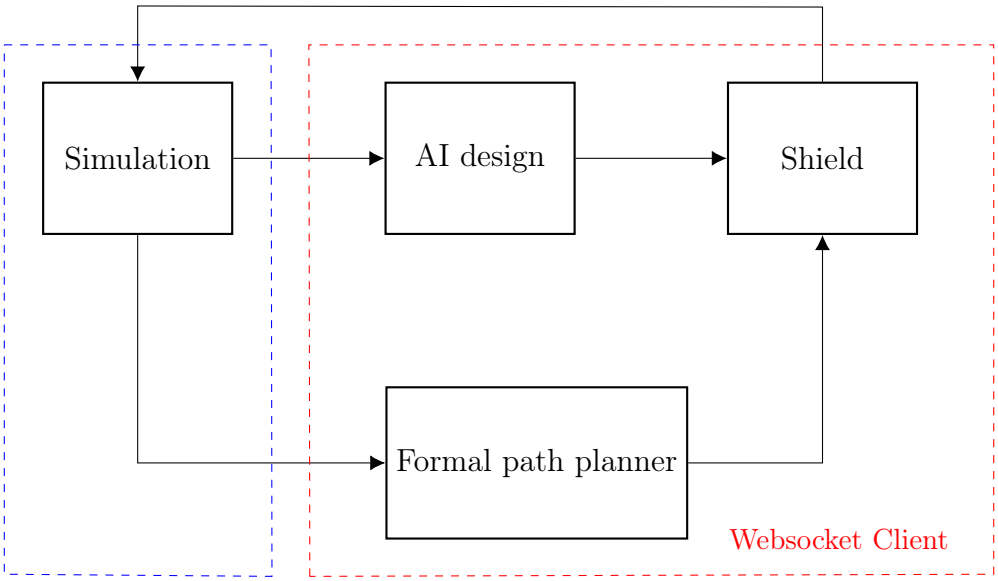


Figure 6.2: Final model

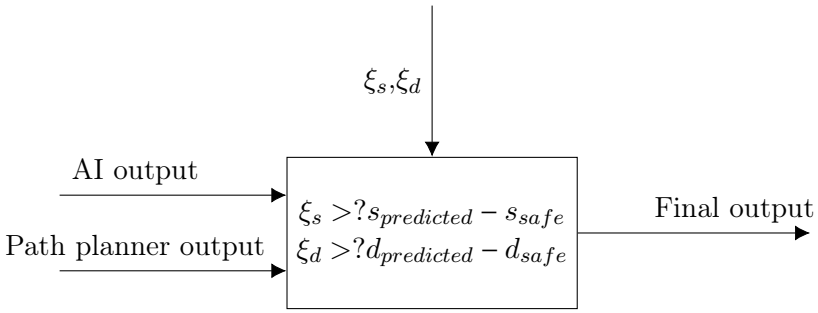


Figure 6.3: Overall process

6.3 Simulation runs

In this section, the simulation results, using the final model shown in figure 6.2, are presented. Hence, 3 different shields configurations have been implemented. For each configuration, 5 different runs have been performed. The results shows that a conservative shield configuration lead to safer paths but more shield usage and less AI usage. On the other hand, a more fault-tolerant configuration lead to less shield usage, more AI usage but more safety requirements violations.

First configuration (conservative)

Shield tolerance: $\xi_s = s_{predicted} - s_{safe} = 20$ and $\xi_d = d_{predicted} - d_{safe} = 1$

Run	Path planner outputs	AI outputs	Shield usage	AI usage
#1	1682	699	58%	42%
#2	863	200	77%	23%
#3	790	425	47%	53%
#4	1056	473	55%	45%
#5	996	517	48%	52%

Observations:

- No collision
- No violation of the speed limitation
- No violation of the maximum acceleration
- Few violations of the max jerk
- Stable path

Second configuration (fault-tolerant)

Shield tolerance: $\xi_s = s_{predicted} - s_{safe} = 30$ and $\xi_d = d_{predicted} - d_{safe} = 1$

Run	Path planner outputs	AI outputs	Shield usage	AI usage
#1	1023	871	15%	85%
#2	1075	946	12%	88%
#3	1008	792	22%	78%
#4	1096	797	28%	72%
#5	1122	953	15%	85%

Observations:

- No collision
- No violation of the speed limitation
- Violation of the maximum acceleration
- Some violations of the max jerk
- Stable path for most of the time

Shield tolerance: $\xi_s = s_{predicted} - s_{safe} = 40$ and $\xi_d = d_{predicted} - d_{safe} = 3$

Third configuration (extremely fault-tolerant)

Run	Path planner outputs	AI outputs	Shield usage	AI usage
#1	974	905	7%	93%
#2	986	946	8%	92%
#3	889	865	3%	97%
#4	1066	982	8%	92%
#5	1122	953	3%	97%

Observations:

- No collision
- No violation of the speed limitation
- No violation of the maximum acceleration
- Few violations of the max jerk

7 Future work and limitations

In this thesis the future work, which is extrapolated from this thesis's approach limitations, is discussed. Unfortunately, these limitations could not be overcome in this work due to time and resources constraints. This work would be then translated in a scientific paper for publications.

7.1 SCOTS limitations

In this thesis, the ego vehicle dynamics are only 2 dimensional (x-y positions). However, this model of the car is simplistic and irrelevant for real-life settings applications. Therefore, a realistic model such as the single track model, which is shown in below, should be implemented. Unfortunately, the current version (v1.0) of Scots does not provide fast synthesis for a 5 dimensional model. Indeed, the highway traffic simulation need to get a new path each 20ms, which is not always obtained with the 2 dimensional model. In order to overcome this issue, the new version of SCOTS (v2.0) will be implemented. Hence, the new version of SCOTS use parallel computing to synthesize symbolic controllers, which allows the use of complex dynamics such as the single track model [2].

$$\dot{x_1} = x_4 \ cos(x_5)$$
$$\dot{x_2} = x_4 \ sin(x_5)$$
$$\dot{x_3} = u_1$$
$$\dot{x_4} = u_2$$
$$\dot{x_5} = \frac{x_4}{l_{wb}} \ tan(x_3)$$

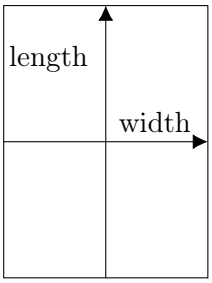
$x_1:$ x position ; $x_2:$ y position

$x_3:$ steering angle ; $x_4:$ velocity

$x_5:$ the heading ; $l_{wb}:$ wheelbase

$u_1:$ steering wheel velocity ; $u_2:$ acceleration

One other limitation is SCOTS is that the ego vehicle as a single point and does not take into account the vehicles dimensions. This feature is very important to the future work since obstacles need always need to be over-specified such as no collision occur. However, this was not provided by the new version of SCOTS. Therefore, a newer version should be implemented with feature such as the task of specifying the environment becomes easier.



7.2 Simulation limitations

The term 3 simulation used in this thesis was designed to send telemetry data to a websocket client and receive an x-y coordinates in order to steer the vehicle. Despite the fact that the path planning is important for autonomous driving, steering the vehicle with a steering wheel velocity and acceleration inputs is more relevant for the research field. In order to overcome this issue, the command server of the simulation should be modified such as it receives a steering wheel velocity and acceleration instead of x-y coordinates. This is already true for the simulation Term 2 which is also developed by Udacity. Therefore, the term 3 simulation should adopt the command server of the term 2 simulation in order to fix this issue.

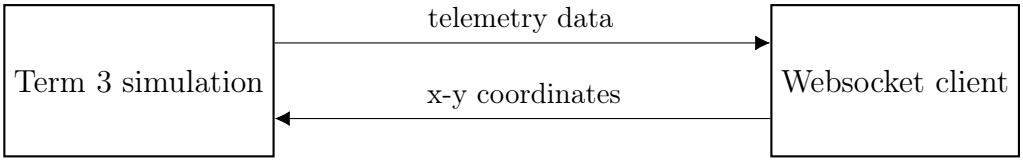


Figure 7.1: Term 3 simulation [5]

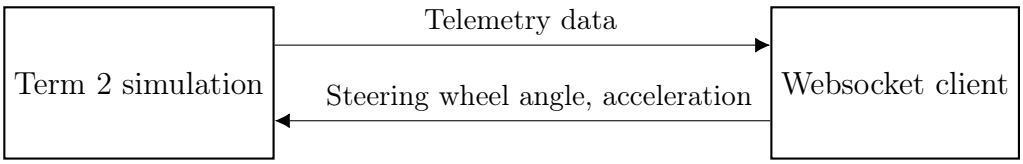


Figure 7.2: Term 2 simulation [6]

Bibliography

[1] Althoff, M. and Dolan, J. M. "Online verification of automated road vehicles using reachability analysis". In: *IEEE Transactions on Robotics* 30.4 (2014), pp. 903–918.

[2] Althoff, M., Koschi, M., and Manzinger, S. "CommonRoad: Composable benchmarks for motion planning on roads". In: *Intelligent Vehicles Symposium (IV), 2017 IEEE*. IEEE. 2017, pp. 719–726.

[3] Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., and Saár, Y. "Synthesis of reactive (1) designs". In: *Journal of Computer and System Sciences* 78.3 (2012), pp. 911–938.

[4] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. "End to end learning for self-driving cars". In: *arXiv preprint arXiv:1604.07316* (2016).

[5] Brown, A. *Udacity's Self-Driving Car Simulator*. Udacity, 2017. URL: `https://github.com/udacity/self-driving-car-sim/tree/term3_collection`.

[6] Brown, A. *Udacity's Self-Driving Car Simulator*. Udacity, 2017. URL: `https://github.com/udacity/self-driving-car-sim/tree/term2_collection`.

[7] Cheng, C., Diehl, F., Hinz, G., Hamza, Y., Nuehrenberg, G., Rickert, M., Ruess, H., and Truong-Le, M. "Neural networks for safety-critical applications — Challenges, experiments and perspectives". In: *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. Mar. 2018, pp. 1005–1006. DOI: `10.23919/DATE.2018.8342158`.

[8] Cheng, C.-H., Hamza, Y., and Ruess, H. "Structural synthesis for GXW specifications". In: *International Conference on Computer Aided Verification*. Springer. 2016, pp. 95–117.

[9] Chollet, F. et al. *Keras*. `https://github.com/fchollet/keras`. 2015.

[10] De Boor, C., De Boor, C., Mathématicien, E.-U., De Boor, C., and De Boor, C. *A practical guide to splines*. Vol. 27. Springer-Verlag New York, 1978.

[11] Fagnant, D. and Kockelman, K. M. "Preparing a nation for autonomous vehicles: Opportunities, barriers and policy recommendations". In: (2013).

[12] Fette, I. and Melnikov, A. *The websocket protocol*. Tech. rep. 2011.

[13] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016, pp. 373–402. URL: `http://www.deeplearningbook.org`.

[14] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016, pp. 423–444. URL: http://www.deeplearningbook.org.

[15] Hermann, T. *Keras*. https://github.com/Dobiasd/frugally-deep. 2018.

[16] Huval, B., Wang, T., Tandon, S., Kiske, J., Song, W., Pazhayampallil, J., Andriluka, M., Rajpurkar, P., Migimatsu, T., Cheng-Yue, R., et al. "An empirical evaluation of deep learning on highway driving". In: *arXiv preprint arXiv:1504.01716* (2015).

[17] Kingma, D. P. and Ba, J. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[18] Könighofer, B., Alshiekh, M., Bloem, R., Humphrey, L., Könighofer, R., Topcu, U., and Wang, C. "Shield synthesis". In: *Formal Methods in System Design* 51.2 (Nov. 2017), pp. 332–361. ISSN: 1572-8102.

[19] Koopman, P. and Wagner, M. "Challenges in autonomous vehicle testing and validation". In: *SAE International Journal of Transportation Safety* 4.2016-01-0128 (2016), pp. 15–24.

[20] Levinson, J., Askeland, J., Becker, J., Dolson, J., Held, D., Kammel, S., Kolter, J. Z., Langer, D., Pink, O., Pratt, V., et al. "Towards fully autonomous driving: Systems and algorithms". In: *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE. 2011, pp. 163–168.

[21] Pike, L., Niller, S., and Wegmann, N. "Runtime Verification for Ultra-Critical Systems". In: *Runtime Verification*. Ed. by Khurshid, S. and Sen, K. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 310–324. ISBN: 978-3-642-29860-8.

[22] Piterman, N. "From nondeterministic Buchi and Streett automata to deterministic parity automata". In: *Logic in Computer Science, 2006 21st Annual IEEE Symposium on*. IEEE. 2006, pp. 255–264.

[23] Piterman, N., Pnueli, A., and Sa'ar, Y. "Synthesis of Reactive(1) Designs". In: *Verification, Model Checking, and Abstract Interpretation*. Ed. by Emerson, E. A. and Namjoshi, K. S. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 364–380. ISBN: 978-3-540-31622-0.

[24] Pnueli, A. "The temporal logic of programs". In: *Foundations of Computer Science, 1977., 18th Annual Symposium on*. IEEE. 1977, pp. 46–57.

[25] Pnueli, A. and Rosner, R. "On the synthesis of an asynchronous reactive module". In: *Automata, Languages and Programming*. Ed. by Ausiello, G., Dezani-Ciancaglini, M., and Della Rocca, S. R. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 652–671. ISBN: 978-3-540-46201-9.

[26] Reissig, G., Weber, A., and Rungger, M. "Feedback refinement relations for the synthesis of symbolic controllers". In: *IEEE Transactions on Automatic Control* 62.4 (2017), pp. 1781–1796.

[27] Rungger, M. and Zamani, M. "SCOTS: A tool for the synthesis of symbolic controllers". In: *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*. ACM. 2016, pp. 99–104.

[28] Safra, S. "On the complexity of omega-automata". In: *Foundations of Computer Science, 1988., 29th Annual Symposium on*. IEEE. 1988, pp. 319–327.

[29] Schwarting, W., Alonso-Mora, J., and Rus, D. "Planning and decision-making for autonomous vehicles". In: *Annual Review of Control, Robotics, and Autonomous Systems* 1 (2018), pp. 187–210.

[30] Torres, D. M. *CarND-Path-Planning-Project-P1*. 2017. URL: `https://github.com/darienmt/CarND-Path-Planning-Project-P1/blob/master/src/main.cpp`.

[31] Udacity. *Udacity's Self-Driving Car Nanodegree*. 2017. URL: `https://github.com/udacity/CarND-Path-Planning-Project/blob/master/src/main.cpp`.

[32] Werling, M., Ziegler, J., Kammel, S., and Thrun, S. "Optimal trajectory generation for dynamic street scenarios in a frenet frame". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010, pp. 987–993.

[33] Wolper, P., Vardi, M. Y., and Sistla, A. P. "Reasoning about infinite computation paths". In: *Foundations of Computer Science, 1983., 24th Annual Symposium on*. IEEE. 1983, pp. 185–194.

[34] Wu, M., Zeng, H., and Wang, C. "Synthesizing runtime enforcer of safety properties under burst error". In: *NASA Formal Methods Symposium*. Springer. 2016, pp. 65–81.