2019 IEEE Intelligent Transportation Systems Conference (ITSC)
Auckland, NZ, October 27-30, 2019

Learning When to Drive in Intersections by Combining Reinforcement Learning and Model Predictive Control

Tommy Tram[1,2,3], Ivo Batkovic[1,2,3], Mohammad Ali[1], and Jonas Sjöberg[2]

Abstract— **In this paper, we propose a decision making algorithm intended for automated vehicles that negotiate with other possibly non-automated vehicles in intersections. The decision algorithm is separated into two parts: a high-level decision module based on reinforcement learning, and a low-level planning module based on model predictive control. Traffic is simulated with numerous predefined driver behaviors and intentions, and the performance of the proposed decision algorithm was evaluated against another controller. The results show that the proposed decision algorithm yields shorter training episodes and an increased performance in success rate compared to the other controller.**

I. Introduction

How can a self-driving vehicle interact and drive safely through intersections with other road users? Interactions between road users in intersections is a complex problem to solve, making it difficult to address using conventional rule based systems. Many advancements aim to solve this problem by trying to imitate human drivers [1] or predicting what other drivers in traffic are planning to do [2]. In [3], the authors show that by modeling the decision process as a partially observable Markov decision process, the model can account for uncertainty in sensing the environment and [4] showed some probabilistic guarantees when solving the problem using reinforcement learning (RL).

Previous research [5] showed that reinforcement learning can be used to learn a negotiation behavior between vehicles without vehicle to vehicle communication when driving in an intersection. The method found a policy that could avoid collisions in an intersection with crossing traffic, where other vehicles have different intentions. Since the previous work separates the framework in a high-level decision maker and a low-level controller, the high-level decision making algorithm can focus on the task when to drive, while the low level controller handles the comfort of passengers in the car by generating a smooth acceleration profile. We showed how this worked for intersections with a single crossing point, where Short Term Goal (STG) actions could choose one car to follow. This architecture, similar to Fig. 1, gives the decision algorithm, the RL policy, the flexibility to choose actions that can safely drive through the intersection by

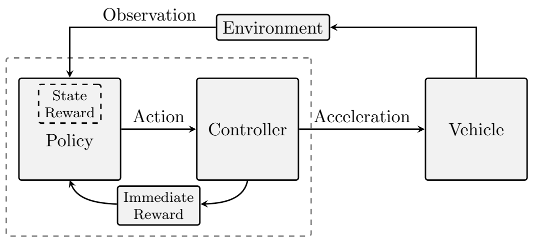


Fig. 1: Representation of the decision making architecture. The dashed line marks the decision algorithm that is separated into two parts; the high-level decision maker, denoted as the policy, and the low-level controller.

switching between numerous STG. A solution with a simple controller holds well when the distance between intersection are far away from each other, but when there are several crossing points in close succession, the system would have a hard time avoiding collisions due to the increased complexity of multiple points and timeing where a collision can occur.

In this paper, we instead propose to combine the high-level decision maker from [5] with a Model Predictive Controller (MPC) in a framework presented in Fig. 1. The performance of the MPC controller is benchmarked against a Sliding Mode controller that was used in [5]. The benefit of the MPC is that it can consider multiple vehicles at the same time and generate an optimal trajectory, which instantly gives feedback on performance and feasibility, i.e. predicting collisions, to the high-level decision maker. In contrast to [6], where the authors prove stability and recursive feasibility using an MPC approach and assuming that agents can cooperate, we restrict ourselves to non-cooperative scenarios.

Applying MPC directly to the problem could lead to a growing complexity with the number of vehicles in the intersection, e.g., the vehicle needs to decide based on multiple options which vehicle to yield for, and which to drive in front of. Therefore, we propose to separate the problem into two parts: the first being a high-level decision maker, which structures the problem, and the second being a low level planner, which optimizes a trajectory given the traffic configuration.

For the high-level decision maker, RL is used to find a policy for how the vehicle should drive through the intersection, and MPC is used as a low-level planner to optimize a safe trajectory. Compared to [7], [8] where all vehicles are controlled using MPC to stay in safe sets, based on models of other vehicles' future trajectory, this could possibly be perceived as too conservative for a passenger. By

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

[1] Affiliated with Zenuity AB, Gothenberg, Sweden {tommy.tram, ivo.batkovic, mohammad.ali}@zenuity.com
[2] Affiliated with the Department of Electrical Engineering, Chalmers University of Technology, Gothenberg, Sweden {tommy.tram, ivo.batkovic, jonas.sjoberg}@chalmers.se
[3] Affiliated with AI Innovation of Sweden, Gothenburg, Sweden

combining RL and MPC, the decision policy will learn which action is optimal by using feedback from the MPC controller in the reward function. Since MPC uses predefined models, e.g. vehicle models and other obstacle prediction models, the performance relies on their accuracy and assumptions. To mitigate this, we use Q-learning, a model-free RL approach, to maximize the expected future reward based on the experience gained during an entire episode. This approach is able to compensate, to some extent, for model errors and is explained more in Section IV-A.

In this work, we focus on the integration between policy and actuation, by having an MPC controller directly giving feedback to the decision maker through immediate reward, allowing the policy to know how comfortably the controller can handle an action and give feedback sooner if the predicted outcome may be good or bad.

This paper is structured as follows. Section II introduces the problem formulation along with the two-layers of the decision algorithm. Section III presents three agents used for simulation and validation. Implementation details are presented in Section IV, and the results are shown in Section V followed by discussion in Section VI. Finally, we draw conclusions in Section VII.

II. Problem formulation

The goal of the ego-vehicle is to drive along a predefined route that has one or two intersections with crossing traffic, where the intent of other road users is unknown. Therefore, the ego-vehicle needs to assess the driving situation and drive comfortably, while avoiding collisions with any vehicle[1] that may cross. In this section, we define the underlying Partially Observable Markov Decision Process (POMDP) and present how the problem is decomposed using RL for decision making and MPC for planning and control.

A. Partially Observable Markov Decision Process

A POMDP [9] is defined by the 7-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ an action space that is defined in section III-A, $\mathcal{T}$ the transition function, the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is defined in IV-D, Ω an observation space, $\mathcal{O}$ the probability of being in state s_t given the observation o_t, and γ the discount factor.

A POMDP is a generalization of the Markov Decision Process (MDP) [10] and therefore works in the same way in most aspects. At each time instant t, an action, $a_t \in \mathcal{A}$, is taken, which will change the environment state s_t to a new state s_{t+1}. Each transition to a state s_t with an action a_t has a reward r_t given by a reward function $\mathcal{R}$. The key difference from a regular MDP is that the environment state s_t is not entirely observable, e.g., the intention of other vehicles is not known. In order to find the optimal solution for our problem, we need to know the future intention of other drivers. Instead, we can only partially perceive the state though observations $o_t \in \Omega$.

[1]Although our approach can be extended to other road users, for convenience of exposition we'll refer to vehicles.

B. Q-Learning

In the reinforcement learning problem, an agent observes the state s_t of the environment, takes an action a_t, and receives a reward r_t at every time step t. Through experience, the agent learns a policy π in a way that maximizes the accumulated reward $\mathcal{R}$ in order to find the optimal policy π^*. In Q-learning, the policy is represented by a state action value function $Q(s_t, a_t)$. The optimal policy is given by the action that gives the highest Q-value.

$$\pi^*(s_t) = \arg\max_{a_t} Q^*(s_t, a_t) \tag{1}$$

Following the Bellman equation, the optimal Q-function $Q^*(s_t, a_t)$ is given by

$$Q^*(s_t, a_t) = \mathbb{E}[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) | s_t, a_t]. \tag{2}$$

III. Agents

The action space $\mathcal{A}$ is made out of six actions. The first two actions: α_1 take way, and α_2 give way, have the simple goal of crossing the intersection and stopping before the intersection, respectively. The actions α_{2+j} has the goal of following a vehicle j.

In the following, we explain the two agents used for control of the ego vehicle and how they apply each action and how the surrounding traffic is modeled with varying intentions.

A. MPC agent

We model the vehicle motion with states $\mathbf{x} \in \mathbb{R}^3$ and control $\mathbf{u} \in \mathbb{R}$, defined as

$$\mathbf{x} := [p^{\mathrm{e}} \quad v^{\mathrm{e}} \quad a^{\mathrm{e}}]^\top, \quad \mathbf{u} := j^{\mathrm{e}}, \tag{3}$$

where we denote the position along the driving path in a Frenet frame as p^{e}, the velocity as v^{e}, the acceleration as a^{e}, and the jerk as j^{e}, see Fig. 2a. In addition, we assume that measurements of other vehicles are provided through an observation $\mathbf{o}$. We limit the scope of the problem to consider at most four vehicles, and define the observations as

$$\mathbf{o} := [p^1 \quad v^1 \quad p_{\mathrm{ego}}^{\mathrm{cross},1} \quad \cdots \quad p^4 \quad v^4 \quad p_{\mathrm{ego}}^{\mathrm{cross},4} \quad]^\top, \tag{4}$$

where we denote the position along its path as p^j, the velocity as v^j, and $p_{\mathrm{ego}}^{\mathrm{cross},j}$ for $j \in [1, 4]$, as the distance to the ego-vehicle from the intersection point, see Fig. 2a. Note that we distinguish between $p_{\mathrm{ego}}^{\mathrm{cross},j}$, since vehicles can cross at different points, see Fig. 4.

We assume that there exists a lateral controller that stabilizes the vehicle along the driving path. To that end, we only focus on the longitudinal control. Given the state representation, the dynamics of the vehicle is then modeled using a triple integrator with jerk as control input.

The objective of the agent is to safely track a reference, i.e. follow a path with a target speed, acceleration, and jerk profile, while driving comfortably and satisfying constraints that arise from physical limitations and other road users, e.g. not colliding in intersections with crossing vehicles. Hence, we formulate the problem as a finite horizon, constrained optimal control problem

$$\min_{\bar{\mathbf{x}},\bar{\mathbf{u}}} \sum_{k=0}^{N-1} \begin{bmatrix} \bar{\mathbf{x}}_k - \mathbf{r}_k^{\mathbf{x}} \\ \bar{\mathbf{u}}_k - \mathbf{r}_k^{\mathbf{u}} \end{bmatrix}^\top \begin{bmatrix} \bar{Q} & \bar{S}^\top \\ \bar{S} & \bar{R} \end{bmatrix} \begin{bmatrix} \bar{\mathbf{x}}_k - \mathbf{r}_k^{\mathbf{x}} \\ \bar{\mathbf{u}}_k - \mathbf{r}_k^{\mathbf{u}} \end{bmatrix} \tag{5a}$$

$$+ \begin{bmatrix} \bar{\mathbf{x}}_N - \mathbf{r}_N^{\mathbf{x}} \end{bmatrix}^\top \bar{P} \begin{bmatrix} \bar{\mathbf{x}}_N - \mathbf{r}_N^{\mathbf{x}} \end{bmatrix}$$

$$\text{s.t.} \quad \bar{\mathbf{x}}_0 = \hat{\mathbf{x}}_0, \tag{5b}$$

$$\bar{\mathbf{x}}_{k+1} = A\bar{\mathbf{x}}_k + B\bar{\mathbf{u}}_k, \tag{5c}$$

$$h(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k, \bar{\mathbf{o}}_k, a) \leq 0, \tag{5d}$$

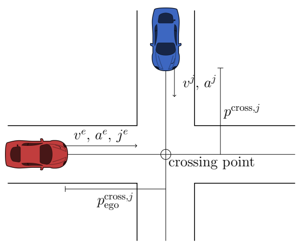
where k is the prediction time index, N is the prediction horizon, $\bar{Q}$, $\bar{R}$, and $\bar{S}$ are the stage costs, $\bar{P}$ is the terminal cost, $\bar{\mathbf{x}}_k$ and $\bar{\mathbf{u}}_k$ are the predicted state and control inputs, $\mathbf{r}_k^{\mathbf{x}}$ and $\mathbf{r}_k^{\mathbf{u}}$ are the state and control input references, $\bar{\mathbf{o}}_k$ denotes the predicted state of vehicles in the environment which need to be avoided, and a is the action from the high-level decision maker in Sec. II-B. Constraint (5b) enforces that the prediction starts at the current state estimate $\hat{\mathbf{x}}_0$, (5c) enforces the system dynamics, and (5d) enforces constraints on the states, control inputs, and obstacle avoidance.

The reference points, $\mathbf{r}_k^{\mathbf{x}}$, $\mathbf{r}_k^{\mathbf{u}}$ are assumed to be set-points of a constant velocity trajectory, e.g. following the legal speed-limit of the road. Therefore, we set the velocity reference according to the speed limit, and the acceleration and jerk to zero.

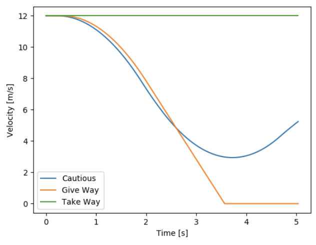
1) Obstacle prediction: In order for the vehicle planner in (5) to be able to properly avoid collisions, it is necessary to provide information about the surrounding vehicles in the environment. Therefore, similarly to [11], we assume that a sensor system provides information about the environment, and that there exists a prediction layer which generates future motions of other vehicles in the environment. The accuracy of the prediction layer does indeed affect the performance of the planner, however, since the high-level decision maker is separated from the low level control, the decisions can still be made robust to handle model errors and prediction errors.

In this paper, for simplicity the future motion of other agents is estimated by a constant velocity prediction model. The motion is predicted at every time instant for prediction times $k \in [0, N]$, and is used to form the collision avoidance constraints, which we describe in the next section. Even though more accurate prediction methods do exist, e.g. [12], [13], we use this simple model to show the potential of the overall framework.

2) Collision avoidance: We denote a vehicle j with the following notation $\mathbf{x}^j := [p^j \; v^j \; a^j]^\top$, and an associated crossing point at position $p^{\text{cross},j}$ in its own vehicle frame, which translated into the ego-vehicle frame is denoted as $p_{\text{ego}}^{\text{cross},j}$. For clarity, see Fig. 2a. With a predefined road topology, we assume that the vehicles will travel along the assigned paths, and that collisions may only occur at the intersection points $p^{\text{cross},j}$ between an obstacle and the ego vehicle. Hence, for collision avoidance, we use the predictions of the future obstacle states $\bar{\mathbf{x}}_k^j$ for times $k \in$



(a) Observations



(b) Velocity profile examples of various intention agents.

Fig. 2: Observations and agents in a scenario

$[0, N]$, provided by the prediction layer outside of the MPC framework. Given the obstacle measurements, the prediction layer will generate future states throughout the prediction horizon. With this information, it is possible to identify the time slots when a vehicle enters and exits the intersection.

Whenever an obstacle j is predicted to be within a threshold of $p^{\text{cross},j}$, e.g. the width of the intersecting area, the ego vehicle faces a constraint of the following form

$$\bar{p}_k^{\text{e}} \geq p_{\text{ego}}^{\text{cross},j} + \Delta, \quad \underline{p}_k^{\text{e}} \leq p_{\text{ego}}^{\text{cross},j} - \Delta,$$

where Δ ensures sufficient padding from the crossing point that does not cause a collision. The choice of Δ must be at least such that p_k together with the dimensions of the ego-vehicle does not overlap with the intersecting area.

3) Take way and give way constraint: Since the constraints from the surrounding obstacles become non-convex, we rely on the high-level policy maker to decide through action $a \in \mathcal{A}$ how to construct constraint (5d) for Problem (5). The take-way action implies that the ego-vehicle drives first through the intersection, i.e., it needs to pass the intersection before all other vehicles. This implies that for any vehicle j that reaches the intersection during prediction times $k \in [0, N]$, the generated constraint needs to lower bound the state p_k according to

$$\max_j p^{\text{cross},j} + \Delta \leq p_k^{\text{e}}. \tag{6}$$

Similarly, if the action is to give way, then the position needs to be upper bounded by the closest intersection point so that

$$p_k^{\text{e}} \leq \min_j p_{\text{ego}}^{\text{cross},j} - \Delta, \tag{7}$$

for all times k that the vehicle is predicted to be in the intersection.

4) Following an obstacle: If action $a \in \mathcal{A}$ is not chosen to give way, or to take way, the remaining options are to follow one of the j vehicles. For such choices on a the ego-vehicle position is upper bounded by $p_k^{\text{e}} \leq p_{\text{ego}}^{\text{cross,j}}$. For other vehicles $i \neq j$, we construct the following constraints

- if $p^{\text{cross,i}} < p^{\text{cross,j}}$ then $p^{\text{cross,i}} + \Delta \leq p_k^{\text{e}}$, which implies that the ego-vehicle should drive ahead of all vehicles i that are approaching the intersection;
- if $p^{\text{cross,i}} > p^{\text{cross,j}}$ then $p_k^{\text{e}} \leq p^{\text{cross,i}} - \Delta$, which implies that the ego-vehicle should wait to pass vehicle j and other vehicles i;

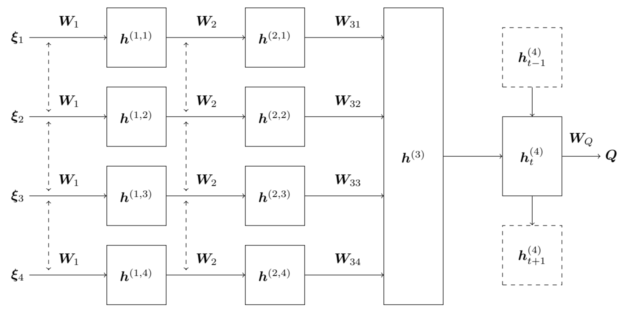


Fig. 3: Representation of the network structure

- if $p^{\text{cross,i}} = p^{\text{cross},j}$ then the constraints generated for vehicle i becomes an upper or lower bound depending on if vehicle i is ahead or behind vehicle j into the intersection.

B. Sliding mode agent

To benchmark the performance of using MPC, we introduce a Sliding Mode (SM) controller that was used in [5] and given by the following

$$a^e_{sm} = \frac{1}{c_2}(-c_1x_2 + \mu sign(\sigma(x_1, x_2))), \tag{8a}$$

$$\text{where} \begin{cases} x_1 = p^t - p^e, \\ x_2 = v^t - v^e, \end{cases} \tag{8b}$$

$$\sigma = c_1x_1 + c_2x_2, \tag{8c}$$

$$a^e_{\text{p}} = K(v_{\max} - v^e), \tag{8d}$$

$$a^e = \min(a^e_{\text{sm}}, a^e_{\text{p}}). \tag{8e}$$

The SM controller aims to keep a minimum distance to a target vehicle with a velocity of v^e, by controlling the acceleration a^e_{sm}. The tuning parameters c_1, c_2, and μ are used to tune the comfort of the controller. In case no target vehicle exists, the controller maintains a target velocity $v_{\max}$ with a proportional control law from (8d) with the proportional constant K. The final acceleration is given by (8e). For more details about the SM agent see [5].

C. Surrounding traffic agents

There are three intentions for agents in surrounding traffic. Examples of some velocity profiles are shown in Fig. 2b. The intention of all agents is implemented with a SM controller with various target values. The take way intention does not yield for the crossing traffic and simply aim to keep its target reference speed. The give way intention, however, slows down to a complete stop at the start of the intersection, until crossing traffic has passed. The third intention is cautious, i.e. slowing down but not to a full stop.

IV. Implementation

A. Deep Q-Network

The deep Q-network is structured as a three layer neural network with shared weights and a Long Short-Term Memory (LSTM) layer based on previous work [5] and shown in Fig. 3. A similar study for lane changes on a highway confirmed the importance of having equal weights for inputs that describe the state of interchangeable objects [14]. The input features ξ_n are composed of observations o_t, introduced in section II-A and shown in Fig. 2a, with up to four observed vehicles

$$\xi_n = [\; p^e_t \quad v^e_t \quad a^e_t \quad \delta^e \quad p^n_t \quad v^n_t \quad a^n_t \quad \delta^n \;]^T \tag{9}$$

Normalization of the input features is done by scaling the features down to values between $[-1, 1]$ using the maximum speed $v_{\max}$, maximum acceleration $a_{\max}$ and a car's sight range $p_{\max}$. Empty observations of other vehicles $[p^n_t \quad v^n_t \quad a^n_t \quad \delta^n]$ has a default value of $-\mathbf{1}$. The input vectors are sent though two hidden layers $\boldsymbol{h}^{(1,i)}$ and $\boldsymbol{h}^{(2,i)}$ with shared weights $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ respectively

$$\boldsymbol{h}^{(1,i)} = \tanh\left(\boldsymbol{W}_1\boldsymbol{\xi}_i + \boldsymbol{b}_1\right), \tag{10}$$

$$\boldsymbol{h}^{(2,i)} = \tanh\left(\boldsymbol{W}_2\boldsymbol{h}^{(1,i)} + \boldsymbol{b}_2\right). \tag{11}$$

The output of each sub-network is then sent through a fully connected layer

$$\boldsymbol{h}^{(3)} = \tanh\left(\sum_{i=1}^{4} \boldsymbol{W}_{3i}\,\boldsymbol{h}^{(2,i)} + \boldsymbol{b}_3\right), \tag{12}$$

that is then connected to an LSTM [15] layer that can store and use previous features

$$\boldsymbol{h}^{(4)}_t = \text{LSTM}\left(\boldsymbol{h}^{(3)}|\boldsymbol{h}^{(4)}_{t-1}\right). \tag{13}$$

The output from the LSTM is then sent through a final layer

$$\boldsymbol{Q} = (\boldsymbol{W}_Q\boldsymbol{h}^{(4)} + \boldsymbol{b}_4) \circ \boldsymbol{Q}_{\text{mask}}, \tag{14}$$

where the operator $\circ$ denotes pointwise multiplication, $\boldsymbol{Q}_{\text{mask}}$ is a masking vector described in the next section, $\boldsymbol{W}_Q$ and $\boldsymbol{b}_4$ are the weights and biases for the final layers, respectively. The optimal policy $\pi^\star$ is then given by maximizing the optimal action value function $\boldsymbol{Q}^\star$ as

$$\pi^\star(s_t) = \arg\max_{a_t} \boldsymbol{Q}^\star(s_t, a_t). \tag{15}$$

B. Q-masking

Q-masking [16] helps the learning process by reducing the exploration space by disabling actions the agent does not need to explore. If there are less than N cars, it would then be meaningless to choose to follow a car that does not exist. Which motivates masking off cars that do not exist. In previous work [5], a high negative reward was given when an action to follow a car that did not exist was chosen, while the algorithm continued with a default action take way. The agent quickly learned to not choose cars that did not exist, but with Q-masking, the agent does not have to explore these options. For further details about the training see [5].

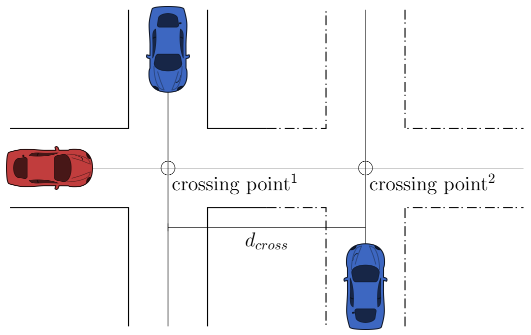


Fig. 4: Illustration of a intersection scenario, where the solid line is a single crossing and together with the dashed line creates a double crossing.

C. Simulation environment

All agents are spawned with a random intention, initial speed $v_0 \in [10, 30]$m/s, and position $p_0^i \in [10, 55]$m. The vehicle dimensions are 2 m wide and 4 m long. The ego car operates within comfort bounds and therefore has a limited maximum acceleration and deceleration of 5 m/s^2. Two main types of crossing were investigated. One and two crossing points as shown in Fig. 4, where the distance d_{cross} between crossing points vary between $[4, 8, 12, 25, 30, 40]$m with each scenarios.

The MPC agent was discretized at 30Hz, with a prediction horizon of $N = 100$ and cost tuning of

$$\bar{Q} = \text{blockdiag}(0.0, 1.0, 1.0),\ \ \bar{R} = 1,\ \ \bar{S} = \mathbf{0}. \tag{16}$$

D. Reward function tuning

There are three states that terminates an episode: success, failure, and timeout. Success is when the ego agent reaches the end of the road defined by the scenario. Failure is when the frame of the ego agent overlaps with another road users' frame, e.g., in a collision, this frame can be the size of the vehicle or a safety boundary around a vehicle. The final terminating state is timeout, which is simply when the agent can not reach the two previous terminating states before the elapsed time τ reaches the timeout time τ_m.

According to [17], the Q_π values and gradient can grow to be very large if the total reward values are too large. All rewards are therefore scaled with the episode timeout time τ_m, which is set to 25s, to keep the total reward $r_t \in [-2, 1]$. The reward function is defined as follows:

$$r_t = \begin{cases} 1 & \text{on success,} \\ -1 & \text{on failure,} \\ 0.5 & \text{on timeout, i.e. } \tau \geq \tau_m, \\ f(p_{\text{crash}}, p_{\text{comf}}) & \text{on non-terminating updates,} \end{cases}$$

where $f(p_{\text{crash}}, p_{\text{comf}})$ consists of

$$f(p_{\text{crash}}, p_{\text{comf}}) = \alpha p_{\text{crash}} \frac{\tau_m}{\tau - t_{pred}} + \beta p_{\text{comf}} \frac{\tau_m}{\tau}, \tag{17}$$

with $\alpha \in [0, 1]$, $\beta \in [0, 1]$ being weight parameters, and $\alpha + \beta = 1$. The first term p_{crash} corresponds to a feasibility check of Problem (5), which to a large extent depends on the validity of the accuracy of the prediction layer. The high-level decision from the policy-maker affects how the constraints are constructed, and may turn the control problem infeasible, e.g. if the decided action is to take way, while not being able to pass the intersection before all other obstacles. Therefore, whenever the MPC problem becomes infeasible we set $p_{\text{crash}} = 1$, otherwise $p_{\text{crash}} = 0$, to indicate that the selected action most likely will result in a collision with the surrounding environment. Because p_{crash} usually only triggers close to a potential collision, t_{pred} is set to the first time a crash prediction is triggered, to scale the negative reward relatively higher the later it is predicted.

The second term p_{comf} relates to the comfort of the planned trajectory, which is estimated by computing and weighting the acceleration and jerk profiles as

$$p_{\text{comf}} = \frac{1}{\sigma N}\left(\sum_{k=0}^{N-1} \bar{a}_k^2 \bar{Q}^a + \bar{j}_k^2 \bar{R}^j + \bar{a}_N^2 \bar{Q}^a\right),$$

where $\bar{a}$, and $\bar{j}$ are the acceleration component of the state and jerk component of the control, respectively, Q^a and R^j are the corresponding weights, and σ is a normalizing factor which ensures that $p_{\text{comf}} \in [0, 1]$. For the simulation we used $\bar{Q}^a = 1$ and $\bar{R}^j = 1$.

The timeout reward 0.5 was set to be higher than the average accumulated reward from p_{comf}, so that the total accumulated reward would be positive in case of timeouts.

V. Results

For evaluation we compared the success rate of the decision-policy together with a collision to timeout ratio (CTR). The success rate is defined as the number of times the agent is able to cross the intersections without colliding with other obstacles or exceeding the time limit to cross. Since we define a time-out to be a failure, we use the CTR to separate potential collisions from the agent being too conservative.

Fig. 5 shows a comparison in success rate between the proposed MPC architecture and the previous SM agent for scenarios with only one intersection. In this scenario, the MPC agent converges after 10^4 training episodes, while the previous SM agent converges after $4 \cdot 10^4$ training episodes. In addition, comparing the CTR metric, Tabel. I shows that the MPC agent has CTR of 0.45, while the SM agent has a CTR of 0.72. Evidently, it is apparent that the MPC is able to leverage future information into its planning horizon in order to achieve faster training, higher success rates, and also avoiding more collisions as a result.

We evaluate the performance of the MPC and SM agents for the more difficult double intersection problem, where we vary the distance between the intersection points. Table I shows the performance of the MPC and SM agent for both the single and double scenarios. The performance decreases for both agents in the double crossing scenario. However, it is evident that the MPC agent suffers less performance degradation compared to the SM agent. The CTR more than doubles for the MPC agent for the double crossing, while the already high CTR rate for the SM agent increases above

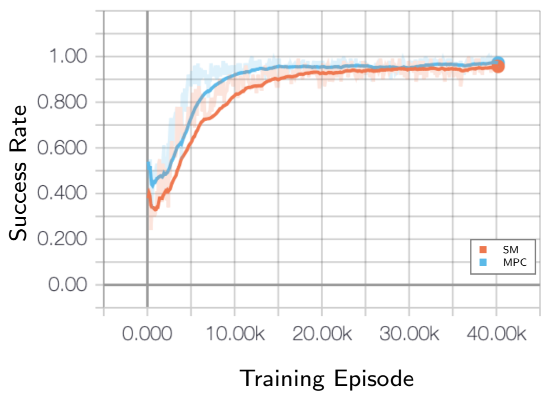


Fig. 5: Average MPC and SM success rate for a single corssing after evaluating the policy 300 episodes.

TABLE I: Average success rates and collision to timeout rates.

Controller	Success Rate		CTR	
	Single	Double	Single	Double
SM	96.1%	90.9%	72%	93%
MPC	97.3%	95.2%	45%	76%

rates of 0.9. Still, the MPC agent manages to outperform the SM agent.

VI. Discussion

The benefit of being able to use a prediction horizon for the MPC is shown to mostly impact the training time for the traffic scenarios compared to the SM agent. This allows the RL decision-policy to get feedback early in the training process to see whether an action most likely will lead to a collisions. In addition, the lower CTR also implies that the use of a prediction horizon also makes the decision-policy more conservative, since it rather times out than risk collisions.

It is important to note that only little effort was put into tuning the MPC agent, and that we used very primitive prediction methods that do not hold very well in crossing scenarios, e.g. the simulated agents did not keep constant speed profiles while approaching the intersections. However, under these circumstances, the decision algorithm still managed to obtain a success rate above 95% for the double crossings.

In practice, a full decision architecture system would include a safety layer that limits which acceleration values the system can actuate in order to stay safe, followed by the decisions algorithm from this work that generates an acceleration request. The environment state, together with the new acceleration request, could be sent through a collision avoidance system that checks if the current path has a collision risk, and avoiding collision by allowing higher acceleration limits. This way, a failure would correspond to a intervention by the collision avoidance system instead of a crash.

VII. Conclusion

In this paper, we proposed a decision making algorithm for intersections which consists of two components: a high-level decision maker that uses Deep Q-learning to generate decisions for how the vehicle should drive through the intersection, and a low-level planner that uses MPC to optimize safe trajectories. We tested the framework in a traffic simulation with randomized intent of other road users for both single and double crossings. Results showed that the proposed MPC agent outperforms the previous SM agent with 95.2% success rate in scenarios with double crossings compared to 90.9% for the SM agent. Results also showed that the crash timeout ratio was also significantly lower at 45% for the MPC agent compared to 72% for the SM agent in single crossings. Meaning, the proposed method is better at handling scenarios with multiple intersections and vehicles.

References

[1] M. Bansal, A. Krizhevsky, and A. Ogale, "ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst," 2018.
[2] A. Zyner, S. Worrall, J. Ward, and E. Nebot, "Long short term memory for driver intent prediction," in *IEEE Intelligent Vehicles Symposium, Proceedings*, 2017.
[3] S. Brechtel, T. Gindele, and R. Dillmann, "Probabilistic decision-making under uncertainty for autonomous driving using continuous pomdps," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2014, pp. 392–399.
[4] M. Bouton, J. Karlsson, A. Nakhaei, K. Fujimura, M. J. Kochenderfer, and J. Tumova, "Reinforcement learning with probabilistic guarantees for autonomous driving," in *Workshop on Safety Risk and Uncertainty in Reinforcement Learning, Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.
[5] T. Tram, A. Jansson, R. Grönberg, M. Ali, and J. Sjöberg, "Learning negotiating behavior between cars in intersections using deep q-learning," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov 2018, pp. 3169–3174.
[6] R. Hult, M. Zanon, S. Gros, and P. Falcone, "Optimal coordination of automated vehicles at intersections: Theory and experiments," *IEEE Transactions on Control Systems Technology*, pp. 1–16, 2018.
[7] X. Qian, J. Gregoire, A. de La Fortelle, and F. Moutarde, "Decentralized model predictive control for smooth coordination of automated vehicles at intersection," in *2015 European Control Conference (ECC)*, July 2015, pp. 3452–3458.
[8] G. R. de Campos, P. Falcone, and J. Sjöberg, "Autonomous cooperative driving: a velocity-based negotiation approach for intersection crossing," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. IEEE, 2013, pp. 1456–1461.
[9] M. J. Kochenderfer *et al.*, *"Decision Making Under Uncertainty: Theory and Application"*. The MIT Press, 2015.
[10] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957.
[11] I. Batkovic, M. Zanon, M. Ali, and P. Falcone, "Real-time constrained trajectory planning and vehicle control for proactive autonomous driving with road users," in *European Control Conference (ECC) 2019*, 2019.
[12] S. Lefèvre, D. Vasquez, and C. Laugier, "A survey on motion prediction and risk assessment for intelligent vehicles," *ROBOMECH journal*, vol. 1, no. 1, p. 1, 2014.
[13] I. Batkovic, M. Zanon, N. Lubbe, and P. Falcone, "A computationally efficient model for pedestrian motion prediction," in *2018 European Control Conference (ECC)*, June 2018, pp. 374–379.
[14] C. Hoel, K. Wolff, and L. Laine, "Automated speed and lane change decision making using deep reinforcement learning," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov 2018, pp. 2148–2155.
[15] S. Hochreiter and J. Urgen Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: http://www7.informatik.tu-muenchen.de/ hochreit http://www.idsia.ch/ juergen
[16] M. Mukadam, A. Cosgun, A. Nakhaei, and K. Fujimura, "Tactical decision making for lane changing with deep reinforcement learning," in *NIPS Workshop on Machine Learning for Intelligent Transportation Systems*, 2017.
[17] H. P. van Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver, "Learning values across many orders of magnitude," in *NIPS*, 2016.