Efficient Speed Planning in the Path-Time Space for Urban Autonomous Driving

Thibaud Duhautbout, Reine Talj, Véronique Cherfaoui, François Aioun, Franck Guillemard

Efficient Speed Planning in the Path-Time Space for Urban Autonomous Driving

Thibaud Duhautbout[1,2], Reine Talj[1], Véronique Cherfaoui[1], François Aioun[2], Franck Guillemard[2]

***Abstract*— In this article, an algorithmic speed planning method for an autonomous vehicle dealing with moving obstacles is presented. Using the path-time space to represent collision zones with other vehicles, algorithms are proposed to pass before and after, while ensuring the respect of safety distances. Simulation results are proposed to show the generated speed profiles in common driving scenarios.**

I. INTRODUCTION

Autonomous driving for passenger vehicles is still an ongoing topic in public and private research today. Advanced driving assistance systems become more and more available in the cars, but autonomous functions are rare and usually limited to a few situations, such as highways or traffic jams. The urban environment is still a challenge, due to its highly dynamic nature which requires a constant adaptation to the evolution of the scene. Defining an appropriate speed profile is thus an important task to realize to enable self driving on urban roads. In this article, we present a new algorithmic speed planning method, used to compute efficient and safe predictive speed profiles on predefined paths, adjusted to the other actors that cross the way of the ego-vehicle. This algorithm relies on the path-time diagram to check and adapt the candidate trajectories in order to avoid potential collisions and ensure safety margins.

The remainder of the paper is organized as follows. Section II reviews some existing works about speed planning. Section III presents the path-time space and the structure used by our algorithm to represent the interactions with the other actors of the scene, and Section IV details the algorithm used to compute the speed profile. In Section V, we provide some simulation results to show the advantages of the proposed method. Finally, Section VI will draw a conclusion and point out some future works.

II. RELATED WORKS

Many works have addressed the problem of motion planning in the literature, to compute the right trajectory for a vehicle or more generally a mobile robot [1]. Trajectory planning can be done either by combined optimization of longitudinal and lateral motion, or by separated path and speed planning. While combined motion planning and path planning have been widely studied, we find that speed planning alone has received less attention. However, it is important for an autonomous vehicle to properly adapt its speed to the situation, more so in a dynamic urban environment.

In [2], a combined optimization approach is used to perform trajectory planning in an urban scenario. This method consists in optimizing an objective cost function under constraints, such that the resulting trajectory is compatible with the vehicle's physical constraints, and collisions are avoided. This requests a heavy optimization framework which might not be adapted to a high frequency replanning. Other combined approaches use a graph representation. In [3], state lattices are used to directly generate a collision-free trajectory from a spatiotemporal graph. This method suffers from rigidity due to the fixed discretization of the lattice. [4] integrate speed and acceleration in the lattice to provide more alternatives, but the search algorithm is implemented on GPU to provide acceptable planning times. To improve flexibility, the authors of [5] use a spatiotemporal RRT* algorithm to find a coarse valid trajectory, which is then refined by optimization. However, finding an optimal trajectory can make it more difficult to integrate behavior-based rules such as passing before or after an obstacle.

Regarding the path-velocity decomposition approaches, [6] introduces a hybrid trajectory planner for static environment, in which the speed is optimized along a previously generated path to minimize the travel time under physical and comfort constraints. In [7], a speed smoothing algorithm is presented to generate a time-optimal speed profile along a predefined path with position-based speed constraints. Both of these methods do not consider dynamic obstacles. In [8], speed planning is realized with a MPC approach, and dynamic obstacles are considered by a constraint in the optimization problem. In [9], a motion planning approach is proposed which handles dynamic obstacles by applying a different strategy depending on the orientation of the obstacle. In the case that the obstacle moves perpendicularly to the ego-vehicle, the maximum speed is iteratively reduced until a valid trajectory can be found. However, increasing the acceleration to pass before a vehicle is not presented.

In [10], the coordination space is used to plan speed profiles for multiple vehicles at an intersection to avoid collisions. This representation is interesting but its dimension is linked to the number of considered vehicles, which makes it quickly complex. In [11], trajectories are represented in a path-time diagram, on which spatio-temporal constraints for the vehicles are represented. This representation is interesting but in this case, it was just used to visually check the safety of the trajectory. In [12], a safety corridor is defined in

[1]Université de technologie de Compiègne, CNRS, Heudiasyc (Heuristics and Diagnosis of Complex Systems), CS 60 319 - 60 203 Compiègne Cedex (name.surname@hds.utc.fr)

[2]Stellantis, Centre Technique de Vélizy, Route de Gisy, 78140 Vélizy-Villacoublay (name.surname@stellantis.com)

This work was realized under cooperation contract between Stellantis and Heudiasyc laboratory.

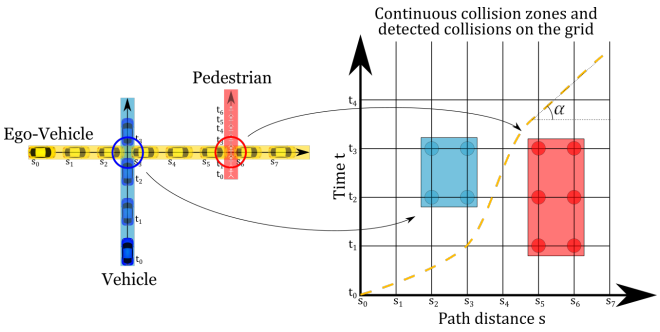


Fig. 1: Illustration of the path of the EV and of two other vehicles on the left with their predicted positions, and associated collision zones on the right.

this space so that the trajectory can be optimized inside for highway driving. In [13], similar constraints are represented in this space and a spatiotemporal A* planning algorithm is used to compute trajectories with respect to these constraints. In the present work, this path-time space is used to represent constraints linked to other dynamic actors, frequently seen in urban situations. Our main contribution is to propose a speed planning algorithm to adjust the speed based on this representation capable to generate efficient and safe speed profiles for the ego-vehicle.

III. THE PATH-TIME SPACE

A. Path-Time Representation

The path-time diagram, illustrated in Figure 1, is used to represent the interactions between the ego-vehicle (EV) and the other dynamic actors of the scene. The horizontal axis of this diagram corresponds to the position s of EV along a previously computed path on which we want to build an appropriate speed profile. The vertical axis represents the time in the future. By convention, when running the planning algorithm, the EV is supposed to be located at $(s = 0, t = 0)$. We chose to put the position on the horizontal axis so that the evolution of the EV along the path can be visualized from left to right. A trajectory for the EV can thus be represented in this path-time diagram by plotting the $t(s)$ function, which is the $s(t)$ function mirrored by the $s = t$ line.

In this representation, if we note $\alpha(s)$ the angle of the tangent at the trajectory with the horizontal axis at position s, then the speed of the EV is given by $v(s) = 1/\tan\alpha(s)$. Indeed, because the trajectory is given by $t(s)$, $\tan\alpha(s) = dt/ds(s) = 1/v(s)$, hence the result. A null speed will be represented by a vertical tangent, and a high speed will be represented by a tangent close to the horizontal axis.

Let us denote V_k another vehicle or actor of the scene, for which a motion prediction is available. If the predicted trajectory for V_k crosses the path of the EV, it is then possible to deduce a space-time area (the *collision zone*) that the EV must not cross to avoid the collision. This area corresponds to the set $CZ_{V_k} = \{(s_{CZ}, t_{CZ})\}$ such that if the EV is at the space-time coordinates $(s, t) \in CZ_{V_k}$, then it will collide with V_k. Figure 1 represents the path of the EV and two other vehicles, and the associated collision zones. It should be noted that this set is not necessarily connected and may be composed of multiple connected subsets, depending on the paths of the EV and V_k. In this case, all subsets of the collision zone must be avoided.

B. Collision Matrix Construction

In theory, the collision zones are continuous areas. To integrate the speed planning algorithm with our representations of the path and predictions for the other vehicles, we use a discretization of the path-time diagram to represent the collision zones in what we call the *collision matrix*.

Let $\mathcal{C} = \{P_i, i \in \{1, \dots, N\}\}$ be a path computed for the EV, composed of N representative 2D points P_i, and $S_\mathcal{C} = \{s_i, i \in \{1, \dots, N\}\}$ the curvilinear abscissa associated with this path $\mathcal{C}$. Let $\mathcal{V}$ be a set of K predictions $\mathcal{P}_{V_k} = \{P'_j, j \in \{1, \dots, M\}\}$ $(k \in \{1, \dots, K\})$ for the dynamic actors of the scenes, and $\mathcal{T} = \{t_j, j \in \{1, \dots, M\}\}$ the associated time discretization, which is supposed to be regular with a time step δ_t. By convention, $s_1 = 0$ in the EV path and $t_1 = 0$ in the time discretization.

A polygon is associated to each position P_i of the EV and P'_j of V_k, representing their global shape and oriented accordingly. An intersection-checking function is used to determine if these polygons intersect or not. In the following, when referring to the intersection between points P_i and P'_j, we mean the intersection between their associated polygons.

The collision matrix CM is defined by an $N \times M$ matrix of integers, indexed starting from 1, where

$$\begin{cases} CM(i,j) = 0 & \text{if } [s_i, s_{i+1}] \times [t_j, t_{j+1}] \text{ is free} \\ CM(i,j) = \mathbf{\Lambda} & \text{if not.} \end{cases} \quad (1)$$

$\mathbf{\Lambda}$ corresponds to a value identifying the connected collision zone in the collision matrix. $[s_i, s_{i+1}] \times [t_j, t_{j+1}]$ represents the interval on the path of the EV between positions s_i and s_{i+1} at time between t_j and t_{j+1}, and corresponds to a rectangle in the path-time diagram. This interval is said to be free if it can be traveled by the EV without colliding with any other vehicle. Because each value of the collision matrix corresponds to a rectangle, the collision matrix can be represented by a grid in the path-time plane, as proposed by Figure 1.

If an intersection is found between points P_i of the EV and P'_j for V_k, then this means that the position of the EV at position s_i intersects that of V_k at time t_j. Due to the discretization, at the border of the collision zone, there may be an intersection between P_i and P'_j, but not between P_{i-1} and P'_j. In this case, you only can deduce that the collision occurs between s_{i-1} and s_i, and lasts between s_i and s_{i+1}. The same reasoning holds with respect to time: there may not be a collision between P_i and P'_{j-1}, and you can only deduce that the collision occurs between t_{j-1} and t_j and lasts between t_j and t_{j+1}. The representation can naturally handle letting a safe time at the front of the other vehicles. Here, if the collision occurs between t_{j-1} and t_j, then given a safety time t_{safe}, the EV must avoid the current position from $t_{j-1} - t_{safe}$ to t_j. To map $t_{j-1} - t_{safe}$ in the temporal

Algorithm 1: Collision Matrix computation

```
μ ← ⌊t_safe/δ_t⌋ + 1 ;
for i ← 1 to N do
    // For each point P_i of the EV path
    for k ← 1 to K do
        // For each prediction P_{V_k}
        for j ← 1 to M do
            // For each point P'_j of P_{V_k}
            if P_i intersects P'_j of prediction P_{V_k} then
                for a ∈ {i − 1, i} do
                    for b ∈ {j − 1 − μ, ..., j} do
                        CM(a, b) ← −1 ;
```

discretization, it can be noted that the number of steps μ required to fully cover the time t_{safe} is given by :

$$\mu = \left\lfloor \frac{t_{safe}}{\delta_t} \right\rfloor + 1$$

Therefore, if a collision is found between P_i and P'_j, the interval $I = [s_{i-1}, s_{i+1}] \times [t_{j-1-\mu}, t_{j+1}]$ must be marked not free. This corresponds to the values $CM(a, b)$ such that $a \in \{i-1, i\}$ and $b \in \{j-1-\mu, \ldots, j\}$.

Algorithm 1 gives a naive approach to compute the collision matrix. The processing time of building the CM matrix is bounded by $K \times N \times M$ polygon intersection checks at most. This can be accelerated by using geometric filters or spatial indexing, but for the sake of clarity, these details are not presented here. Further information on this topic can be found in [14]. Checks for valid indices values are not presented either, but a proper implementation should consider these edge cases.

It can be noted that the algorithm fills the CM matrix with -1 values when an intersection is found. To deduce the $\mathbf{\Lambda}$ values presented in Equation 1, a clustering algorithm must be applied to give a unique value to each connected zone in the matrix. This can be done for example by defining a value $\mathbf{\Lambda}_1$ to replace a -1 in a cell, and by recursively propagating this value to all neighbour cells. Then, a new value $\mathbf{\Lambda}_2$ is defined and applied to a remaining cell with a -1 value, until all -1 values have been replaced.

The main advantage of this representation is that, because it is based on the path, it is indifferent to the executed trajectory. Checking a candidate trajectory can thus be realized efficiently with the path-time diagram. It should also be noted that, because a motion prediction for the dynamic actors is used, their global motion is considered in the collision matrix. Hence, if the dynamic actors have lateral speed or a strange behavior, or if they cross the path of the EV in a non-perpendicular fashion, this will be accounted for in the collision matrix. The definition of this prediction is out of the scope of this paper, relevant information can be found in [15].

IV. SPEED PLANNING ALGORITHM

The aim of the following speed planning algorithm is to find a valid trajectory, i.e. one which avoids crossing the collision zones in the collision matrix. The algorithm first generates a comfortable trajectory without considering the collision zones, and then checks its validity with respect to the collision matrix. If the trajectory is not valid, it is adjusted to avoid the collisions. Two options can be taken to adjust a trajectory: either pass before or pass after another dynamic actor. In the path-time space, passing before means that the curve of the trajectory is below the collision zone, and passing after means the curve is above the collision zone. Both options are explored by the algorithm, as will be described by the following sections.

A. Definitions and notations

The following algorithms will all refer to the path $\mathcal{C}$ and its associated curvilinear abscissa $\mathcal{S}_\mathcal{C}$ as defined in Section III.

A trajectory T is defined as position s, time τ and speed v values:

$$T = \{X_i = (s_i, \tau_i, v_i), i \in \{1, \ldots, N\}\} \quad (2)$$

where s_i are the elements of $\mathcal{S}_\mathcal{C}$, and where the points X_i are ordered by time, such that $\tau_{i+1} > \tau_i$. It it worth noting that the trajectory can exceed the time horizon of the prediction of the moving obstacles, meaning that only the points such that $\tau_i \leq t_M$ will be considered in the checking algorithm.

For the path $\mathcal{C}$, a set V_{max} of N values is provided, which holds the maximum speed allowed on the corresponding points of $\mathcal{C}$. Acceleration and deceleration bounds are given by a comfortable and a maximum value: $[a_{comf}, a_{max}]$ for the acceleration and $[d_{comf}, d_{max}]$ for the deceleration. The deceleration values are positive but should be interpreted as reducing the speed.

B. Trajectory checking

The function ***CheckTrajectory***(T, CM) checks the validity of T with respect to CM.

In order to ensure the safety time t_{safe} at the front of the ego-vehicle, the trajectory of the virtual point located with the required advance is defined by:

$$T_{av} = \left\{\bar{X}_i = (s_i + t_{safe} \cdot v_i, \tau_i, v_i), i \in \{1, \ldots, N\}\right\} \quad (3)$$

To determine if the trajectory is safe, the algorithm must check if the space between the two curves of these trajectories overlaps with a collision zone in the collision matrix. The points are processed in order, such that if a collision zone is crossed, it is necessarily the first encountered by the EV. If the proposed trajectory is not valid, the function returns the $\mathbf{\Lambda}$ value of the first collision zone crossed in CM. In the other case, the function returns the value -1. In sections IV-C and IV-D, it is considered that the initial trajectory T is invalid with respect to the collision zone $\boldsymbol{\lambda}$ (directly referred to as $\boldsymbol{\lambda}$), and the algorithms used to adjust the trajectories are presented. It should be noted here that the algorithms are presented on the whole trajectory with only one collision zone, but could be applied only on small portions of the $\mathcal{S}_\mathcal{C}$ axis to be used with multiple collision zones at different positions on the path.

Algorithm 2: Pass-before $\boldsymbol{\lambda}$ trajectory computation

```
// Check for the maximal acceleration
T1 ← SmoothTrajectory(T, V_max, a_max) ;
if CheckTrajectory(T1, CM) = λ then
    return no solution
// There is a solution, find the minimal
   acceleration value
a⁻ ← a_comf ;                          // lower bound
a⁺ ← a_max ;                           // upper bound
while a⁺ − a⁻ ≥ ε_a do
    a ← (a⁺ + a⁻)/2 ;                  // middle value
    T2 ← SmoothTrajectory(T, V_max, a) ;
    if CheckTrajectory(T2, CM) = λ then
        a⁻ ← a ;                       // increase acceleration
    else
        a⁺ ← a ;                       // decrease acceleration
        T1 ← T2 ;                      // save the valid result
return T1
```

C. Pass-before trajectory

Algorithm 2 presents the procedure to compute the pass-before trajectory with respect to $\boldsymbol{\lambda}$, with a result illustrated on Figure 2. It is assumed that the function ***SmoothTrajectory***(T,V_{max},a) generates a trajectory with a smoothed speed profile on the same s values as T, below the maximum speed V_{max} and with acceleration a. The deceleration bounds are fixed so they do not appear in the arguments.

First, the algorithm checks if the maximal acceleration is enough to pass before $\boldsymbol{\lambda}$. If not, this means that it is not possible to find a solution, thus no trajectory is returned, and only the pass-after trajectory will be considered. If a trajectory $T1$ is found, the acceleration value is then adjusted by dichotomy in the $[a_{comf}, a_{max}]$ interval to efficiently find the minimum value giving a valid result. The threshold ε_a defines the end of the dichotomy process. In the algorithm, variable $T1$ acts as a buffer variable so that when the dichotomy is finished, the last valid trajectory is directly available and can be returned without computing it again.

This acceleration adjustment can be useful in situations in which the EV needs to accelerate to evade a crossing, or to merge in the traffic when starting from a lower speed. The maximum speed value is not modified in this algorithm, so that the legal speed profile is not exceeded. Figure 2 illustrates such a situation, where the EV starts at a low speed, and an other vehicle is expected to cross its path in about 4 seconds. As can be seen, the initial trajectory crosses $\boldsymbol{\lambda}$, so the pass-before trajectory increases the acceleration to avoid it and pass below.

D. Pass-after trajectory

When the EV needs to slow down to pass after another actor ($\boldsymbol{\lambda}$), multiple strategies can be taken. A first way could be to start slowing down at the last moment to pass after $\boldsymbol{\lambda}$. Another way could be to immediately slow down, until the re-acceleration trajectory can safely pass after $\boldsymbol{\lambda}$. A last way could be also to immediately slow down but to a certain point, keep the speed at this lower level and then

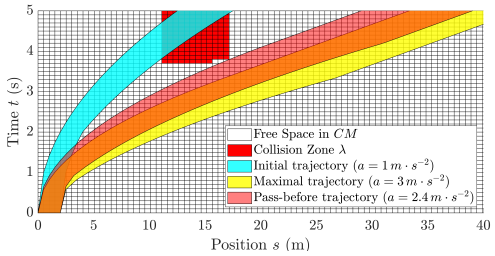


Fig. 2: Illustration of the pass-before trajectory computed in Algorithm 2

Algorithm 3: Pass-after $\boldsymbol{\lambda}$ trajectory computation

```
// Compute the immediate braking trajectory
   which avoids λ
T1 ← BrakingTrajectory(T, CM, λ, d_comf, d_max) ;
if CheckTrajectory(T1, CM) = λ then
    // λ cannot be avoided
    return T1 ;
for i ← 1 to N do
    // Reduce the maximum speed from i
    Ṽ_max ← T1[v] ;                    // speed values from T1
    Ṽ_max(i+1:N) ← min(Ṽ_max(i+1:N), Ṽ_max(i)) // [1]
    T2 ← SmoothTrajectory(T1, Ṽ_max, a_comf) ;
    if CheckTrajectory(T2, CM) ≠ λ then
        // T2 is the first valid trajectory
        i_s ← i ;
        exit for loop ;
for i ← i_s + 1 to N + 1 do
    // Reset the maximum speed from i
    Ṽ_max ← T2[v] ;                    // speed values from T2
    Ṽ_max(i:N) ← V_max(i:N);
    T3 ← SmoothTrajectory(T2, Ṽ_max, a_comf) ;
    if CheckTrajectory(T3, CM) ≠ λ then
        return T3 ;
```

re-accelerate to reach the reference speed while passing after $\boldsymbol{\lambda}$. The last strategy is the one which has the lowest impact on longitudinal acceleration. This approach is then used to define the pass-after function, presented on Algorithm 3 and illustrated on Figure 3.

First, the algorithm computes the minimum deceleration value to apply immediately so that the EV is able to stop before crossing $\boldsymbol{\lambda}$. This is done by the function ***BrakingTrajectory***(T, CM, $\boldsymbol{\lambda}$, d_{comf}, d_{max}), which uses the same dichotomy principle as used in Algorithm 2 to generate the new trajectory $T1$. If the braking trajectory crosses $\boldsymbol{\lambda}$, the deceleration is increased to brake stronger; if not, the deceleration is reduced to preserve comfort. The obtained trajectory must be checked in case it is not possible to avoid crossing $\boldsymbol{\lambda}$, even with the highest braking. In this situation, given the dichotomy rules, $T1$ is an emergency braking trajectory which is returned as a fallback strategy.

Then, the algorithm finds the first point of $T1$ from which

[1]Here, $a{:}b = \{a, \ldots, b\}$ with a and b integers. If $a > b$, $a{:}b = \emptyset$ and the associated line in the algorithm does nothing.

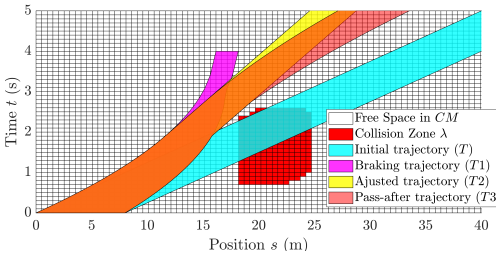


Fig. 3: Representation of the different trajectories computed in Algorithm 3

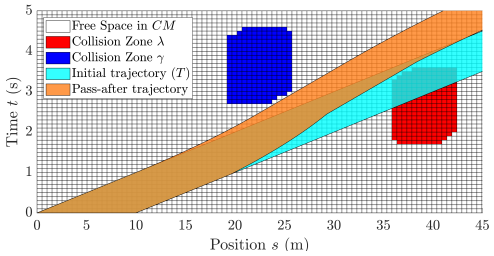


Fig. 4: Trajectory obtained when passing between two collision zones

the speed can be kept constant. At each point i, a new trajectory $T2$ is generated by using the ***SmoothTrajectory*** function with the comfortable acceleration. A new maximum speed vector is computed by copying the braking speed values of $T1$ between 1 and i, and using the obtained speed at i as an upper bound for the values V_{max} on the remaining points. The idea is to progressively increase the braking duration until the reached speed allows to pass after $\boldsymbol{\lambda}$. Such a solution will be obtained at worst at the last iteration of the loop, because it will generate the braking trajectory $T1$ which was checked valid. The first solution found will be the trajectory which has the shortest braking time, and thus which limits the most the speed loss.

Finally, the algorithm continues on the points to find the index from which the initial maximum speed can be reset. A solution can be found here because the trajectory $T2$ was checked valid before, and at the last iteration when $i{=}N{+}1$, we have $T3 = T2$, so $T3$ will be valid.

Figure 3 presents the different trajectories $T1$, $T2$ and $T3$ computed by the algorithm on a particular case. In the algorithm, the different loops are presented checking all the points from the beginning for clarity, but they can be improved by using a dichotomy over the indices.

It can be noted that this algorithm only considers one collision zone $\boldsymbol{\lambda}$. However, it is possible that when adjusting the trajectory between the initial position of the vehicle and $\boldsymbol{\lambda}$, a second collision zone $\boldsymbol{\gamma}$, located before $\boldsymbol{\lambda}$ on the $\mathcal{S}_\mathcal{C}$ axis, gets crossed. Since the new trajectory is slower, this means that the EV was able to pass before $\boldsymbol{\gamma}$ with its initial trajectory. To keep passing before $\boldsymbol{\gamma}$, the EV thus has to keep its initial speed longer before starting to brake. Therefore, Algorithm 3 can be adjusted to adapt the initial braking point on the trajectory, until a solution is found to pass after $\boldsymbol{\lambda}$ while passing before $\boldsymbol{\gamma}$, as illustrated in Figure 4. As can be seen, the adjusted trajectory follows the initial one for about one second before starting to slow down, to make sure to pass before $\boldsymbol{\gamma}$. Depending on the distance between $\boldsymbol{\lambda}$ and $\boldsymbol{\gamma}$ and on the braking parameters, it is not always possible to pass between them. In this case, the algorithm defaults to passing after $\boldsymbol{\gamma}$.

E. Adaptation to multiple obstacles

As explained at the beginning of this section, the presented algorithms adjust the speed to one particular collision zone, but they can be applied on reduced parts of the path to adapt to multiple collision zones. Indeed, although the pass-after algorithm can account for another collision zone located before the first one, it is not possible, for example, to pass after one obstacle and then adjust the speed to pass before another one. Let us consider two collision zones $\boldsymbol{\lambda}$ and $\boldsymbol{\gamma}$, such that $\boldsymbol{\lambda}$ is located before $\boldsymbol{\gamma}$ on the $\mathcal{S}_\mathcal{C}$ axis. Consider also the initial trajectory T of the EV which crosses $\boldsymbol{\lambda}$, and the computed pass-after trajectory $T1$ which avoids $\boldsymbol{\lambda}$ but crosses $\boldsymbol{\gamma}$. If the presented algorithms are directly applied to adjust T to $\boldsymbol{\gamma}$ from the beginning, the resulting trajectories might cross $\boldsymbol{\lambda}$ again.

To make this possible, Algorithms 2 and 3 are used as elementary functions operating on reduced portions of the $\mathcal{S}_\mathcal{C}$ axis. In such a situation, the initial trajectory T should be adjusted to the first collision zone $\boldsymbol{\lambda}$. Once the adjusted trajectory T' is computed, it is possible to find the first point i_p at which $\boldsymbol{\lambda}$ is completely passed, either on the position or on the time axis. This point is then marked, and only the remainder of the T' trajectory is checked. If another collision zone $\boldsymbol{\gamma}$ is found, the trajectory T' is adjusted but only starting from point i_p. To account for multiple obstacles, the two presented algorithms can be included in a recursive function, and stacks can be used to keep trace of the different starting points. This way, it is possible to remove the most recent constraint if an adjustment is impossible to do.

V. SIMULATION RESULTS

In this section, we give some simulation results of the integration of this speed planning method inside our local planning method. We also propose a comparison with our previous speed planning method presented in [16], later referred as the "stop-point method". In this method, each position of the candidate trajectory for the EV is compared with all predicted positions of the dynamic obstacles at the same time. If an intersection is found, the previous position on the EV's trajectory is marked as a stop point. Then, a new trajectory is generated and checked again, until the candidate trajectory avoids all collisions.

We use a reactive planning algorithm, where the local trajectory is composed of $N = 100$ points, and is recomputed every $200\,\mathrm{ms}$. There is no specific path planning here, the EV follows a predefined path, so that the results focus on speed planning. Speed and acceleration values used by the smoothing algorithm are presented in Table I. The other vehicles drive at the legal speed, and it is supposed that a

TABLE I: Acceleration and speed values used in the presented results

	Comfort	Maximum
Acceleration	$1\,\mathrm{m\,s^{-2}}$	$3\,\mathrm{m\,s^{-2}}$
Deceleration	$-2\,\mathrm{m\,s^{-2}}$	$-10\,\mathrm{m\,s^{-2}}$
Legal speed	NA	$10\,\mathrm{m\,s^{-1}}$

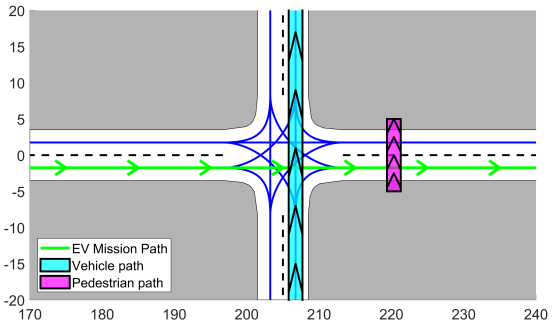


Fig. 5: Situations for the simulated scenarios

prediction of their motion for the next 5 seconds is available, with a discretization time step $\delta_t = 0.1\,\mathrm{s}$, leading to $M = 51$ points. The safety time to respect is $t_{safe} = 1\,\mathrm{s}$. The algorithm is implemented within MATLAB/Simulink, used in co-simulation with SCANeR Studio, a professional vehicle simulator. A basic PI controller is used to determine the accelerator pedal value, in order to track the speed profile defined by the planning algorithm. The control loop is executed every $40\,\mathrm{ms}$. Figure 5 illustrates the situation considered for these results, which is a four-branches crossing without particular regulation, where the EV comes from the left branch.

A. Scenario 1 : Yield to right

In the first scenario, one vehicle is coming from the bottom branch. The vehicles are initially placed such that they collide if the EV does not adapt its speed. Figure 6 presents the resulting speed profile on this scenario. This shows that with the proposed method, the EV anticipates more by starting to slow down earlier, and stabilizes on a constant speed until the other vehicle has left the path. When the maximum speeds are reached again, the proposed method has an advance of about $2.2\,\mathrm{m}$ compared to the previous one, so the maneuver was quicker by $0.22\,\mathrm{s}$. The minimum speed is higher and the maximum speed is reached sooner, meaning that less braking and acceleration are required.

B. Scenario 2 : Accelerate to cross before

In the second scenario, the EV is stopped before the crossing, and one vehicle is coming from the bottom branch, initially at a distance such that the EV can't cross with the comfort acceleration. Figure 7 show the resulting speed profiles in this situation. This result shows that the proposed acceleration adaptation makes the EV able to cross between before the other vehicle, compared to the previous method where it stops and passes after.

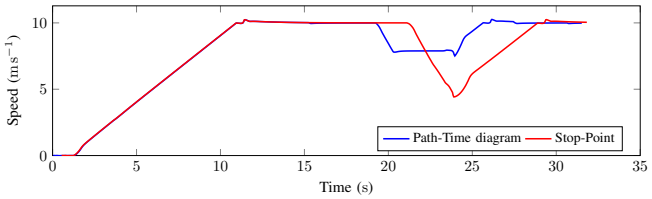
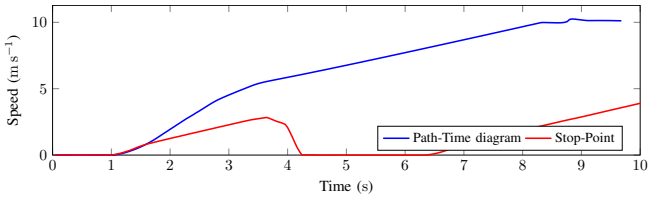
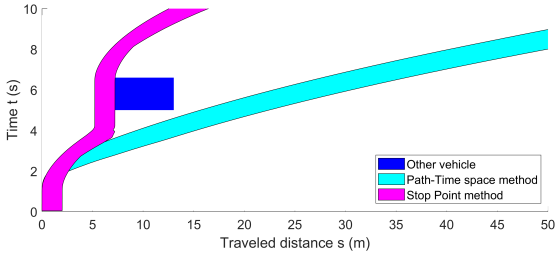


Fig. 6: Speed profiles obtained with one moving obstacle in the first scenario.



(a) Speed profiles



(b) Representation of the final trajectories in the path-time space

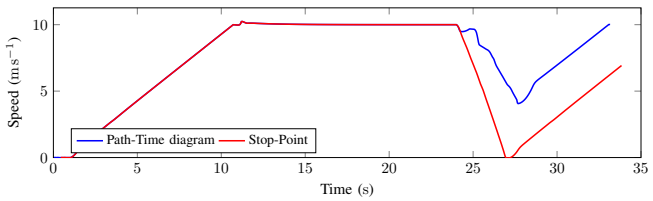
Fig. 7: Results of the simulation with two moving obstacles in the second scenario

C. Scenario 3 : Adapt the speed to pass between

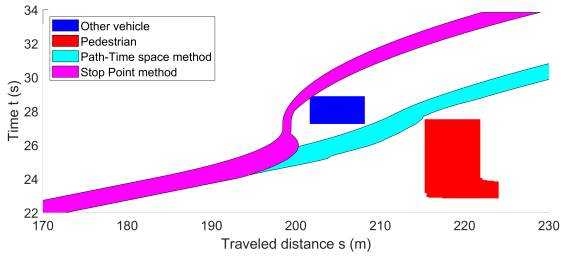
The third scenario uses a vehicle coming on the bottom branch and a pedestrian crossing the right branch after the crossing. The vehicle in the bottom lane is placed such that the EV is able to safely pass before when the pedestrian is not considered. Figure 8 gives the results of the execution of this scenario. As shown, when using the path-time method, the EV is able to find a valid trajectory passing before the vehicle while slowing down to let the pedestrian cross. When using the previous method, which does not consider this situation, the EV stops before the crossing to let the vehicle pass.

D. Processing time analysis

Figure 9 presents the distribution of the processing times for the proposed speed planning method, collected on a $2.70\,\mathrm{GHz}$ CPU during the execution of the algorithm. The measured time encloses all the planning process, namely path generation as a 5^{th} degree polynomial curve, speed constraints integration and speed adaptation to the obstacles for one trajectory. Only the processing times when there is at least one obstacle present in the collision matrix are represented in the given distribution. In this implementation, Algorithm 1 is used as presented in the paper without particular optimization. The results show that the resulting processing times are all below $20\,\mathrm{ms}$ with a median processing time around $5\,\mathrm{ms}$, which are already compatible with



(a) Speed profiles



(b) Representation of the final trajectories in the path-time space

Fig. 8: Results of the simulation in the third scenario

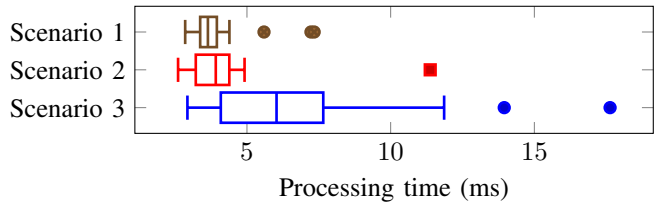


Fig. 9: Processing times distribution for the speed planning function when at least one obstacle is present in the visibility range of the vehicle

a real-time usage based on the control loop used in these simulations. Based on these results, we believe that with a dedicated implementation and appropriate optimization for Algorithm 1, faster processing times will be achieved, enabling the computation of several trajectories in the same planning step.

VI. CONCLUSIONS AND FUTURE WORKS

This article presents an algorithmic speed planning method for an autonomous vehicle, designed to handle moving obstacles. This is done by using the space-time diagram to represent spatio-temporal constraints on the path of the EV and to check the validity of candidate trajectories. Algorithms are proposed to generate trajectories to pass before or after one or more moving obstacles, and simulation results show that the obtained speed profiles are more efficient than with our previous method, and less conservative in constrained situations. Only interactions with other vehicles were presented here, but information from the infrastructure can also be integrated. For example, connected traffic lights could be represented as obstacles at a given position for a given time. This way, the speed could also be adjusted to avoid stopping if a traffic light can be passed by slowing down earlier.

Future works will focus on improving the robustness of this representation. Indeed, due to the discretization of the path-time space, the obstacles are over-estimated and can occupy more or less cells in the collision matrix depending on their position. This can lead to stability issues in the consecutively planned trajectories. This method will also be optimized to further evaluate its efficiency, and integrated in a realistic driving simulator to evaluate closed-loop results in an urban environment.

REFERENCES

[1] C. Katrakazas, M. Quddus, W.-H. Chen, and L. Deka, "Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions," *Transportation Research Part C: Emerging Technologies*, vol. 60, pp. 416–442, Nov. 2015.

[2] J. Ziegler, P. Bender, T. Dang, and C. Stiller, "Trajectory planning for Bertha - A local, continuous method," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*. MI, USA: IEEE, Jun. 2014, pp. 450–457.

[3] J. Ziegler and C. Stiller, "Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 1879–1884.

[4] M. McNaughton, C. Urmson, J. M. Dolan, and J.-W. Lee, "Motion planning for autonomous driving with a conformal spatiotemporal lattice," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 4889–4895.

[5] B. Li, Q. Kong, Y. Zhang, Z. Shao, Y. Wang, X. Peng, and D. Yan, "On-road trajectory planning with spatio-temporal rrt* and always-feasible quadratic program," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, 2020, pp. 942–947.

[6] Y. Zhang *et al.*, "Hybrid trajectory planning for autonomous driving in highly constrained environments," *IEEE Access*, vol. 6, pp. 32 800–32 819, 2018.

[7] A. Artuñedo, J. Villagra, and J. Godoy, "Jerk-limited time-optimal speed planning for arbitrary paths," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–15, 2021.

[8] X. Qian, I. Navarro, A. de La Fortelle, and F. Moutarde, "Motion planning for urban autonomous driving using bézier curves and mpc," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, 2016, pp. 826–833.

[9] A. Artuñedo, J. Villagra, and J. Godoy, "Real-time motion planning approach for automated driving in urban environments," *IEEE Access*, vol. 7, pp. 180 039–180 053, 2019.

[10] X. Qian, J. Gregoire, F. Moutarde, and A. De La Fortelle, "Priority-based coordination of autonomous and legacy vehicles at intersection," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2014, pp. 1166–1171.

[11] Ö. Ş. Taş and C. Stiller, "Limited visibility and uncertainty aware motion planning for automated driving," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 1171–1178.

[12] W. Lim, S. Lee, M. Sunwoo, and K. Jo, "Hybrid trajectory planning for autonomous driving in on-road dynamic scenarios," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–15, 2019.

[13] L. Wang, Z. Wu, J. Li, and C. Stiller, "Real-time safe stop trajectory planning via multidimensional hybrid a*-algorithm," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020, pp. 1–7.

[14] Y. Manolopoulos, Y. Theodoridis, and V. J. Tsotras, "Spatial Indexing Techniques," in *Encyclopedia of Database Systems*, L. Liu and M. T. Özsu, Eds. New York, NY: Springer New York, 2014, pp. 1–7.

[15] S. Lefèvre, D. Vasquez, and C. Laugier, "A survey on motion prediction and risk assessment for intelligent vehicles," *ROBOMECH journal*, vol. 1, no. 1, pp. 1–14, 2014.

[16] T. Duhautbout, R. Talj, V. Cherfaoui, F. Aioun, and F. Guillemard, "Generic Trajectory Planning Algorithm for Urban Autonomous Driving," in *2021 20th International Conference on Advanced Robotics (ICAR)*. Ljubljana, Slovenia: IEEE, Dec. 2021, pp. 607–612.