Prioritized multi-robot velocity planning for trajectory coordination of arbitrarily complex vehicle structures

Felix Keppler and Sebastian Wagner
Fraunhofer Institute for Transportation and Infrastructure Systems IVI, Dresden, Germany
{felix.keppler, sebastian.wagner}@ivi.fraunhofer.de

***Abstract*—Prioritized planning is a widely used technique to coordinate motions of multiple robots along fixed paths. Velocity profiles are planned sequentially avoiding previously computed trajectories represented as dynamic obstacles. To derive these, the vehicle structures are usually approximated with simple shapes, which fails for complex robots like automated truck-trailer combinations. A novel method to derive the spatiotemporal conflict zones directly from arbitrarily complex geometries swept along pre-planned paths is proposed and it is shown that the resulting obstacles in the distance-time-space allow velocity planning with precise space utilization and short execution times.**

I. Introduction and related work

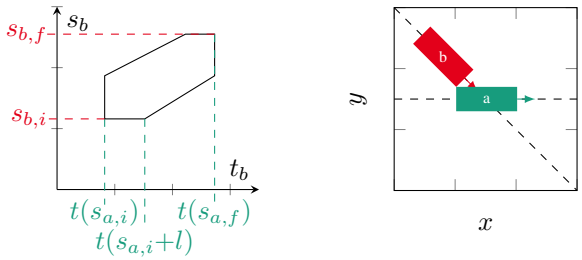
With automation being a key factor to raising productivity and efficiency in manufacturing and logistics, mobile robots are finding their way into various industrial applications. Examples include, but are not limited to, intralogistics such as warehouse automation, autonomous work machines in agriculture, and yard automation, where trucks are requested to drive autonomously in fenced areas like maritime ports or distribution centers.

When operating multiple robots in a shared workspace, one fundamental problem is to coordinate the individual motions to prevent mutual collisions. Once such coordinated trajectories are found, the robots can efficiently share a given area and not only increase efficiency and productivity, but also enhance safety, as the majority of accidents arise from human driving errors.

Planning coordinated motions of multiple agents avoiding both static obstacles and mutual collisions in a composite state space (in [1] referred to as *centralized* approach) is known to be computationally intractable [2]. Path-velocity-decomposition as proposed by Kant and Zucker [3] allows to reduce the problem complexity by first planning a path around the static obstacles and then planning velocity profiles avoiding other moving agents in a second step (*decoupled* approach).

In contrast to recent research in the field of autonomous driving ([4], [5]), this paper does not focus on independent coordinated maneuvers with the scope of a few seconds (e.g., overtaking, emergency braking) after which the robots continue with normal operation until the next situation arises. Instead, the aim is to efficiently schedule entire trajectories from start to target in continuous operation.

The path planning problem can be considered as solved even for robots with complex vehicle structures and multiple steered axles as shown in graph search based algorithms like TruckTrix® presented in [6]. Once the paths and their geometric intersections (conflict zones) are known, the problem of scheduling the robots can be solved with prioritized planning [7]. While the first robot moves freely along its path without considering other agents, subsequent tasks are assigned a lower priority and must consider all previously planned trajectories as dynamic obstacles. With the knowledge, *where* and *when* a collision would occur, the planner calculates a collision-free velocity profile in a distance-time-space.



(a) Dynamic obstacle in the s-t-diagram (b) Robot b following a in a single conflict zone

Fig. 1: Example situation in distance-time-space

Determining *where* a conflict zone is located along the path of a robot is straightforward for simple geometries, but gets complicated for robots with complex vehicle structures like a truck with one or more trailers. The commonly used concept of growing the obstacles so that the moving object can be shrunk to its reference point [8] fails particularly for such long, articulated vehicles. Other approaches like calculating the bounding segments of subpaths between conflict zones [9] could possibly be extended to conquer the latter but tend to get very complex and computationally expensive. Thus, a novel approach to tracking the spatiotemporal progression of the exact areas occupied by a robot along its path in a discretized grid map is proposed in this paper.

In the following, the concept of dynamic obstacles in a distance-time-space is explained in detail. Then, challenges arising for complex vehicle structures and how to overcome them with the presented new methodology are pointed out in Section II and III. In Section IV an adapted algorithm for planning a collision-free velocity profile in the distance-time-space is presented.

II. Problem Statement

A. Preliminaries

A variety of path planning algorithms exist which will not be discussed in detail here. A comprehensive overview can be found in [10]. To compute the path for a complex robot such as a truck with various trailers, the TruckTrix algorithm [6] is used over the course of this research. Given a static obstacle map and the geometric dimensions and kinematics of all vehicles, it returns a list of discrete steps including position, orientation and steering angles for all axles.

As the name implies, velocity planning in distance-time-space requires a vector of distances along the pre-planned path and returns a time vector of the same size according to the calculated velocity profile. Depending on the output of the path planner, the distance vector can be constructed with constant step lengths between the samples as in the presented case or using Euclidean distances between the coordinates forming the path.

Further, it is assumed that the prioritization scheme for the robots is implicitly given by the subsequent assignment of tasks.

B. The distance-time-space

The distance-time-space of a robot can be visualized in a so-called distance-time-diagram (s-t-diagram). From an ego-perspective of the respective robot, one axis denotes at which distance along its own path a collision could occur. Using the known velocity profiles of all higher priority trajectories, another axis measures the temporal distance to the conflict zones. A velocity profile avoiding the dynamic obstacles representing higher priority agents can then be obtained using linear programming or graph based search methods as further described in Section IV.

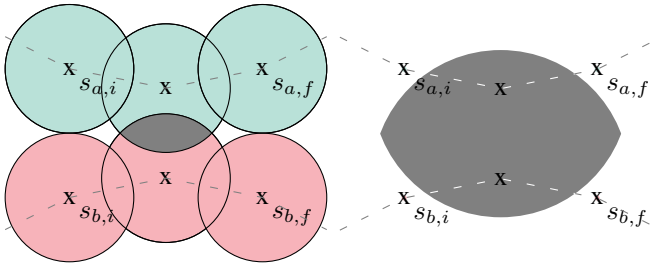
Figure 1 shows an example where robot b has to avoid robot a. The lower left edge of the obstacle $(t(s_{a,i}), s_{b,i})$ denotes where both robots are about to enter the shared conflict zone, at $(t(s_{a,f}), s_{b,f})$ both have left the conflict zone. Note that at $(t(s_{a,i+l}), s_{b,i})$, robot a moved far enough away from the edge of the conflict zone that robot b can already follow as shown in Figure 1b. Hence, the conflict zone is not blocked entirely if long shared path segments occur.

C. Conflict zones

In order to coordinate the motions of multiple agents, areas used by more than one robot need to be identified. For each pair of potentially conflicting robots, this is achieved by superposing the swept areas of the robots moving along their paths. Once the conflict zones are known, the distances at which a robot enters and exits a conflict zone along its path need to be calculated considering the robot's dimensions.

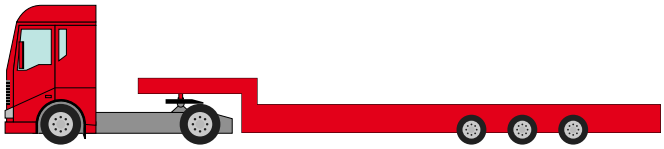
As the path consists of discrete steps, the goal is to find those steps where any part of the robot lies in the conflict zone for the first and for the last time.

Since a conflict can arise from overhanging parts of the robot, the reference point itself is not necessarily covered by the conflict zone as shown in the example in Figure 2a.

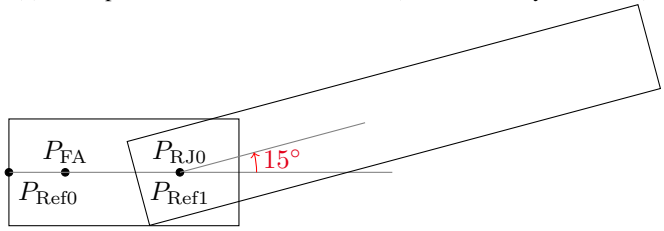


(a) Conflict arising from overhanging parts of circular robots (b) Shrunk robots in grown conflict zone

Fig. 2: Crucial steps at conflict zones



(a) Example truck-trailer combination (Source: HeavyGoods.net)



(b) Vehicle polygons derived from robot geometry and kinematics

Fig. 3: Geometric structure of a robot (entire combination) consisting of multiple vehicles (truck and trailer)

Thus, checking whether step positions lie in the conflict zone or whether linear connecting segments intersect with their boundaries is insufficient.

To overcome this problem, one could grow the conflict zones so that the moving objects can be shrunk to the reference points in analogy to [8]. As shown in Figure 2b, this works well for circular robots where the conflict zones are grown by the robots radii and thus cover all crucial step coordinates. For more complex vehicle structures, a growth factor reliably covering all areas swept when off-tracking in curved segments is not as straightforward to determine and can generally get very large. For instance, the truck-trailer combination in Figure 3a could be approximated with a large circle around the rear joint P_{RJ0} covering the entire geometry. As a result, the dynamic obstacles for the distance-time-space-planning are over-sized, which leads to significantly less optimal solutions.

Alternative approaches include taking into account the Euclidean distances between the conflict zone and the step coordinates. However, it remains unsolved which point in the conflict zone to consider for comparison. Further, this methodology may fail for situations, where a conflict zone arose from overhanging parts off-tracking in a curved segment and the robot passes the same area on a straight path segment before or afterwards. It could then happen

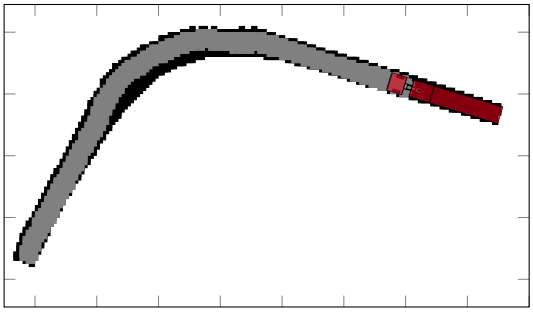


Fig. 4: Swept area/trace (black) of a vehicle combination (red) compared to the area covered by a swept circle (gray)

that reference points at which the robot was actually not in conflict are considered crucial as the distance is smaller. If the actually relevant steps are neglected, an uncontrolled collision could occur.

III. Approach

To overcome the issues indicated in II-C, the exact occupied areas can be identified according to the vehicle geometries and checked for interference with the conflict zones in each step. As this is computationally expensive, it is desirable to perform the calculations once per robot rather than within every step of the velocity planning algorithm.

A. Geometric occupancy and conflict zones

A shared workspace geometrically covering all individual paths is defined and discretized in $M \times N$ cells of predefined width and height. For simplicity, a rectangular shape is assumed – without loss of generality, as any arbitrary shape can be represented in a zero-padded rectangle. For each robot, a binary occupancy grid map $\mathbf{O} \in \{0,1\}^{M \times N}$ with elements $(o)_{i,j} \in \{0,1\}$ is created. This approach originates from common practice in digital image processing, where each of the $M \times N$ discrete points represents a pixel of an image.

For all steps $k = \{1, \ldots, L\}$ along the pre-computed path, the cells covered by the robot in step k are set to 1. This is straightforward for point-like and for circular robots, where the coordinates of each step along the path $(x, y)_k$ are mapped to the occupancy map and the latter is dilated by the robot's radius, if applicable. For car-like robots with rectangular box shape or more complex vehicle structures like a truck with one or more trailers, the actual swept area can differ significantly from the area covered by the path dilated by the robots' width as depicted in Figure 4. Therefore, it should rather be approximated by constructing all vehicle polygons of the robot using the absolute position P_{FA} and the orientation of the first vehicle and all relative positions and orientations for the subsequent parts at step k as shown in Figure 3b. In image processing terminology, this operation corresponds to fill-drawing the occupied area on the grid map.

Care should be taken when choosing a grid resolution higher than the sampling distance of the path. Not all cells covered in the continuous movement of the robot are then marked as such in the discrete calculation. Solutions include interpolating between the steps (linearly, using splines or considering the kinematic model) or introducing safety margins, e.g., by dilating the calculated areas.

For each pair of potentially conflicting robots, temporarily denoted as robot a and robot b, a binary conflict zone matrix $\mathbf{C} \in \{0,1\}^{M \times N}$ is obtained from the Hadamard product of $\mathbf{O}^A$ and $\mathbf{O}^B$ (corresponds to a logical AND operation):

$$\mathbf{C} = \mathbf{O}^A \circ \mathbf{O}^B = (o^A)_{i,j} \cdot (o^B)_{i,j} \tag{1}$$

B. Spatiotemporal occupancy

To track the occupancy throughout all steps, the concept of binary grid matrices is extended with the information *when* a given cell is occupied: $\tilde{\mathbf{O}} \in \{0,1\}^{M \times N \times L}$ with elements $(\tilde{o})_{i,j,k} \in \{0,1\}$. Rather than mapping the occupancy throughout all steps to a two-dimensional matrix, each layer of $\tilde{\mathbf{O}}$ stores the occupancy in step k.

C. Condensed entrance and exit matrices

In order to transform the spatiotemporal occupancy to the distance-time-space of a robot and visualize it as s-t-obstacles, the relevant information on entrance and exit of conflict zones can be condensed as follows:

The occupancy for each step k is calculated in a temporary matrix $\mathbf{O_k} \in \{0,1\}^{M \times N}$. A matrix $\mathbf{O}^i \in \mathbb{N}^{M \times N}$ is defined to store the step index where the robot enters a given cell for the first time, while another matrix $\mathbf{O}^f \in \mathbb{N}^{M \times N}$ stores where it occupies the cell for the last time. The elements of both matrices are initialized with 0.

All exit map cells occupied by the polygon configuration in step k are then set to k. The same applies for the entrance map matrix, but only for fields which are still equal to the initialization value 0, i.e. which have not yet been occupied:

$$(o^i)_{i,j} = k \mid (o_k)_{i,j} = 1 \wedge (o^i)_{i,j} = 0 \tag{2}$$

$$(o^f)_{i,j} = k \mid (o_k)_{i,j} = 1 \tag{3}$$

The following matrices show a simple example of a robot moving by one cell:

$$\mathbf{O}_1 = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \; \mathbf{O}_2 = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix},$$

$$\mathbf{O}^i = \begin{pmatrix} 1 & 1 & 2 \\ 0 & 0 & 0 \end{pmatrix}, \; \mathbf{O}^f = \begin{pmatrix} 1 & 2 & 2 \\ 0 & 0 & 0 \end{pmatrix}$$

Note that with this approach, a conflict zone cell is blocked by a robot from the very first to the very last occupancy. Thus, it is only feasible if it can be ensured that the path returned from the upstream path planner does not cover a cell twice or more in large temporal distance (e.g. driving a loop).

D. Derivation of s-t-obstacles

Dynamic obstacles in the distance-time-space (s-t-obstacles) can now be derived directly from the calculated matrices:

The conflict zone matrix $\mathbf{C}$ is decomposed in p diagonally connected regions to handle multiple interferences of the paths of one conflicting robot pair. For each region, the position of non-zero elements in the matrix is used to index

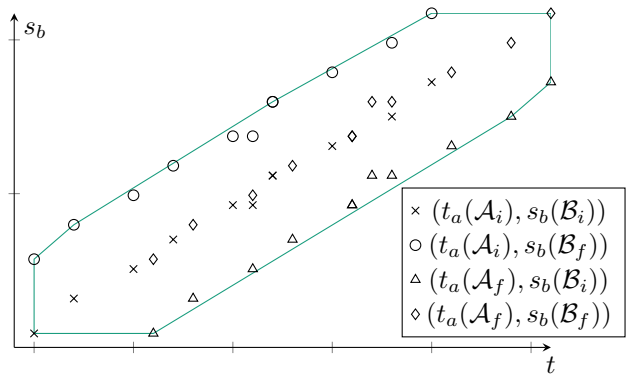


Fig. 5: s-t-obstacle constructed from swept geometries

the entrance matrix $\mathbf{O}^{A,i}$ resulting in a set of step indices $\mathcal{A}_i$. The step indices $\mathcal{A}_f$, $\mathcal{B}_i$ and $\mathcal{B}_f$ are found likewise.

Evaluating the time values of the higher priority robots' trajectory $t_a(k)$ at $k \in \mathcal{A}_i$ and the lower priority robots' distance values $s_b(k)$ at $k \in \mathcal{B}_i$ yields (t, s)-coordinates forming the lower left corner of an obstacle ($\times$). The set contains $(t(s_{a,i}), s_{b,i})$ where both robots enter the conflict zone as pointed out in Section II.

The upper right corner including $(t(s_{a,f}), s_{b,f})$ is formed by evaluating $t_a(k)$ at $k \in \mathcal{A}_f$ and $s_b(k)$ at $k \in \mathcal{B}_f$ ($\diamond$).

Intermediate coordinates ($\circ$, $\triangle$) originate from $t_a(k)$ at $k \in \mathcal{A}_i$ and $s_b(k)$ at $k \in \mathcal{B}_f$ and vice versa from $t_a(k)$ at $k \in \mathcal{A}_f$ and $s_b(k)$ at $k \in \mathcal{B}_i$ respectively. Eventually, an obstacle of the expected form is obtained by calculating the convex hull containing all points of the four sets as shown in Figure 5. Algorithm 1 shows the procedure in pseudocode.

As all existing tasks are *potentially conflicting* with a newly assigned task, the procedure is repeated to obtain the dynamic obstacles of all higher priority trajectories. When saved sequentially, the conflict matrices for the permutation of all conflict partners (a, b) are enumerated with $n = \frac{a^2+a}{2} - (a - b)$ as depicted in Table I:

TABLE I: Example to enumerate mutual interference of 4 robots

$a \backslash b$	0	1	2	3
0				
1	0			
2	1	2		
3	3	4	5	
4	6	7	8	9

Upon implementation in high-level programming languages, the occupancy information can be saved in a combined grid holding advanced data structures.

IV. Velocity Profile Planning

To plan a velocity profile around the dynamic obstacles in the distance-time-space, a state space $\langle s, v \rangle$ for the robot is considered which is discretized with small time steps Δt as proposed by Fraichard [11]. Allowed transitions between the states are restricted to maximum acceleration, maximum

Algorithm 1 Algorithm to generate s-t-obstacles

Input: previous tasks **tasks**, new task **b**
Output: obstacles in s-t-diagram for new task
Initialization
b.entrance, b.exit = zeros(M, N)
Sweep geometry for all steps
for step in b.steps **do**
  b.exit[step.covered_area] = step.id
  untouched = (b.exit[step.covered_area]!=0)
  b.entrance[untouched] = step.id
end for
Loop through all previous tasks
for a in tasks **do**
  conflictzones = (a.entrance!=0) AND (b.entrance!=0)
  Loop through all conflict zones
  for zone in conflictzones **do**
    coords.append(a.t[entrance[zone]], b.s[entrance[zone]])
    coords.append(a.t[exit[zone]], b.s[exit[zone]])
    coords.append(a.t[entrance[zone]], b.s[exit[zone]])
    coords.append(a.t[exit[zone]], b.s[entrance[zone]])
    b.obstacles.append(convex_hull(coords))
  end for
end for
return b.obstacles

deceleration or maintaining the current velocity with zero acceleration, which leads to a grid with $\Delta v = a_{\max} \Delta t$ and $\Delta s = \frac{1}{2} a_{\max} \Delta t^2$ [12]:

$$a \in \{-a_{\max}, 0, a_{\max}\} \tag{4}$$

$$v(t + \Delta t) = v(t) + a\Delta t \tag{5}$$

$$s(t + \Delta t) = s(t) + v(t)\Delta t + \frac{1}{2} a \Delta t^2 \tag{6}$$

Thus, from a given state $\langle s, v \rangle$, the following states can be reached:

$$\text{with } a_{\max}: \quad \langle s + (2\frac{v}{\Delta v} + 1)\Delta s, v + \Delta v \rangle \tag{7}$$

$$\text{with } a = 0: \quad \langle s + 2\frac{v}{\Delta v}\Delta s, v \rangle \tag{8}$$

$$\text{with } -a_{\max}: \langle s + (2\frac{v}{\Delta v} - 1)\Delta s, v - \Delta v \rangle \tag{9}$$

A. Prioritized breadth-first search

As presented by van den Berg and Overmars [12], the grid can be searched for collision-free combinations connecting start and target configurations using a prioritized breadth-first search. Starting from $\langle 0, 0 \rangle$, all possible transitions are explored in descending preference favoring maximum acceleration over zero acceleration or deceleration. First, states with $v > v_{\max}$ and $v < 0$ are rejected.

It is important to mention that this does not prevent the robot from driving backwards if it is intended in the path, as v is metered along the monotonically increasing distance s along the path regardless of the driving direction.

Next, it is checked whether the new state is covered by the convex hull of any potential collision partner which was

derived in Section III-D. Upon rejection, a less preferable transition is explored. Otherwise, the current state is updated including a backlink reference to the previous state. If all transitions fail, the algorithm recursively steps back and explores all unvisited transitions from the previous state. Algorithm 2 shows the overall procedure in pseudocode. With the continue statement, the rest of the code inside a loop is skipped and the algorithm proceeds with the next iteration.

Algorithm 2 Prioritized breadth-first search in distance-time-space

Input: previous tasks **tasks**, new task **b**
Output: collision-free velocity profile for new task

```
stack.append(⟨s = 0, v = 0, t = 0⟩)
while stack not empty do
    current = stack.pop()
    if v > v_max then
        continue
    end if
    if v < 0 then
        break to ⟨s, 0, t⟩
    end if
    for all collision partners do
        for all s-t-obstacles with collision partner do
            if ⟨s, v, t⟩ in obstacle then
                continue
            end if
        end for
    end for
    if s > s_end then
        return backtracking(⟨s, v, t⟩)
    end if
    stack.append(decelerate)
    stack.append(maintain velocity)
    stack.append(accelerate)
end while
```

1) Stopping at target: In normal operation, the robots should start their missions with $v = 0$ and eventually reach their target in complete standstill. The first requirement is fulfilled by simply initializing the first state with $v = 0$. To slow down at the end and reach $s = s_{\text{end}}$ with $v = 0$, the graph search algorithm needs to be extended as follows: it is checked whether s is greater than $s_{\text{end}} - \Delta s$ in every loop iteration. Once triggered, the condition forces the search algorithm to recursively explore states with reduced velocities – the same way as when colliding with obstacles – until $v < \Delta v$. It is then safe to append a final step to state $\langle s = s_{\text{end}}, v = 0 \rangle$.

The final velocity profile is obtained by linking all states using their backpointer.

2) Reversing maneuvers: In order to reach a position in a desired orientation, it may be necessary for the robot to drive backwards. A common example is parking a truck's trailer at a cargo loading gate. TruckTrix is capable of planning paths including such reversing segments. In the distance-time-space, moving from one step to the next is considered

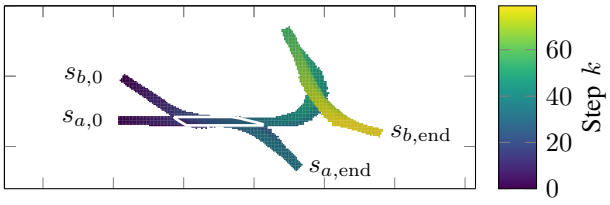
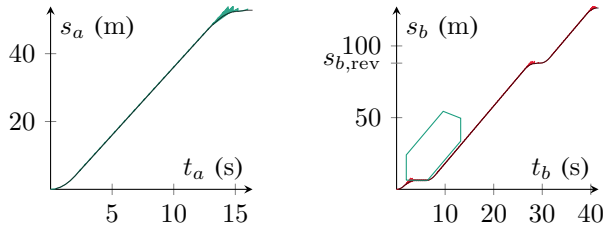


Fig. 6: Superposed exit maps with highlighted conflict zone



(a) Prioritized robot a (b) Subsequent robot b

Fig. 7: Coordinated velocity profiles for two robots

following the path and the distance vector is monotonically increasing regardless of the driving direction. Therefore – apart from different maximum speed and acceleration values, if applicable – the procedure of planning the velocity profile remains basically the same. However, it is necessary to ensure that the robot stops to complete standstill at each turning point. The graph search then resumes normally allowing the robot to accelerate along the path again, but in opposite direction.

V. Results

Figure 6 shows an example where a prioritized truck-trailer combination a moves freely from west to south east with the velocity profile shown in Figure 7a. The velocity profile calculated for another agent b, sharing a part of the track before performing a reversing maneuver, is shown in Figure 7b. All states explored by the graph search algorithm are plotted in the respective color in the background. It is clearly visible that robot b waits at the entrance of the conflict zone before following robot a. Further, at $s_{b,\text{rev}}$, it stops to complete standstill in order to reverse.

VI. System Integration

The algorithm can be integrated in a toolchain to execute coordinated maneuvers upon user requests like in the AutoTruck project funded by the German Federal Ministry for Economic Affairs and Energy, where trucks drive autonomously on fenced areas like distribution centers. All robots are registered in a central control unit and individual tasks can be assigned providing a starting time as well as a start and a target configuration, e.g., using a web interface. First, a path avoiding static obstacles is planned by an algorithm like TruckTrix. The resulting list of steps and the information about the vehicle structures and kinematics serve as input for the algorithm described in this paper. To transfer data between the services and planners, a database holding intermediate results serialized as JSON strings can be

used. The coordinated trajectory is then passed to the robot and executed using either a combination of a path tracking controller like the one presented in [13] and a longitudinal (speed) controller to match the requested velocity profile or using a combined controller like the one presented by Kanayama et al. [14]. Minor deviations from the planned trajectory, e.g., due to errors in the calculation of the distance vector in curved segments, are compensated in the control loop as well. While the communication between the central planning software and the robots could be implemented with frameworks such as Robot Operating System (ROS) [15], the control algorithms are likely to be implemented on dedicated electronic control units for reliable real-time operation. This also allows to integrate low-level emergency stop systems relying on robust, short-range environment perception sensors.

VII. Limitations

The presented approach requires all robots to share the same workspace definition in order to determine geometric intersections, which is usually reasonable for shared areas where coordinated multi-robot trajectory planning is needed. In scenarios where multiple workspaces are spread far apart but movements still need to be coordinated e.g. for commuting traffic between them, one could think of separate planners with handover management.

The total execution time for a task is not known a priori but determined at the end of the velocity planning as it depends on the configuration space varying with the starting time. Therefore, requests to reach the target at a given time can only be realized based on approximations.

As comprehensively discussed in [16], the infrastructure in which the robots move has to be *well-formed*. That is, robots parking at start and destination positions shall not hinder other robots moving along their paths, as they would endlessly wait for clearance.

When the prioritization is not implicitly given by the subsequent assignment of tasks, methods to find good priority sequences are needed.

VIII. Conclusion and future work

Using the presented approach, coordinated velocity profiles can be found for robots with arbitrarily complex vehicle structures moving along fixed paths. Challenges arising when using geometric approximations are overcome by analyzing the precise spatiotemporal progression of areas occupied by the robots in a discretized workspace representation. As a result, the available area is utilized to the maximum extent and total execution times are reduced to the minimum within the limits of prioritized planning.

The approach can not only be used for planning motions in a two-dimensional plane. By considering swept volumes instead of swept areas, it can be extended for trajectory planning in three dimensions. Conflicts between robots then form conflict volumes holding spatiotemporal information, which can be condensed to knowledge on entrance and exit along the fixed paths.

In real-world applications, disturbances in the exact execution of the planned motions are likely, e.g., due to collision avoidance stops with passive agents (humans, non-automated traffic). Especially for high priority robots, a deviation from the planned trajectory can quickly affect others and leave the system prone to deadlocks. Thus, methods to handle interruptions with minimal impact on other planned tasks are needed and are a key focus of upcoming research.

The graph search algorithm to find suitable velocity profiles can be further optimized for enhanced performance. For comfort and energy saving purposes, less extreme acceleration values and velocities adapted to the curvature could be considered.

References

[1] J.-C. Latombe, *Robot Motion Planning*, ser. The Springer International Series in Engineering and Computer Science. Springer US, 1991.

[2] J. Hopcroft, J. Schwartz, and M. Sharir, "On the Complexity of Motion Planning for Multiple Independent Objects; PSPACE- Hardness of the "Warehouseman's Problem"," *The International Journal of Robotics Research*, vol. 3, no. 4, pp. 76–88, Dec. 1984.

[3] K. Kant and S. W. Zucker, "Toward Efficient Trajectory Planning: The Path-Velocity Decomposition," *The International Journal of Robotics Research*, vol. 5, no. 3, pp. 72–89, Sep. 1986.

[4] M. Naumann and C. Stiller, "Towards Cooperative Motion Planning for Automated Vehicles in Mixed Traffic," *arXiv:1708.06962 [cs]*, Aug. 2017.

[5] J. Eilbrecht and O. Stursberg, "Cooperative driving using a hierarchy of mixed-integer programming and tracking control," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2017, pp. 673–678.

[6] S. Beyersdorfer and S. Wagner, "Novel model based path planning for multi-axle steered heavy load vehicles," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, Oct. 2013, pp. 424–429.

[7] M. Erdmann and T. Lozano-Pérez, "On multiple moving objects," *Algorithmica*, vol. 2, no. 1, p. 477, Nov. 1987.

[8] T. Lozano-Perez and M. A. Wesley, "An Algorithm for Planning Collison-Free Paths Among Polyhedral Obstacles," *Communications of the ACM*, vol. 22, 1979.

[9] T. Simeon, S. Leroy, and J. P. Laumond, "A collision checker for car-like robots coordination," in *1998 IEEE International Conference on Robotics and Automation*, vol. 1, May 1998, pp. 46–51.

[10] D. Gonzalez, J. Perez, V. Milanes, and F. Nashashibi, "A Review of Motion Planning Techniques for Automated Vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, Apr. 2016.

[11] T. Fraichard, "Trajectory Planning in a Dynamic Workspace: A 'State-Time Space' Approach," *Advanced Robotics*, vol. 13 (1), pp. 74–94, 1994.

[12] J. van den Berg and M. Overmars, "Kinodynamic motion planning on roadmaps in dynamic environments," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. San Diego, CA, USA: IEEE, Oct. 2007, pp. 4253–4258.

[13] J. K. Kolb, G. Nitzsche, and S. Wagner, "A simple yet efficient Path Tracking Controller for Autonomous Trucks," in *IFAC Proceedings Volumes*, Gdansk, Poland, 2019, p. 6.

[14] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi, "A stable tracking control method for an autonomous mobile robot," in *, IEEE International Conference on Robotics and Automation Proceedings*, May 1990, pp. 384–389 vol.1.

[15] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[16] M. Cap, P. Novak, A. Kleiner, and M. Selecky, "Prioritized Planning Algorithms for Trajectory Coordination of Multiple Mobile Robots," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 3, pp. 835–849, Jul. 2015.