Fast collision detection through bounding volume hierarchies in workspace-time space for sampling-based motion planners

Ulrich Schwesinger[1], Roland Siegwart[1] and Paul Furgale[1]

***Abstract*— This paper presents a fast collision-detection method for sampling-based motion planners based on bounding volume hierarchies in workspace-time space. By introducing time as an additional dimension to the robot's workspace, the method is able to quickly evaluate time-indexed candidate trajectories for collision with the known future motions of other agents. The approach makes no assumptions on the shape of the objects and is able to handle arbitrary motions. We highlight implementation details regarding the application of the collision detection technique within an online planning framework for automated driving. Furthermore, we give detailed profiling information to show the capability for real-time operation.**

***Index Terms*— Collision detection, workspace-time space, bounding volume hierarchy, axis-aligned bounding box tree**

I. Introduction

For a large number of motion planners, collision detection is the most time-consuming operation in its planning cycle. Both sampling-based planners like the Rapidly Exploring Random Tree (RRT) algorithm [1] and RRT* [2], as well as state-lattice approaches [3] have to evaluate numerous explicitly constructed paths for collision. The majority of implementations of sampling-based motion planners operate in the time-agnostic configuration- or state-space of the robot [1], [3]. Operation in dynamic environments is tackled through frequent replanning. However, planning in dynamic environments eventually necessitates the detection of collisions of time-parametrized trajectories. Objects are not considered to be in collision if they occupy the same space at a different point in time. This problem can be transformed into the problem of static collision detection in configuration-time space $\mathcal{CT}$ [4], [5] by introducing time as an additional dimension.

Valuable work was done applying sampling-based motion planning techniques in configuration-time space directly [6], [7]. In [7], Hsu et al. adopt a Probabilistic Roadmap Planner in state-time space to plan for kinodynamic systems among moving obstacles. Though mentioning the existence of more sophisticated collision detection techniques, this task is performed with a naive approach (multiple interference test) of checking every robot pose against every obstacle. Van den Berg et al. approach the planning problem in a dynamic environment by precomputing a roadmap of motions avoiding static obstacles. Later, they utilize a two-dimensional state-time space to find a collision-free trajectory on the roadmap

*The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013, Challenge 2, Cognitive Systems, Interaction, Robotics, under grant agreement No 269916, V-Charge (http://www.v-charge.eu).

[1]Autonomous Systems Lab, ETH Zurich, Tannenstrasse 3, 8092 Zurich Corresponding author: ulrich.schwesinger@mavt.ethz.ch

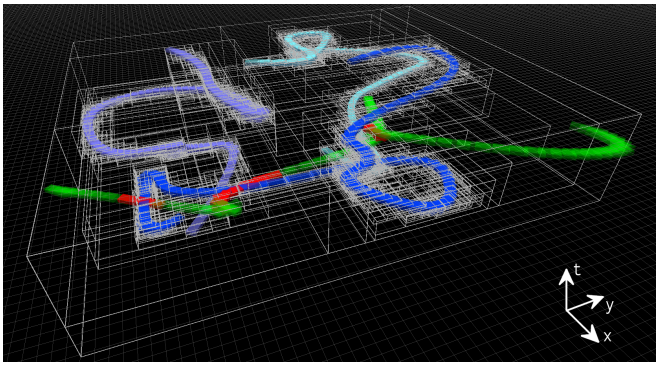


Fig. 1: Visualization of an axis aligned bounding box tree in workspace-time space for three obstacle trajectories generated by a random walk (blue shades). The ego trajectory (green/red) can be efficiently checked for collision with the help of the tree structure (wireframes).

[8]. Collision detection is done with the help of a regular grid in the two-dimensional state-time space. Lawitzky et al. presented an algebraic collision detection algorithm for automotive applications in highway scenarios [9]. It assumes trajectories for rectangular objects given as quintic polynomials and neglects the orientation of the vehicles. Given these restrictions, it provides runtimes around $5\,\mu s$ for a collision test of two individual trajectories. As the authors mention, the negligence of the object's orientation is acceptable for highspeed driving in certain environments such as highways, but intolerable for most other applications which clearly limits the range of application of this method. In [10], Ziegler et al. target planning for a car-like vehicle along a reference path. A set of candidate trajectories is generated and tested for collision with moving objects with a method presented in [11]. This approach however necessitates a circular approximation of all agents' shapes, which might result in poor navigation performance in tightly occupied workspaces (imagine a car navigating through a corridor framed by sideways parked cars). Closest to our approach comes the work of Ferguson et al. conducted within the context of Carnegie Mellon's entry in the DARPA Urban Challenge (team Boss) [12]. Collisions are detected via a three-stage hierarchical sequence of checks. In order to check two trajectories, they fist perform a test with two axis-aligned bounding boxes in a two-dimensional (2D) workspace. In case the axis-aligned bounding boxes do overlap, a multiple interference test with 2D bounding circles is performed. Afterwards, collision tests with oriented bounding boxes are performed where bounding circles overlap. Though the

initial axis-aligned bounding box (AABB) test speeds up computation for objects that are far apart, time consuming multiple interference tests are conducted in case objects get closer. Our approach is a consistent continuation of the idea of hierarchical collision tests.

Within this paper we report on the implementation of an AABB data structure in workspace-time space for fast collision detection of candidate trajectories. This collision detection approach allows for fast trajectory collision queries with no constraints on the objects' shapes or motions. The approach has been implemented in the sampling-based online motion planning framework presented in [13] to prove its real-time capabilities within an automotive application.

The Axis-Aligned Bounding Box Tree belongs to the class of bounding volume hierarchy (BVH) approaches. They are widely used in computer graphics and physical simulation [14], [15] in the so-called *broad phase* of the collision detection. They aim at reducing the number of exact collision computations (*narrow phase*) between collision shapes by pruning most candidates through a hierarchically ordered sequence of simpler queries. BVH approaches build a tree of bounding volumes on an object level. Bounding volumes proposed so far are bounding spheres, axis-aligned bounding boxes (AABB), oriented bounding boxes [16] or discrete-orientation polytypes [17]. For applications in three-dimensional (3D) workspaces, implementations are readily available in well-maintained software packages like the Open Dynamics Engine [14], the Bullet Physics Library [15] or the recent Flexible Collision Library (FCL) [18].
Closely related to BVH approaches, spatial partitioning techniques like KD-Trees, Quadtrees, Octrees and voxel- or grid-based collision detection approaches apply the partitioning in space rather than on the level of objects. The choice between spatial and object-oriented partitioning techniques strongly depends on the size of the environment under consideration and the amount of obstacles occupying it. In this paper, we target an online automotive receding-horizon planning framework [13] where a moderate amount of objects has to be considered within the vicinity of the vehicle. In this case, an object-based partitioning scheme is superior.

In his pioneering work [19]–[21], S. Cameron introduces three approaches to the collision detection problem of moving rigid objects. Firstly "Multiple Interference Detection", which tests the shapes for collision at subsequent fixed instances in time. Secondly the "Swept Volume" approach that aims at computing volumes in the time-agnostic workspace that are swept by the objects over time and testing the volumes for intersection. However it may happen that the swept volumes intersect when no collision actually takes place due to different time instances at which the objects occupy the same space. And thirdly "Extrusion", which is the process of creating a collision object in the workspace-time space that we follow within this paper.

II. Contribution

Within this paper we report on the implementation of an AABB data structure in workspace-time space for fast collision detection of candidate trajectories inside sampling-based motion planning frameworks for dynamic environments. Existing online applicable techniques make either restricting assumptions or simplifications on object's motions [9] or the object's shape [10]. Our approach provides fast responses to collision queries with almost no restrictions on particular motions or shapes of the objects and is thus well suited for application in a broad variety of planning environments. Though the idea of using partitioning techniques in workspace-time is not new [19]–[21], we are not aware of any application within an online motion planning framework, especially within the autonomous driving domain. Already conducted work on configuration-time space collision detection either focuses on the principles and lacks profiling information [20]–[22] or was performed with complex collision shapes (e.g. manipulators) [18], [23] or nowadays outdated hardware [23]. Thus the results are not directly transferable to the motion planning task at hand. This paper highlights various implementation considerations and provides detailed runtime statistics to facilitate decisions regarding the usage of this approach within other online planners operating in dynamic environments.

III. Approach

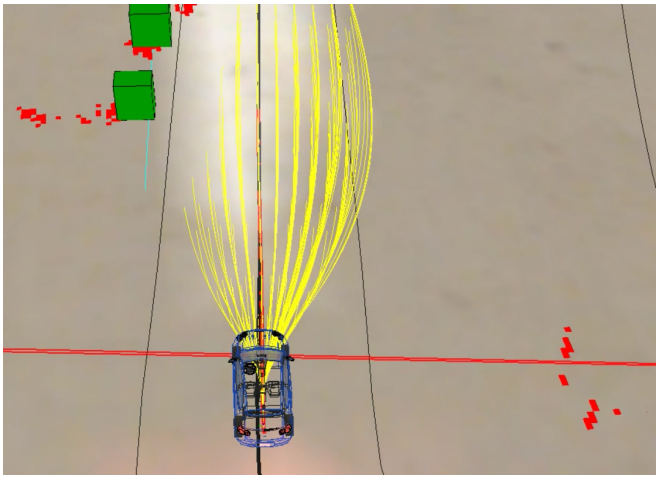


Fig. 2: Candidate trajectories (yellow) of motion planner to be checked for collision with dynamic objects (green) over time.

In this work, we consider the task of detecting collisions of candidate trajectories for a single robot operating in a two-dimensional workspace, $\mathcal{W}_2 = \mathbb{R}^2$. Candidate trajectories have to be checked for collision against movement predictions of objects over time. Configurations of objects are only considered in collision in case they collide at a certain time. An exemplary setup obtained with the motion planning framework presented in [13] is illustrated in Figure 2. The collision tests are accelerated with the help of a bounding volume hierarchy data structure. Note that the application is not limited to 2D workspaces. Bounding volumes may also be defined in higher dimensions.

First, we would like to recap the notion of a workspace and a configuration space and explain the domain in which

the AABB tree is constructed. A rigid body that translates and rotates within $\mathcal{W}_2$ has a three-dimensional configuration space $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1$. Obstacles moving within this workspace create obstacle regions at certain time instances, imposing a configuration-time space obstacle, $\mathcal{CT}_{obs}$. Explicitly constructing $\mathcal{CT}_{obs}$ is difficult and time-consuming for arbitrarily shaped robots, since it requires a description of the obstacle region with respect to all possible rotations. Therefore we explicitly construct the workspace-time obstacle region, $\mathcal{WT}_{obs}$, and incorporate the rotation at query time.

The volume occupied in $\mathcal{WT}$ by an agent moving along its trajectory forms a collision object. For general motions this collision object is usually non-convex. We define a trajectory for agent i, $\mathbf{T}^i$, as a time-ordered sequence of tuples of time, t_{k_i}, and configuration, $\mathbf{c}_{k_i}$: $\mathbf{T}^i = [(\mathbf{c}_{1_i}, t_{1_i}), (\mathbf{c}_{2_i}, t_{2_i}), \ldots, (\mathbf{c}_{K_i}, t_{K_i})]$. This restriction to discretized movements stems from the general difficulty to compute intersections for arbitrary analytic curves. The union of all volumes of the N agents (excluding the agent acquiring the plan),

$$\mathcal{WT}_{obs} = \bigcup_{i=1}^{N} \underbrace{\bigcup_{k=1}^{K_i} \mathbf{T}_k^i}_{\substack{\text{workspace-time} \\ \text{collision object} \\ \text{of agent } i}} , \tag{1}$$

defines the obstacle region $\mathcal{WT}_{obs}$.

In a naive implementation (multiple interference test), the ego configuration of a candidate trajectory must be checked against the configuration of each agent at each discretized time, resulting in $O(KN)$ collision tests, where K is the number of configurations in the trajectory query, and N is the number of obstacle trajectories. By using a BVH data structure during broad phase collision detection, the number of required exact checks (narrow phase) between collision objects can be greatly reduced. Therefore, each pose of each collision object is inserted into the AABB tree (see Figure 1). The root bounding box of the AABB tree fully contains all collision objects. Upon a query the tree is traversed in depth solely where the bounding boxes of the query and the tree intersect. Should the bounding box within the tree be a leaf node, the collision detection goes into a narrow phase step in order to test the associated collision shapes for overlap (e.g. box-to-box or sphere-to-sphere). Note that narrow phase collision algorithms exist for numerous collision shapes. Within this paper we exclusively illustrate AABB trees for oriented box shapes.

In order to integrate this collision detection technique into an online planning framework, there are various considerations within this approach that require further attention.

A. Isolated Configuration Queries Against the AABB Tree

The most simple application is isolated configuration queries. Each of the ego configurations, $\mathbf{T}_k^{\text{ego}}$, for each time instance t_k is checked for collision against the union of collision objects over time, $\mathcal{WT}_{obs}$, individually. Though being inferior to the "Tree vs. Tree" query (presented below) in terms of computation time, this approach avoids the non-negligible overhead of computing the BVH data structure for each trajectory in the set of queries. In a usual setting, the number of trajectories in the queries is much larger than the number of obstacles which could prevent this computation to be realizable with real-time constraints for a planner in which the candidate trajectories are generated on-the-fly.

Another fairly obvious choice is to abort the collision test for a trajectory, as soon as a collision is detected. Within the context of this paper, we call this the *early-exit option*. Without prior knowledge of the index of the collision within the pose sequence a linear traversal of the sequence is as good as any other. Applying a biased ordering (based on collision tests in previous planning cycles) can potentially turn out beneficial.

B. "Tree vs. Tree" Queries

Another option is to utilize a second BVH data structure for the trajectory in the query as displayed in Figure 3. In the broad phase of collision detection, two trees are tested against each other. This is done by traversing both trees top to bottom. This improves runtime performance with respect to the query, but comes at the additional cost of creating the BVH data structure for every candidate trajectory. For planners with predefined motion primitives such us e.g. state-lattice planners or randomized trees with a fixed set of primitives, the tree structures for each candidate trajectory can be precomputed, further reducing the spare computation time for online operation. We provide profiling information in Section IV to help to decide whether this operation is beneficial in case that precomputation is not possible. This is for example the case for planners presented in [13] or [10], where the construction of primitives depends on the shape of a reference path.

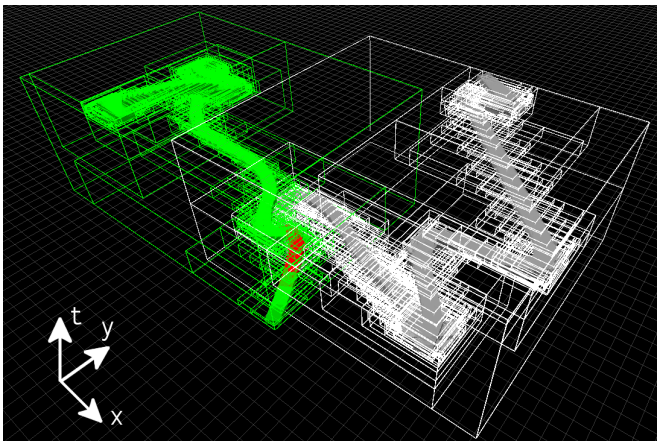
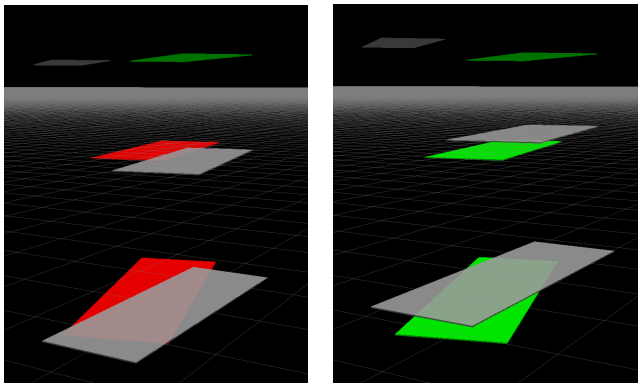


Fig. 3: Collision test with two AABB tree data structures. The white tree contains all collision objects, the green tree all query configurations of one trajectory. Collisions can be detected faster through a more efficient resolution of the candidate points of collision in workspace-time space.



(a) (b)

Fig. 4: Problems arising from discretized trajectories. Grey boxes: obstacle shapes, red/green boxes: query shapes detected as colliding/not colliding, a) Equal time discretization for ego and obstacle trajectories. Collisions are detected correctly. b) Different time-discretization for ego and obstacle trajectories causes missed collision detections.

C. Time Discretization of Motions and Temporal Extent of Primitives

By default, a collision object does not have a temporal extent, e.g. a discretized trajectory of rectangular collision objects consists of a sequence of flat rectangular objects. Special attention has to be paid to apply the same time discretization for the obstacles' motion predictions and the candidate trajectories. Collisions will certainly be missed unless the positions in the time dimension are equally spaced as illustrated in Figure 4.

Though applying an equal time discretization eliminates the danger of missed collisions for two-dimensional objects, there are two arguments for introducing 3D primitives in $\mathcal{WT}$.

- Enforcing a *minimum time-gap* between trajectories becomes straight forward by setting an extent of the collision shapes in the time dimension. The minimum time-gap setting will label configurations as colliding if they occupy the same space close in time. This is a desirable behavior for many motion planning tasks—above all in automotive applications—as it results in less aggressive behavior of the robot.
- It allows the use of highly optimized software libraries such as [14], [15], [18] without modification as these libraries are designed for collision detection between three-dimensional objects.

D. Convex Hull Representation of Trajectories

The usage of discretized motions gives rise to potentially missed collisions depending on the coarseness of the time-resolution. A way to overcome this issue is to compute the convex hull of every subsequent tuple of configurations within a trajectory. The convex hull computation should be performed for both the collision objects and the query trajectories. The workspace-time obstacle of agent i is represented as

$$\mathcal{WT}_{obs,i} = \bigcup_{j=2}^{N} \text{convex\_hull}\left(\mathbf{T}^i_{j-1}, \mathbf{T}^i_j\right) , \qquad (2)$$

resulting in a sequence of $N-1$ convex hulls. Again, each of the convex hull shapes can be inserted into the AABB tree for accelerated collision queries. Figure 5 depicts the convex hull representations and the resulting AABB tree for three obstacles and one candidate ego trajectory, each coarsely discretized.

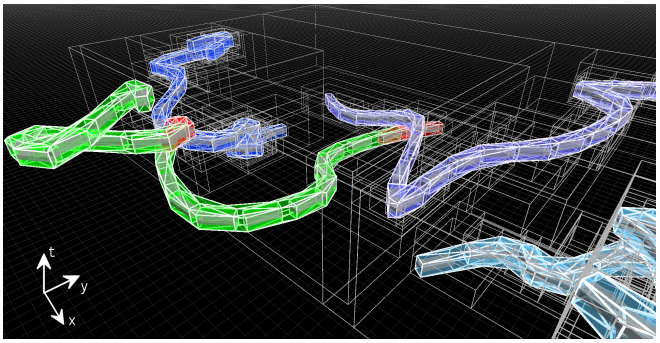


Fig. 5: Convex hull representations in workspace-time space eliminate the possibility of collisions missed due to discretization and enable obstacle inflation in the temporal domain. This figure illustrates three obstacle trajectories (blue shades) and one ego trajectory (green/red). Collision queries between convex hull shapes can be accelerated with the AABB tree data structure (wireframe boxes).

IV. Results

Within this section we provide runtime results in order to give implementation guidance for the interested reader. The computations were performed on a single core of an Intel Core i7@2.67 GHz processor with 32 KiB/256 KiB/4096 KiB of L1/L2/L3 cache. Profiling was done measuring CPU cycles. For the sake of a more intuitive interpretation, CPU cycles are converted to timings ($t_{\text{comp}} = {}^{N_{\text{cycles}}}/_{2.67 \cdot 1E9}$). To account for disturbances (such as multitask handling) during the profiling step, each timing measurement is generated by the minimum CPU cycle count of multiple measurements. We utilized the AABB tree data structure from Willow Garage's Flexible Collision Library (status May 21st, 2014). The code was compiled with gcc 4.6.3. Note that a slightly more efficient implementation is possible via a specialized narrow phase collision detection algorithm accounting for the fact that roll and pitch angles of the workspace-time space objects are always equal to zero for the 2D workspace at hand. In our experiments we used the standard implementation of the Flexible Collision Library library without the specialized narrow phase routine.

Since runtimes are highly dependant on the proximity and collision state of the trajectories, these are randomly generated. In order to create a randomized trajectory, a time vector with equidistant spacing is generated and a corresponding sequence of configurations is generated via a random walk.

Figure 6 shows the dependency between the number of obstacle trajectories and the runtime for three different approaches. The figure gives the runtimes of the "Isolated Configurations" (section III-A), "Tree vs. Tree" (section III-B) compared to the naive "Multiple Interference Test" approach. To provide a fair comparison, the same shape primitives and the same narrow phase collision detection implementation were used. Due to our focus on automotive applications, we use 3D boxes which approximate the shape of most cars fairly well. For a varying amount of obstacle trajectories from 1 to 30, a total of 1000 random scenes (a scene is comprised of the ego trajectory and all obstacles trajectories) were generated and evaluated for collision with each approach. The timing information is separated into the two cases in which the ego trajectory is in collision with at least one obstacle and in which the ego trajectory is collision-free.

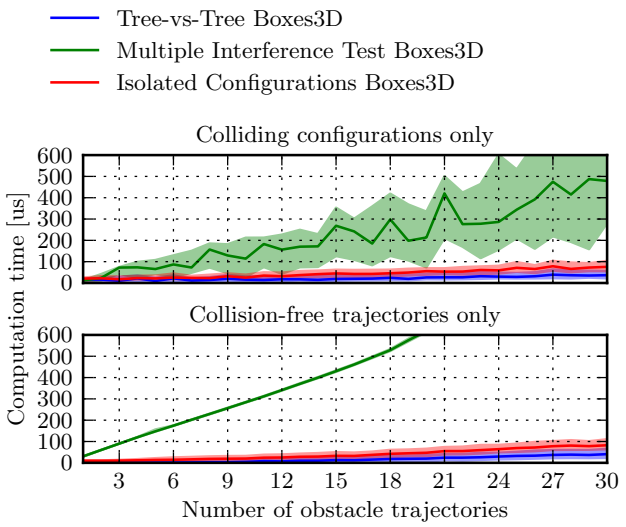


Fig. 6: Median (solid lines) and 0.25/0.75-percentiles (filled area) of runtimes given in µs for collision test of one query trajectory with 150 poses with respect to number of obstacle trajectories for three different approaches. In both cases the tree approaches are superior to the naive "Multiple Interference Test" approach.

The case of a collision-free constellation represents the worst-case for the naive "Multiple Interference Test" approach. All configurations have to be tested in order to rule out a potential collision. As expected, this approach shows a linear dependency with respect to the number of obstacles. Runtimes for the approaches utilizing the accelerating AABB tree data structure likewise grow due to the increasing depth of the tree, yet outperform the naive approach more and more as the number of obstacle trajectories increases. The "Tree vs. Tree" approach—more effectively pruning the potential points of collisions in workspace-time space—is superior to the "Isolated Configurations" approach.
The performance difference shrinks when collisions start to occur due to the random point of occurrence within the ego trajectory. In case the collision occurs early, there is only a small number of narrow phase checks to be performed in the "Multiple Interference Test" approach. Depending on the depth of the tree, this is often faster than the sum of AABB tests that have to be performed upon tree traversal. Nonetheless the tree approaches tend to provide lower computation times in average.

For better visibility, Figure 6 is again displayed in Figure 7 with different limits in the computation time axis. For the no-collision case (which has to be taken as a reference regarding wort-case computation time compared to the naive "Multiple Interference Test" approach), the "Tree vs. Tree" approach provides remarkable runtimes.

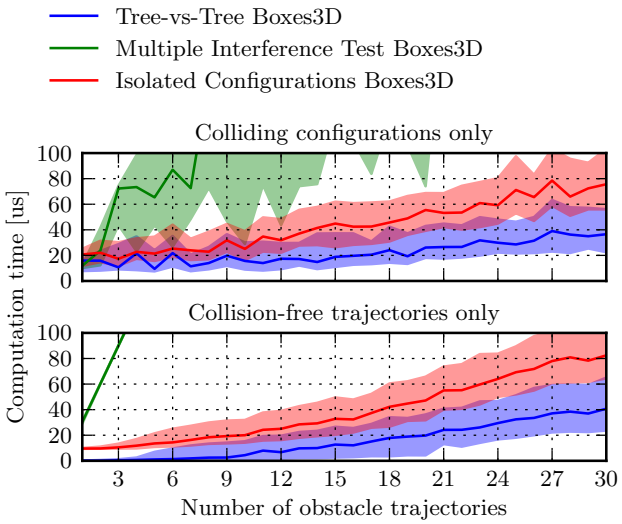


Fig. 7: Zoomed plot of Figure 6

Figure 8 illustrates the time needed for construction of the AABB tree data structures for an increasing amount of obstacle trajectories. In case of fixed motion predictions throughout the planning cycle, this operation has to be performed once per planning cycle. For a total of 30 obstacle trajectories, the computation time for constructing the accelerating tree structure is around 7.8 ms. We argue that 30 obstacles represents a fairly large number for a broad spectrum of applications and that the computation time for constructing the tree structure for the obstacles is acceptable for online operation.

The runtime for the construction of the AABB tree data structure for one trajectory is 260 µs. This clearly limits the number of ego trajectories that can be tested for collision within the "Tree vs. Tree" approach in a real-time scenario, assuming a planning rate of approximately 100 ms is targeted. Sampling-based planners such as those presented in [10] and [13] explore up to thousands of candidate trajectories, hence rendering the 'Tree vs. Tree" approach infeasible for planners relying on an online generation of the candidate trajectories. Yet it is a valuable option if the tree structures can be precomputed for the candidate trajectories.

The computation time for the creation of the convex hull representation of one trajectory is $740 \pm 38\,(683 - 968)$ µs (mean $\pm$ std (min - max)). We performed our tests with

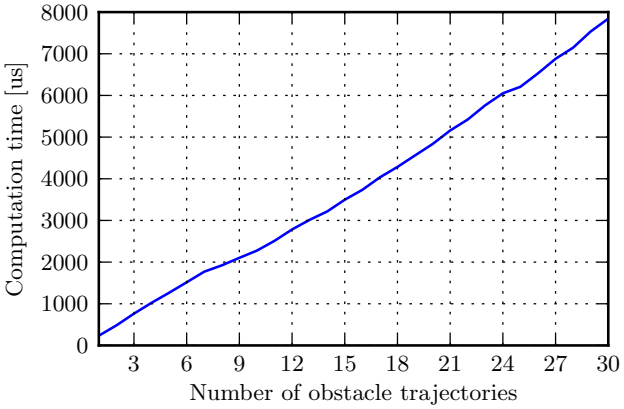


Fig. 8: Mean runtimes given in µs for construction of AABB tree data structure for varying number of obstacle trajectories.

convex hulls with the Bullet Physics Library [15] due to the immature support of convex hulls within the Flexible Collision Library (FCL) [18]. Similar to the computation times of the accelerating AABB tree for the candidate trajectories, this exceeds the computation time available for online operation but again constitutes a valuable option given the possibility for offline preprocessing. Although the narrow phase collision query computation time for a single convex hull shape is two to three times slower than the one for a 3D box, the computation time for complete trajectory collision queries is only around 20% higher. Narrow phase collision checks only have to be performed in cases where the AABB intersect at a leaf of the tree, thus achieving computation times far below the accumulated ones of the narrow phase checks.

V. Conclusion

In this paper we reported on the implementation of an AABB tree data structure for fast collision queries in workspace-time space. The approach makes no assumptions on the shapes of moving obstacles and can handle arbitrary motions. We discussed and compared different approaches. Testing single configurations against an AABB tree capturing the union of workspace-time obstacles results in substantial improvements in runtimes over the naive multiple interference test, while keeping the computational overhead that comes with the construction of the accelerating AABB tree data structures low. Our results on computation times underline the real-time applicability of this approach. Computing an AABB tree for every candidate trajectory provides further speed-up but comes with a significant additional computational cost for common sizes of the candidate trajectory set, rendering this approach only feasible if precomputation of the tree structures is possible.

VI. Acknowledgement

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 269916, V-Charge and grant agreement no 610603, EUROPA2.

References

[1] S.M. LaValle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. Technical report, 1998.

[2] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[3] Mihail Pivtoraiko and Alonzo Kelly. Constrained motion planning in discrete state spaces. In *Field and Service Robotics*, volume 25 of *Springer Tracts in Advanced Robotics*, pages 269–280. Springer Berlin Heidelberg, 2006.

[4] Michael Erdmann and Tomás Lozano-Pérez. On multiple moving objects. *Algorithmica*, 2(1-4):477–521, 1987.

[5] JC Latombe. *Robot motion planning*. 1990.

[6] M. Zucker, J. Kuffner, and M. Branicky. Multipartite rrts for rapid replanning in dynamic environments. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1603–1609, April 2007.

[7] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized Kinodynamic Motion Planning with Moving Obstacles. *The International Journal of Robotics Research*, 21(3):233–255, 2002.

[8] J P van den Berg and M H Overmars. Roadmap-based motion planning in dynamic environments. In *Intelligent Robots and Systems, IEEE/RSJ International Conference on*, volume 2, pages 1598–1605, 2004.

[9] A. Lawitzky, D. Wollherr, and M. Buss. Maneuver-based risk assessment for high-speed automotive scenarios. In *Intelligent Robots and Systems, IEEE/RSJ International Conference on*, pages 1186–1191, Oct 2012.

[10] M. Werling, S. Kammel, J. Ziegler, and L. Groll. Optimal trajectories for time-critical street scenarios using discretized terminal manifolds. *The International Journal of Robotics Research*, 31(3):346–359, December 2011.

[11] J Ziegler and C Stiller. Fast collision checking for intelligent vehicle motion planning. In *Proc. of the IEEE Intelligent Vehicles Symposium (IVS)*, pages 518–522, June 2010.

[12] Dave Ferguson, M. Darms, C. Urmson, and S. Kolski. Detection, prediction, and avoidance of dynamic obstacles in urban environments. In *Proc. of the IEEE Intelligent Vehicles Symposium (IVS)*, pages 1149–1154, June 2008.

[13] Ulrich Schwesinger, Martin Rufli, Paul Furgale, and Roland Siegwart. A Sampling-Based Partial Motion Planning Framework for System-Compliant Navigation along a Reference Path. *Proc. of the IEEE Intelligent Vehicles Symposium (IVS)*, (269916):6, 2013.

[14] Russell Smith. Open Dynamics Engine. `http://www.ode.org/`, May 2014.

[15] Bullet physics library. `http://bulletphysics.org/wordpress/`, May 2014.

[16] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 171–180, New York, NY, USA, 1996. ACM.

[17] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *Visualization and Computer Graphics, IEEE Transactions on*, 4(1):21–36, Jan 1998.

[18] Jia Pan, Sachin Chitta, and Dinesh Manocha. FCL: A general purpose library for collision and proximity queries. In *IEEE International Conference on Robotics and Automation*, pages 3859–3866. IEEE, May 2012.

[19] S. Cameron. *Modelling Solids in Motion*. PhD thesis, University of Edinburgh, 1984.

[20] S. Cameron. A study of the clash detection problem in robotics. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 488–493, Mar 1985.

[21] S. Cameron. Collision detection by four-dimensional intersection testing. *Robotics and Automation, IEEE Transactions on*, 6(3):291–302, Jun 1990.

[22] Pablo Jiménez, Federico Thomas, and Carme Torras. 3d collision detection: a survey. *Computers & Graphics*, 25(2):269–285, 2001.

[23] F. Schwarzer, M. Saha, and J.C. Latombe. Adaptive dynamic collision checking for single and multiple articulated robots in complex environments. *Robotics, IEEE Transactions on*, 21(3):338–353, June 2005.