Efficient, Risk-Encoding Octrees For Path Planning With A Robot Manipulator

Tobias Werner, David Harrer, and Dominik Henrich

Lehrstuhl für Robotik und Eingebettete Systeme,
Universität Bayreuth, D-95440 Bayreuth, Germany,
`tobias.werner@uni-bayreuth.de`,
`http://robotics.uni-bayreuth.de`

Abstract. Recent research in robotics envisions shared human-robot workspaces to combine individual advantages of humans and robots. As part of this vision, robot manipulators must avoid collisions with humans and other a priori unknown obstacles in the shared workspace. State-of-art approaches (e.g. certainty grids, bounding volume hierarchies) test robot poses for collisions under individual limitations (e.g. memory or processing overhead, assumption of global views or noiseless sensors). In contrast, we contribute a sample-based pose test alongside a hierarchical representation of risk over the shared workspace. Our contribution has low memory and processing overhead, and allows for local, outdated, or noisy sensor views. Experiments with real-world data validate this claim and show advantages and limits of our approach over competing variants. We conclude that our pose test and risk representation enhance real-time path planning for robot manipulators in current and future use cases.

Keywords: path planning, robot manipulator, risk-based environment model, shared human-robot workspaces

1 Introduction

Human-robot collaboration promises to combine individual virtues of humans and robot manipulators. Respective research envisions various future use cases for robots, from flexible industrial automation to applications in small businesses and the service sector. However, human-robot collaboration mandates shared human-robot workspaces, which contain a priori unknown obstacles (e.g. humans or human-placed objects). The robot manipulator must thus be able to avoid obstacles in real-time. State-of-art solutions for real-time obstacle avoidance (e.g. [13], [14]) propose a two-tiered approach: Real-time workspace monitoring with a multi-camera system finds 3D workspace volumes that are occupied by obstacles. Concurrently, real-time path planning finds collision-free or risk-minimized robot trajectories by testing robot poses against the occupied workspace volumes.

Pose tests are a crucial aspect when developing a joint monitoring and path planning solution. Tests must be efficient to enable real-time robot reactions, but tests also depend on the results of workspace monitoring (e.g. data structure, available information about obstacles). For instance, representing obstacles as a certainty grid (e.g. [11]) lends itself to sampling-based risk queries per pose (e.g. [1]), while a bounding volume hierarchy over obstacle points (e.g. [12]) enables hierarchical distance queries per pose (e.g. [3]). A choice of obstacle representation and pose test furthermore limits the capabilities of later path planners: Certainty grids, for example, natively support planning risk-minimized robot trajectories, while voxel grids with binary occupation (e.g. [9]) do not. In total, a holistic approach to monitoring and pose tests is necessary.

Formally, state-of-art path planners (e.g. [7]) for a robot manipulator with d joints use a generic pose test $t : \mathbb{R}^d \to C$. This generic test evaluates a pose p to one value $t(p)$ in regard to some criterion C, such as to discard randomly sampled poses or to weight multiple trajectory alternatives. For example, $C = \{\text{true}, \text{ false}\}$ for collision-free trajectory planning (i.e. $t(p)$ indicates collision or no collision at pose p), and $C = \mathbb{R}$ for risk-minimized trajectory planning (i.e. $t(p)$ names the risk of a collision at pose p or a suitable risk heuristic).

Pose tests in turn ideally compare the workspace volume $V : \mathbb{R}^d \to 2^{\mathbb{R}^3}$ that the robot occupies for a pose p to any obstacles. Let $O : 2^{\mathbb{R}^3} \to C$ be the ground-truth result of this comparison, thus $t(p) = O(V(p))$. However, a ground-truth O is not readily available (e.g. due to a priori unknown obstacles, sensor noise, or inherent discretization). This leads to our problem statement: We seek an approximation $O_H : 2^{\mathbb{R}^3} \to C$ of O. Note the approximation O_H encompasses both a data structure for available information on workspace obstacles and a corresponding per-pose test algorithm. To support real-time path planning, the data structure of O_H must be efficient to build, and the pose test O_H must be fast to compute, which gives two conflicting constraints.

Current state-of-art approaches to an approximation O_H have several limits. We discuss select approaches and their limits in Section 2. In contrast to existing approaches, we contribute an alternative data structure and query for O_H: We use a hierarchical representation for O_H that is both efficient to build and to test. Additionally, our O_H supports risk queries. We detail our data structure and pose test algorithm in Section 3. Experimental results with real-world data and a critical comparison against state-of-art approaches from afore follow in Section 4 and substantiate our conclusion in Section 5.

2 Related Work

Data structures currently in use for O_H include point clouds with a superimposed tree structure (e.g. [12]), voxel grids (e.g. [9]), distance grids (e.g. [10]), distance octrees (e.g. [6]), certainty grids (e.g. [11]), or octrees over binary (e.g. [13]) or probabilistic (e.g. [4]) payload. Robots are commonly represented as exact triangle meshes (e.g. [14]), as convex volume approximations (e.g. [3]), or as sampling-based representations (e.g. here), all optionally with a superimposed bounding volume hierarchy. In general, hierarchical data structures take more time to build, but require less memory and afford more efficient pose tests.

Pose tests, on the other hand, depend on the chosen data structure. Exact tests on primitives (e.g. [15]) give exact results, but are slow if O_H consists of many primitives. Sampling-based pose tests (e.g. here) check individual robot points against O_H and are particularly suited for grid-based data structures. Optimizing variants (e.g. [2]) require a volumetric data structure, but are more efficient than sampling if highly precise results are required. Finally, hierarchical collision or distance tests (e.g. [3]) accelerate all other pose tests, but require superimposed hierarchical data structures and thus increase build times.

Basic holistic variants use voxel grids and sample these over the robot model to check for collisions (e.g. [9]). While intuitive to implement, this approach does not scale well with increasing resolution of O_H. Efficient holistic approaches, in contrast, build bounding volume hierarchies over robots and obstacles and evaluate these with exact (e.g. [15]) or optimizing (e.g. [3]) pose tests.

3 Our approach

As a basis for a risk-encoding obstacle approximation O_H, we need to establish an understanding of risk in the workspace of a robot: Risk commonly is defined as the probability of an event times the loss resulting from that event (e.g. [5]). The event, in our case, is a collision between the robot manipulator and any obstacle in the work cell. We assume a worst-case, constant loss (e.g. independent of robot speed). We furthermore assume that there is a direct correlation between the distance to indeterminately moving obstacles and collision probability.

Under above assumptions, we propose to use the closest distance to obstacles as a heuristic for risk associated with a single point of the workspace. The risk approximation O_H for a robot pose then is the minimum closest distance to obstacles over all points in the workspace volume that the robot manipulator occupies in that pose. Extended risk options may additionally weight the per-point or per-pose distance (e.g. by sensor noise, data age, or robot speed).

In the following, we discuss a holistic approach to a risk-based O_H. To this end, we propose a variant of octree as a data structure to encode distance-based risk over the workspace. Notably, we extend the well-known Brushfire algorithm to efficiently build such an octree from a traditional octree over binary obstacle occupation. Finally, we examine sampling of the risk octree at points on the surface of the robot model as a pose test.

3.1 Hierarchical Brushfire algorithm

The Brushfire algorithm (e.g. [10]) calculates an approximation of the distance to the closest obstacle for all cells in a grid representation of the workspace. To do so, the Brushfire algorithm initially marks distances for cells in free space as unknown, for cells in obstacles as zero. Brushfire then iteratively propagates and increments known closest distances from grid cells to any neighboring cells of unknown distance.

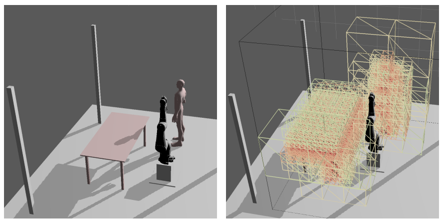

Fig. 1. A mock-up workspace (left) and an excerpt of the risk-encoding octree (right) with high-risk leaf nodes close to obstacles (red) and low-risk leaf nodes far from obstacles (orange).

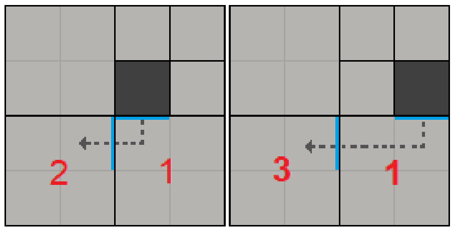

Fig. 2. An octree section over free space (light) and obstacles (dark). Any Brushfire distances over node corners depend on the propagation direction (dotted arrow). Marks (blue) keep track of this.

We extend the Brushfire algorithm to work on the nodes of an octree over binary obstacle occupation (e.g. [13]): We initialize closest distances of all nodes in free space to unknown, closest distances of all occupied nodes to zero. Thereafter, we iteratively propagate and increment closest distances from nodes to any larger neighboring nodes. We stop once each node has a closest distance. See Figure 1 for a workspace and the resulting risk-encoding octree.

In the above outline, actual distance propagation requires further elaboration: Since nodes of an octree in general do not share the same size, we usually increase any propagated distance not by a fixed cell size, but by the size of the originating octree node. However, to get a correct and reasonably exact lower bound on the distance to the closest obstacle, we must consider three special cases: Propagation of distances over corners, propagation to smaller nodes, and refinement of too large nodes. An in-depth discussion of each special case follows. See Algorithm 1 for accompanying pseudo-code.

Special case: Propagation over corners. In a grid, each Brushfire iteration increases the propagated distance by the size of a cell regardless of the direction to the closest obstacle. In an octree, however, we can only increase the propagated distance by one full node size if the next obstacle lies in the opposing direction of the neighboring node. When propagating around a corner, adding a full node size yields too large distances. Figure 2 illustrates this problem.

We solve above problem by tracking from which direction a closest distance propagated to some octree node. To this end, we introduce node marks: We classify all sides of octree nodes in regard to global coordinates (e.g. positive X side or negative Z side) and assign local 2D coordinates. Marks then consist of an identifier for a cube side and a position in local 2D coordinates.

By placing and evaluating marks, we can propagate correct closest distances over corners: When propagating a closest distance from an originating node to a neighboring target node, we need to check the originating node for marks on sides perpendicular to the shared side between both nodes. Any mark closest to the shared side is the correct mark to propagate distance from. Instead of a full node size, we add only the distance between mark and target node to the closest distance of the originating node to find the closest distance of the target node. Finally, we place a new mark on the shared side of the target node, with local coordinates closest to the selected mark of the originating node.

Special case: Propagation to smaller nodes. Our Brushfire variant iterates over the octree from smaller nodes to larger ones. This iteration order is efficient, as it requires no tracking or searching of all (possibly many) smaller neighbors for large nodes. However, our iteration order also implies that marks cannot propagate distances into the opposite direction: from large nodes to neighboring, smaller nodes. In turn, the distances for smaller nodes may be wrong when the direction to the closest obstacle leads through a larger node.

To solve this problem, we separately propagate distances from larger nodes to smaller nodes. To this end, we store distance information not only in leaf nodes, but also in inner nodes of the octree. Each inner node then propagates distance information to its children: If a child has a distance that is larger than or equal to the distance of a parent inner node and if this child shares one or more marks with this parent, we copy the marks of the parent to the child. Note this possibly entails an offset in local coordinates. If the parent additionally has a lower closest distance than the child, then the child discards its marks and assumes the closest distance of its parent. See Figure 3 for an example.

Special case: Refinement of too large nodes. Depth and thus resolution of the original, binary octree depend on matches in the structure of octree and obstacles. This implies that there may be large octree nodes close to obstacles if the obstacle and node surfaces are similar. In turn, our hierarchical Brushfire may associate a large volume of the workspace with a single closest distance.

Too small distances still give a conservative and thus safe approximation of risk. However, path planners find better (e.g. more reliable) trajectories if the

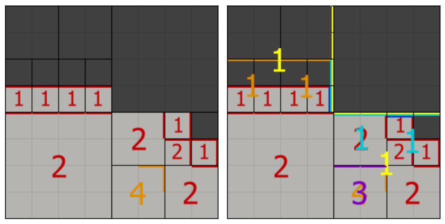

Fig. 3. Left: Incorrect closest distances (e.g. orange four), as not propagated from the large, occupied upper-right node. Right: Propagating intermediate distances on inner nodes fixes this issue (additional numbers and marks).

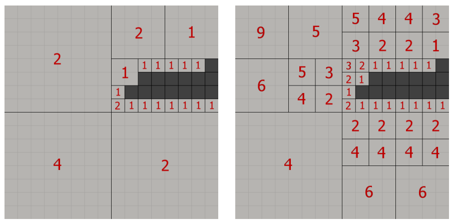

Fig. 4. Left: Overlapping structures of octree and obstacles cause large nodes close to the obstacles. Distance errors impede path planning in remote regions. Right: Adaptive subdivision improves distances on remote nodes.

```
Input  : Octree $O_H$ over binary obstacles
Output: Octree $O_H$ with additional risk encoding

heap openlist, topdownlist       // Ordered by increasing node size

Insert all occupied nodes of $O_H$ into openlist

while !openlist.empty() and !topdownlist.empty() do
    foreach node $k$ = openlist.pop() do
        foreach neighbour $n$ of $k$ do
            Calculate distance $d$ for $n$ from $k$
            if $d < n$.distance then
                save $d$ as $n$.distance and copy marks
                if $n$.isleaf() then
                    openlist.insert($n$)
                else
                    topdownlist.insert($n$)

    foreach node $k$ = topdownlist.pop() do
        foreach child $c$ of $k$ do
            if !$c$.occupied() and $c$.distance > $k$.distance
                                and $k$ has mark over $c$ then
                copy distance and marks from $k$ to $c$
                if $c$.isleaf() then
                    openlist.insert($c$)
                else
                    topdownlist.insert($c$)
```

Algorithm 1: Hierarchical Brushfire Algorithm.

risk approximation O_H is close to the ground-truth obstacle occupation O. Our octree Brushfire therefore increases the precision of O_H by splitting large nodes close to obstacles: Whenever we store a closest distance in a node, we evaluate a splitting heuristic with thresholds for distance and node size. If the heuristic is met, we split the node and treat any new children as specified under propagation to smaller nodes. This procedure results in an octree with finer resolution around obstacles. Figure 4 illustrates this process.

3.2 Pose tests

Once a risk-encoding octree is available (e.g. built by the above hierarchical Brushfire algorithm), all that remains is to evaluate O_H with this octree for a given robot pose. To do so, we propose to sample the robot surface by regularly spaced points. After rigid transformations to world coordinates for a given robot pose, we can push each point through the octree to determine a matching leaf node. Fusing values of all leaf nodes (e.g. closest obstacle distance) then gives the desired result of O_H. The actual fusing method depends on the risk encoded in the octree. For a distance-based risk, we choose the closest distance over all of the samples as the output of O_H.

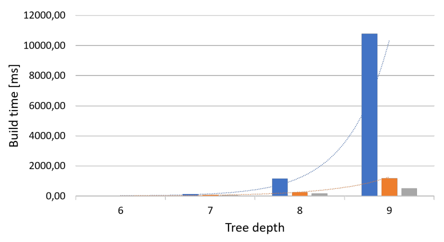

Fig. 5. Build times for our hierarchical Octree brushfire without (gray) and with (orange) refinement as opposed to a grid variant (blue) over select resolutions.

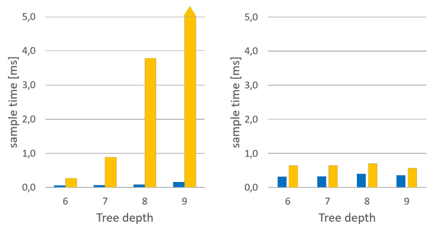

Fig. 6. Sampling times for robot poses with points (blue) or triangles (orange) as samples in a grid (left) or our risk-encoding octree (right).

4 Evaluation

In the following, we evaluate our hierarchical octree Brushfire and the associated sampling-based pose test. As ground-truth obstacles, our mock-up workspace contains a human and a table. Furthermore, we use a CAD model of a Stäubli TX90-2L robot arm for pose tests. This model consists of 286 triangles, which we approximate by around 3000 points of mostly equal spacing. See Figure 1 for an image of our mock-up workspace.

We first compare building times of our Brushfire algorithm with and without refinement against a traditional, grid-based variant. To this end, we used grids of various resolutions and octrees of matching lowest-level resolutions. Overall, our octree Brushfire outperforms the grid-based variant, with the benefit growing super-linearly by resolution. Figure 5 shows respective timings.

Next, we evaluate pose test times on our risk-encoding octree against a risk-encoding grid. As a counterpart to our approximate point sampling test, we used a naive triangle-based pose test to determine exact risks. Each experiment involved timing pose tests for 200 random poses. Triangle tests on the grid quickly become infeasible with growing grid resolution due to the number of cells to check for intersection. In contrast, our risk-encoding octree effortlessly handles triangles even at a large resolution. Pose tests with point sampling are substantially faster on the grid, since no octree traversal is necessary. Figure 6 presents timings for each combination of data structure and pose test.

Overall, slower sampling times partially offset the benefit of our efficient octree Brushfire. A reasonable comparison thus needs to amortize the build time over sampling times. Our experiments show that for a grid with $2^9 = 512$ subdivisions and a matching octree of depth 9, our hierarchical Brushfire amortizes building costs only for less than about 100,000 pose tests per build of the data structure. Otherwise, the grid is faster. In practice, reasonable real-time path planners are bound to real-time update rates around or above 10 Hz (e.g. [14]). At the measured 0.5 ms for 200 pose tests, such planners can test at most 10,000 poses per update and thus do not reach the break-even point of grids. We conclude that our octree outperforms grids in practically relevant cases.

5 Conclusion

In the preceding, we have contributed a risk-encoding octree and an algorithm that builds this octree from an existing, binary octree over obstacle occupation. The resulting risk-encoding octree balances acceptable build times against fast pose tests and improves timings over naive grid variants for practically relevant numbers of pose tests. Future work examines further effects of weighting risk by robot speed or sensor noise.

Acknowledgements

This work has partly been supported by the Deutsche Forschungsgemeinschaft (DFG) under grant agreement He2696/11 SIMERO.

References

1. J. Borenstein, Y. Koren, "Real-time obstacle avoidance for fast mobile robots", Systems, Man, and Cybernetics, 1989.
2. S. Cameron, "Enhancing GJK: Computing minimum and penetration distances between convex polyhedra", ICRA, 1997.
3. D. Henrich, X. Cheng, "Fast Distance Computation for On-line Collision Detection with Multi-Arm Robots", ICRA, 1992.
4. A. Hornung et al., "OctoMap: an efficient probabilistic 3D mapping framework based on octrees", Autonomous Robots, 2013.
5. ISO, "ISO31000 Risk management–principles and guidelines", International Organization for Standardization, 2009.
6. D. Jung, K.K. Gupta, "Octree-based hierarchical distance maps for collision detection", Journal of Robotic Systems, 1997.
7. L. Kavraki, P. Svestka, MH. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", 1994.
8. B. Lacevic, D. Osmankovic, A. Ademovic, "Burs of free C-space: A novel structure for path planning", ICRA, 2016.
9. A. Ladikos, S. Benhimane, N. Navab N, "Efficient visual hull computation for real-time 3D reconstruction using CUDA", CVPRW, 2008.
10. B. Lau et al., "Efficient grid-based spatial representations for robot navigation in dynamic environments", Robotics and Autonomous Systems, 2013.
11. H. P. Moravec, "Sensor fusion in certainty grids for mobile robots", Sensor devices and systems for robotics, 1989.
12. J. Pan, S. Chitta, D. Manocha, "FCL: A general purpose library for collision and proximity queries.", ICRA, 2012.
13. T. Werner, D. Henrich, "Efficient and Precise Multi-Camera Reconstruction", ICDSC, 2014.
14. T. Werner, D. Henrich, D. Riedelbauch, "Design and Evaluation of a Multi-Agent Software Architecture for Risk-Minimized Path Planning in Human-Robot Workcells", Kongress Montage Handhabung Industrieroboter, 2017.
15. T. Werner; D. Henrich; M. Sand, "Sparse and Precise Reconstruction of Static Obstacles for Real-Time Path Planning in Human-Robot Workspaces", ISR, 2018.