2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Real-time Batched Distance Computation for Time-Optimal Safe Path Tracking

Shohei Fujii [1,2] and Quang-Cuong Pham[1,3]

[1]*School of Mechanical and Aerospace Engineering, Nanyang Technological University, Singapore*
[2]*DENSO CORP., Japan*
[3]*Eureka Robotics, Singapore*

***Abstract*— In human-robot collaboration, there has been a trade-off relationship between the speed of collaborative robots and the safety of human workers. In our previous paper, we introduced a time-optimal path tracking algorithm designed to maximize speed while ensuring safety for human workers [1]. This algorithm runs in real-time and provides the safe and fastest control input for every cycle with respect to ISO standards [2]. However, true optimality has not been achieved due to inaccurate distance computation resulting from conservative model simplification. To attain true optimality, we require a method that can compute distances 1. at many robot configurations to examine along a trajectory 2. in real-time for online robot control 3. as precisely as possible for optimal control. In this paper, we propose a batched, fast and precise distance checking method based on precomputed link-local SDFs. Our method can check distances for 500 waypoints along a trajectory within less than 1 millisecond using a GPU at runtime, making it suited for time-critical robotic control. Additionally, a neural approximation has been proposed to accelerate preprocessing by a factor of 2. Finally, we experimentally demonstrate that our method can navigate a 6-DoF robot earlier than a geometric-primitives-based distance checker in a dynamic and collaborative environment.**

I. Introduction

Collaborating with robots while ensuring human safety has been a critical challenge, as slowing down the robot operation to mitigate injuries will impede productivity. To maximize the productivity of collaborative robots while guaranteeing the safety, we have proposed time-optimal path tracking algorithm [1] which runs in real-time and provides the safe and fastest control input with respect to *Speed and Separation Monitoring* in ISO standards [2]. In this path-tracking method, distances between the obstacles and a robot for waypoints along an executing trajectory must be given. Given the distances, the algorithm computes the fastest velocity profile and navigates the robot in a time-optimal manner (Fig. 1). Finally, the control input is sent to a robot to follow the derived velocity profile. This whole process must run in every control cycle, which is about 10 ms according to the communication protocol of industrial robots[1].

To achieve true optimality in path tracking, the precise distances need to be given. In our previous paper, the robot is simplified with spheres and the distances between the spheres and voxels are computed with *hypot* function using their center positions. Such distance checking with a simplified model can run almost in real-time. However, the computed distances are smaller than its actual value due to its simplification, which makes the robot's behavior conservative and exacerbates the productivity of the robot. In contrast, a exact mesh-to-mesh distance checker cannot run in real-time (experimentally, 130 µs per one configuration, 65 ms per one trajectory with FCL [3]). To the best of our knowledge, no existing distance checker is applicable to real-time safety control.

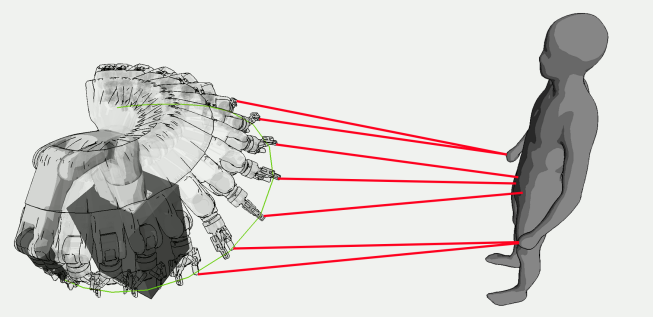


Fig. 1: Problem Setting Overview: Computing distances in real-time, across multiple robot configurations, with precision. See Section I for more information.

In this paper, we propose a batched, fast and precise distance checker based on pre-computed link-local Signed Distance Fields(SDFs) to address this issue. Leveraging GPU parallelization for pre-processing of robot's SDFs, the proposed method is able to check distances for multiple robot configurations within less than 1 ms at runtime. Additionally, a neural approximation of the pre-processing has been proposed, resulting in 2x faster pre-processing. Finally, we experimentally demonstrate that our distance checker actually navigates a robot faster than the method using a robot modeled with spheres in a dynamic, collaborative environment.

The paper is organized as follows. We survey related work in Section II. Section III presents our parallel distance checking method and some techniques to reduce the pre-processing time in a constant order including the neural approximation. In Section IV, we evaluate the performance of the neural

E-mail: SHOHEI001@e.ntu.edu.sg; Corresponding author

[1]For example, the control period is 8 ms in the case of DENSO b-CAP communication protocol `https://www.denso-wave.com/en/robot/product/function/b-CAP.html`.

approximation and also examine that the approximation does not affect the precision of distance computation. Then, the experimental comparison is shown for the real-time safe path tracking in a collaborative environment. Finally, we discuss the limitations of our approach and conclude with some directions for future work in Section V.

II. Related Work

A. Model-based Distance Computation

In collision detection and distance computation between 3D models, hierarchal data structure, or 'broad-phase structure', is commonly applied to filter out object pairs that are far away and dramatically accelerates the collision/distance queries. Examples of such data structures include AABB Tree, OCTree and Inner Sphere Tree [3]–[5]. However, these data structures are optimized for CPU and lack batch-processing capabilities, hence they do not have an sufficient throughput for real-time safety control. For instance, the distance query with an octree for 1000 configurations will take 39.1 ms according to [4] (note that this does not include the octree construction time), which is still slow for real-time safety control. Another challenge is that the throughput depends on the positions of robots and obstacles, which makes it difficult to ensure its constant execution time at the time of deployment of the system. In contrast, our approach does not depend on runtime-varying settings except for the number of configurations.

Simplification of robot/human models with geometric primitives such as spheres and capsules is commonly used for distance computation in motion planning and safe robotic operation [6]–[9]. However, evaluating distances for multiple configurations in a batched manner using primitives other than spheres are actually slow. In fact, our preliminary experiment shows that distance computation between a 6 DoF robot simplified with 7 capsules and (only) 3000 points for 500 configurations took about 70 ms even on GPU, which is not applicable to real-time control. This is primarily because the projection of points onto the axis of capsules requires a time-consuming (batched) matrix multiplication at runtime.

B. Signed Distance Fields (SDFs)

In unknown environments, prior knowledge about obstacles such as their shape, size and position is not always accessible. Therefore, Signed Distance Fields(SDFs) or Unsigned Distance Fields(USDFs) of an *environment* are often used because these does not necessarily require a prior knowledge on obstacles and offers the gradient of (U)SDFs to push the trajectory away from obstacles [6], [10], [11]. There are a number of methods for SDFs construction; some come from a context of SLAM [12]–[14] and some from that of machine learning/NeRF [15]. However, most of them assume a *static* environment and incrementally construct a scene since SDFs reconstruction requires data propagation which is inherently time-consuming. To the best of our knowledge, no previous work satisfies all the requirements for the safe-control application: 'batched', 'real-time' and 'precise'.

One of the promising work is [16] which builds on [17], [18]. This method computes the SDFs of an environment from an incoming sensory pointcloud in parallel on GPU using a Parallel Banding Algorithm [19]. The distance data for the voxels occupied by the robot is then retrieved. In their demonstration, they showcase online motion planning of a mobile manipulator platform. However, the computation time of this method depends on the size of environment due to the data propagation and is not suitable especially for large environments. Most importantly, their reported SDFs construction time is 17.5 ± 0.4 ms for 5cm resolution and 36.2 ± 8.3 ms for 2.5 cm resolution, which is not fast enough for real-time control (Note that the 'SDFs computation time' does not include the time for distance queries).

Another interesting work is ReDSDF [20], a machine learning based SDFs estimator that employs a neural network which takes query points and poses as inputs and outputs distances for each of them. While its estimation accuracy is sufficient for safety-critical use-cases, their architecture is not suitable for real-time safety control due to the following reasons: For robot's SDFs generation: 1. ReDSDF requires retraining of a neural network for any change in a robot, and 2. the neural network needs to be evaluated for each waypoint along a trajectory, repeatedly feeding the same query points. These requirements make ReDSDF unsuitable. For environment's SDFs construction: 1. ReDSDF requires a model that is trained for each individual obstacle, and 2. each obstacle needs to be tracked in some way; these are not realistic for industrial applications.

Some previous methods employ link-local SDFs of a *robot*, instead of an *environment*, for distance computation in motion planning or collision avoidance [21], [22]. In these approaches, the pointcloud is transformed into every link coordinate and the distance is then obtained from link-local SDFs, resulting in a computational complexity of $O(DM)$ where D is the DoF of a robot and M is the number of pointcloud. This computation heavily depends on the number of points and its batch processing is often insufficiently fast for real-time robot control. In contrast, our method performs the transformation of the link-local SDFs onto the coordinates of the environment beforehand, eliminating the need for pointcloud transformations (Fig. 2). This leads to a faster distance retrieval in the real-time distance computation phase.

III. Batched Robot SDFs Computation

A. Overview

The pipeline of our parallel distance checking is illustrated in Fig. 3. We consider a D-DoF robot and examine distances at C robot configurations ($\theta_{\mathbf{c}} \in \Theta$). The environment is discretized into voxels whose extent is $\mathbf{e_e} = (e_{ex}, e_{ey}, e_{ez})$ and resolution is $\mathbf{r_e} = (r_{ex}, r_{ey}, r_{ez})$. The total number of voxels of the environment V_e is $\prod \frac{2\mathbf{e_e}}{\mathbf{r_e}}$.

At the preprocessing stage, given a robot model, we precompute Signed Distance Fields(SDFs) for each link on its local coordinates. We call it as *Link SDFs*. We refer $\mathbf{e_r}$ to the extent of Link SDFs (e_{rx}, e_{ry}, e_{rz}) and $\mathbf{r_r}$ to the resolution of

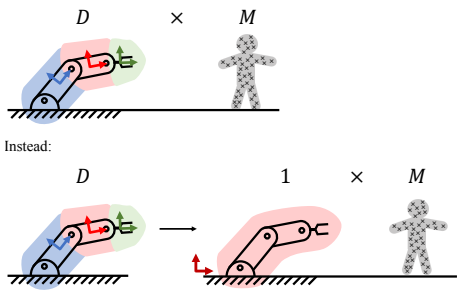


Fig. 2: A common way to compute a distance between pointcloud and SDFs requires pointcloud transformations for each link coordinate. Instead, we transform and merge the link-local SDFs into the global coordinates in a pre-processing stage, and then evaluate it to obtain distances at runtime.

Link SDFs (r_{rx}, r_{ry}, r_{rz}). The size of the precomputed Link SDFs, $2\mathbf{e_r}$, must be divided by the resolution of the voxelized environment $\mathbf{r_e}$ without residue for alignment operation that is later introduced. The resolution of Link SDFs is arbitrary and recommended to be finely voxelized.

Next, given the configurations c, we compute transformations $T_{i,c}$ of each link i by applying parallel forward kinematics. Then, according to $T_{i,c}$, Link SDFs are transformed and aligned into the voxels of the environment. We call the first euclidean transformation operation as "euclidean transformation" and the second alignment operation as "alignment". To compute Robot SDFs for each configuration, a minimum value of the transformed Link SDFs for each link and for each voxel is taken. Besides, obstacles in the environment are voxelized. By extracting the distances at the voxels occupied by the obstacles and taking the minimum value for each link, the distance between the robot and obstacles can be computed.

More specifically, at the "euclidean transformation" stage, we shift the rotated Link SDFs within the half range of the voxel by $\delta t_{i,c} \in (-\frac{r_e}{2}, \frac{r_e}{2})$. And then, at the "alignment" stage, we translate the transformed SDFs by $t_{i,k} - \delta t_{i,k}$ and snap them into the environment voxels. The shift operation is necessary for exact alignment since the position of each link in the environment is not usually at the exact center of the voxel. The transformation can be computed in the scheme of affine grid transformations [23] [2]. $\delta t_{i,c}$ can be computed by following the simple equations:

$$T_{Oi,c} = T_{i,c} - (-e_e) \tag{1}$$
$$k_{i,c} = \lfloor T_{Oi,c}/r_e \rfloor - \lfloor e_e/e_r \rfloor \tag{2}$$
$$\delta t_{i,c} = T_{Oi,c} - (k_{i,c} \cdot r_e + e_r) \tag{3}$$

where $\mathbf{e_e}$ is the 3D extent of environment, $\mathbf{r_e}$ is the 3D resolution of environment, and $\mathbf{e_r}$ is the 3D extent of Link SDFs. The total number of voxels in transformed SDFs V_r is $\prod \frac{\mathbf{e_r}}{\mathbf{r_e}}$.

At runtime, to compute distances against obstacles in the environment based on the Robot SDFs, we voxelize the obstacles and extract the values from Robot SDFs that is occupied by the voxelized obstacles. By reducing the extracted values with *min* for each configuration c, we can obtain minimum distance between a robot and the obstacles for each c. This process is fast because it only reads the data on the GPU memory and does not require any calculation.

As an extra bonus, self-collision detection can be done by aligning "in a predefined and alternating the order of checking, paying attention to the robotś kinematics" as in [24], though it is not applied in our experiment since self-collision is usually examined in the motion-planning phase rather than the execution phase.

B. Techniques for computation time reduction in a constant order

We introduce the following techniques to optimize the computation time in a constant order.

1) Euclidean Grid Approximation with A Tiny Neural Network: The computation of euclidean grid transformation mapping is mathematically a matrix multiplication. Given the center positions of grids $p_{j,xyz}$ for $j \in [0, V_r)$ where V_r is the number of grids in each Link SDFs and link transformations $T_{i,c}$, euclidean grid transformations $G_{i,c}$ can be computed as follows:

$$\begin{aligned} P_{xyz} &:= \begin{pmatrix} \cdots & p_{j,xyz} & \cdots \end{pmatrix} \\ \begin{pmatrix} G_{i,c} \\ 1 \end{pmatrix} &= \begin{pmatrix} R_{i,c} & \frac{\delta t_{i,c}}{e_r} \\ 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} P_{xyz} \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} R_{i,c}^T & -R_{i,c}^T \frac{\delta t_{i,c}}{e_r} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} P_{xyz} \\ 1 \end{pmatrix} \end{aligned} \tag{4}$$

Note that $p_{i,xyz}$ are normalized in the range of $[-1, 1]$ with the voxel resolution of environment voxels $\mathbf{e_r}$. The number of column of P_{xyz} is the total number of voxels in the transformed Link SDFs. This batch processing of matrix-matrix multiplication is actually time-consuming, because general matrix-matrix multiplications of BLAS libraries provided by vendors are optimized for square matrices and we cannot leverage full performance of dedicated devices including GPU for tall-and-skinny matrices [25]. Instead, we use a tiny neural network f which is composed of two fully-connected layers and one ReLU activation layer to approximate and simplify this operation:

$$\begin{aligned} \delta t_{inv\ i,c} &= -R_{i,c}^T \frac{\delta t_{i,c}}{e_r} \\ G_{i,c} &= f(R_{i,c}) + \delta t_{inv\ i,c} \end{aligned} \tag{5}$$

f takes a rotation matrix and output euclidean grid transformations only for the specified rotation. Recent advance in deep learning provides us a highly-optimized API for neural approximation [3]. Since the original operation is deterministic

[2]Please refer to pytorch's documentation as well: `https://pytorch.org/docs/stable/generated/torch.nn.functional.affine_grid.html`

[3]See `https://developer.nvidia.com/cudnn`

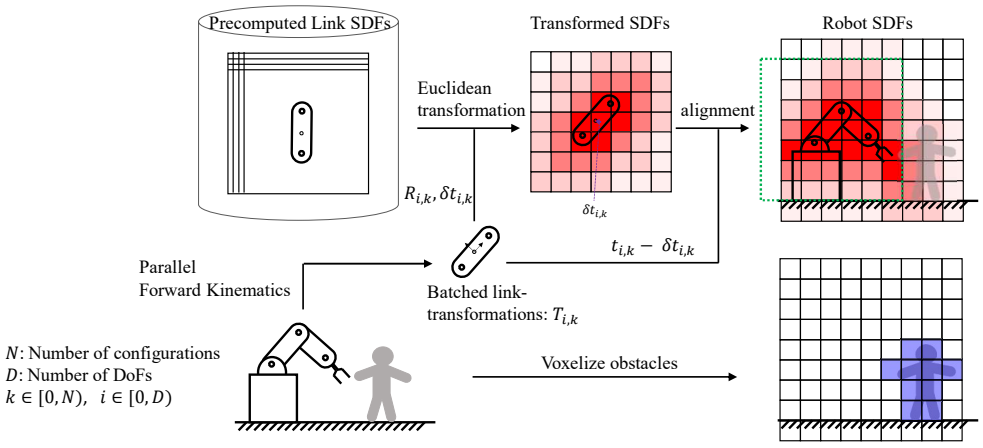


Fig. 3: A pipeline of parallel batched distance checking with pre-computed link-wise signed distance fields (SDFs). This is in 2D for clear illustration, but actual computation is in 3D and in a batched manner. See Section III for detail.

and robot-model agonistic, we can train the neural network quickly (about 10–20 mins) and reuse the pre-trained models for any type of robots once it is trained without any additional training. As described in Section IV-B, the maximum error of this approximation is about 1mm in our setting, which is covered by the discretization error of SDFs and therefore negligible. Therefore, the maximum error from the ground truth which derives from the total pipeline is only the discretization error: $\frac{|r_e|}{2} + \frac{|r_r|}{2}$.

2) Grids in Sphere instead of using Cubic Grids: Another small technique to reduce computation time is to compute transformed SDFs only for the grids in a sphere of a radius $e_r + sqrt(3(\frac{r_e}{2})^2)$ which inscribes the link. We can roughly reduce the number of grids by: $\frac{\frac{4}{3}\pi e_r^3}{(2e_r)^3} \approx 0.53$.

IV. Experiments and Results

A. System setup

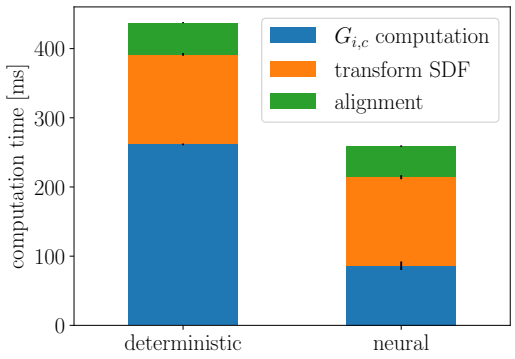
All the experiments are done on a single machine, on which AMD Ryzen™ 9 4900HS and NVIDIA GeForce RTX™ 2060 with Max-Q Design are equipped for CPU and GPU. We use PyTorch and develop custom CUDA kernels for evaluation.

B. Precision and Speed of Neural Approximation for Euclidean Grid Transformations

First, we examine the effect of approximation of euclidean grid transformations in Fig. 4. In this experiment, we train the tiny neural network of 32 hidden parameters using L1 loss and Adam optimizer with a learning rate of $1e^{-4}$. We measured the time to compute euclidean grid transformations for 500 configurations of 6 DoF robot. Deterministic transformations are operated by a function 'torch.matmul' which internally uses cuBLAS's sgemm [4].

We compare the computation speed in Fig. 4a. As a result, the neural approximation of euclidean transformations G is about 3.2x faster than the deterministic one, and the total SDFs computation becomes about 2x faster. According to Figure 22 of [25], ours (3.2x) exceeds the performance gain of [25] (almost 2x) from cuBLAS.

We also test approximation errors from the ground truth.We use 10^7 randomly-generated link transformations to examine the maximum error. The result is that the approximation error of the euclidean grid transformations is less than 0.0013 at the maximum. This means that, in the following experiment, considering the extent of Link SDFs e_r is set to 1.2 m, the actual error in G is $0.0013 \times e_r = 1.56$ mm which is way smaller than the grid discretization size (1 cm) and therefore negligible.



(a)

Fig. 4: Computation speed of euclidean grid approximation by a tiny neural net. See Section IV-B for detail.

C. Comparison with a robot in simulation

Secondly, we compare our method with sphere-based distance checker in simulation (Fig. 5). The robot loops between point A and point B while the experimenter is in close proximity to the robot and randomly moves his arms aside the robot impeding the robot's motion. The robot's motion is planned for each trajectory at runtime. The experimenter's motion is recorded by a Kinect v2, and we replay the

[4] https://developer.nvidia.com/cublas

obtained sequence of pointclouds in each experiment at the same timing. The pointcloud is converted into 4 cm voxels at runtime. We record a total time trajectory execution time for the robot to move back and forth for 6 laps over 10 trials. The protective distance d_{prot} is set to 3 cm and the extent of Link SDFs $\mathbf{e_r}$ is set to 1.2 m. The resolution of pre-computed Link SDFs is set to 1 cm. The clearance threshold is set to $sqrt((4/2)^2 \times 3) + sqrt((1/2)^2 \times 3) \approx 4.3$ cm. Our code is based on OpenRAVE [26].

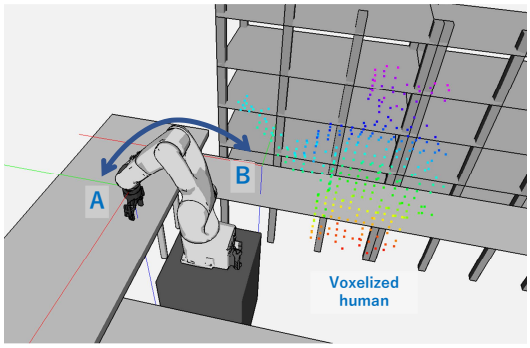


Fig. 5: The experimental setup used to compare our method with a simple method which models a robot with spheres. The robot loops between point A and point B while the experimenter virtually picks up objects from a shelf aside the robot hindering the robot's motion.

At runtime, after planning a trajectory between points using an off-the-shelf RRT-based motion planner in OpenRAVE and before executing the trajectory, intermediate waypoints are sampled using TOPP-RA's automatic gridpoint suggestion feature [27]. The number of waypoints (i.e. the batch size) ranges in 300–500 depending on each trajectory. Robot SDFs are then computed with our proposed method for each waypoint configuration in a parallel, batched manner. During the trajectory execution, the computed SDFs are used to retrieve the distances between a robot and obstacles for each waypoint and time-optimal safe velocity is computed and applied to a robot at every control cycle based on [1].

To ensure the safety, $\mathbf{e_r}$ needs to be large enough to capture the obstacle coming closer to a moving robot. We select the value (1.2 m) as follows: Given the joint velocity limit $v_{limit,i}$ and acceleration limit $a_{limit,i}$ for joint i, the maximum braking time t_{brake} is computed as $\max_i \frac{v_{limit,i}}{a_{limit,i}}$, which is 0.2 sec for DENSO Robot VS-060. Therefore, given the maximum velocity of obstacles v_{obs}, the system needs to capture obstacles coming closer less than $v_{obs}t_{brake} + d_{prot}$ from a robot trajectory. In our case, considering the maximum size from the center of robot links is 0.6 m, $\mathbf{e_r}$ must be larger than $1.6 \cdot 0.2 + 0.03 + 0.6 = 0.95$ m.

As a result, the robot with sphere model takes 39.53 sec to execute its entire task while ours takes 31.81 sec, which is 1.24 times faster (Table I, Fig. 8). The SDFs computation takes approximately 0.2–0.3 sec per one trajectory, which is reasonable to compute at runtime. During trajectory execution, the sphere-based checker takes 5.47 ms to batch-process distances even on GPU, while our batched distance checker requires only 0.4 ms. This is beneficial for achieving high-throughput in a real-time control system (Fig. 6, Table I). It is worth noting that our method invests 2 seconds in total at runtime in SDFs preparation for each trajectory execution and is still faster. If trajectory replanning is unnecessary and the robot follows fixed trajectories and SDFs computation can be done in advance, the total improvement is 1.44 times (from 30.51 s to 21.25 s). The comparison videos can be checked from `https://youtu.be/wO6PiOlsu-w` and `https://youtu.be/YBPpki4fGF8`. Fig. 7 illustrates a typical trajectory execution, logging the minimum distance to obstacles and joint velocities. A dotted horizontal line in the plot represents the protective distance. This plot shows that the safety is secured by stopping the robot before collision happens.

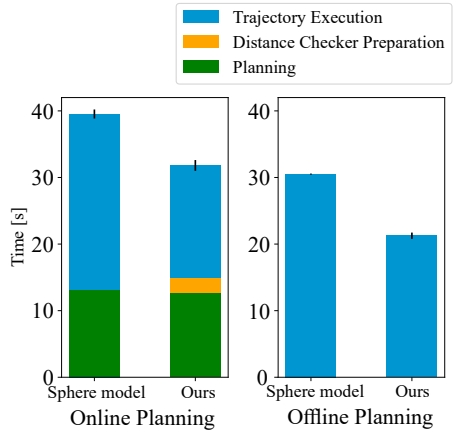


Fig. 6: Comparison of execution times in a dynamic environment. Despite the additional time spent on preparing SDFs, our method exhibits a 1.24x faster total execution time compared to the simple sphere model. In case of offline planning (i.e. trajectories are fixed and preparation is done offline), we observe a 1.44x speedup.

TABLE I: Comparison of execution times per one trajectory

	Sphere model	Ours (neural approx.)
Distance Checker Preparation per 1 trajectory [s]	0.15 ± 0.012	2.34 ± 0.057
Runtime Evaluation per 1 trajectory [ms]	5.47 ± 0.096	0.391 ± 0.0014

D. Real Robot Experiment

Finally, we experimentally test our algorithm with a real 6 DoF robot. During the experiment, the robot moves between several configurations while an experimenter inhibits its motion. The environment is captured using Intel RealSense™D455 and voxelized with a resolution of 4 cm. The maximum speed of obstacles is set to 2.0 m/s, and the robot speed is limited to 80% of its capacity for the safety of the experimenter.The experiment can be viewed at `https://youtu.be/iZP_6rD34lA`.

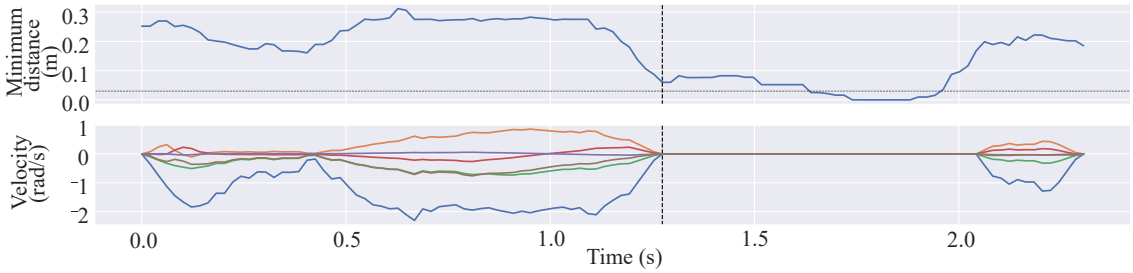


Fig. 7: A result of a trajectory execution, logging Minimum distance from voxelized obstacles and Joint Velocities.

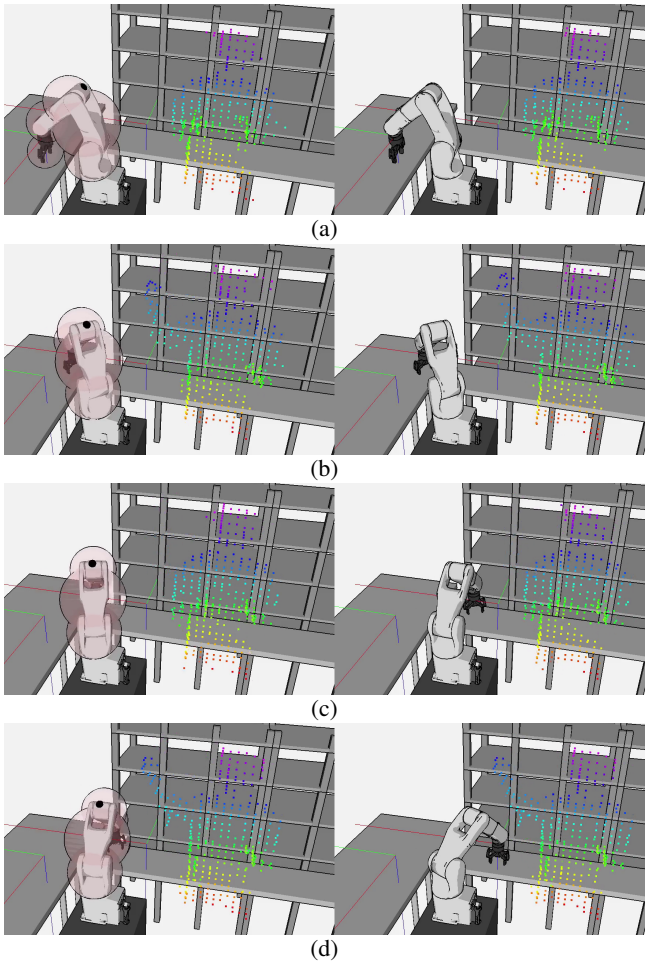


Fig. 8: Experimental comparison of our method with a simple sphere robot model. Left: Sphere model / Right: Our method. The same recording of the experimenter's pointcloud is applied to both methods for comparison. The captured video clips show the first trajectory execution. (a) The robots start to move almost simultaneously. Ours is a little slower due to SDFs computation. (b) Both robots move almost at the same speed initially and starts being hindered by the experimenter. (c) Ours accelerates earlier, (d) arrives at the destination earlier. The full experiment can be viewed at `https://youtu.be/wO6PiOlsu-w`.

V. Conclusion

A batched, fast and precise distance checker has been required for truly time-optimal safe path tracking in human-robot collaborative environments. In this paper, we propose a real-time batched distance checking method based on pre-computed link-local SDFs. Our method can check distances between a robot and obstacles along a trajectory within less than 1 millisecond on GPU, which is suitable for time-critical safety control under a collaborative situation. Additionally, to accelerate a preprocessing process, a neural approximation has been proposed, which makes the preprocessing 2x faster. Finally, we have experimentally demonstrated that our method can navigate a robot earlier than a fast but conservative geometric-primitives based distance checker in a dynamic environment.

It should be mentioned that, by introducing a greater number of smaller spheres in the simple sphere model to enhance its precision, the performance improvement of our method will diminish and the time invested in SDFs preparation may not be justified. However, as the number of spheres and occupied voxels increases, the runtime speed of the sphere-based checker grows linearly, which is critical for ensuring safe control. Indeed, during our experiments, we encountered situations where the runtime speed was insufficient for the control cycle, particularly in larger and congested environments with numerous occupied voxels. On the other hand, our method maintains an advantage in terms of runtime speed. Although it scales linearly to the number of occupied voxels, it remains significantly faster since it only requires GPU memory access at runtime.

One of the limitations of our approach is its scalability in terms of space complexity. It is not well-suited for a large, finely-voxelized environment due to the GPU memory consumption associated with cached SDFs. Although we do not observe a significant memory overhead inherently as the environment size increases, the GPU memory required to store SDFs increases linearly with the number of waypoints and the number of environment voxels. For instance, during the experiment described in Section IV-C, about 3GB GPU memory was consumed at runtime. While this issue can be mitigated by employing multiple GPUs, but considering the hardware cost, further work is needed to reduce memory consumption.

REFERENCES

[1] Shohei Fujii and Quang-Cuong Pham. Time-Optimal Path Tracking with ISO Safety Guarantees. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5926–5933, October 2023.

[2] ISO/TS 15066. Robots and robotic devices collaborative robots, 2016.

[3] J. Pan, S. Chitta, and D. Manocha. FCL: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, pages 3859–3866, 2012.

[4] Jia Pan, Ioan A. Sucan, Sachin Chitta, and Dinesh Manocha. Real-time collision detection and distance computation on point cloud sensor data. In *2013 IEEE International Conference on Robotics and Automation*, pages 3593–3599. IEEE, 2013.

[5] Max Kaluschke, Uwe Zimmermann, Marinus Danzer, Gabriel Zachmann, and Rene Weller. Massively-Parallel Proximity Queries for Point Clouds. In Jan Bender, Christian Duriez, Fabrice Jaillet, and Gabriel Zachmann, editors, *Workshop on Virtual Reality Interaction and Physical Simulation*. The Eurographics Association, 2014.

[6] Nathan Ratliff, Matt Zucker, J. Andrew Bagnell, and Siddhartha Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation*, pages 489–494. IEEE, 2009.

[7] Changliu Liu and Masayoshi Tomizuka. Algorithmic safety measures for intelligent industrial co-robots. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3095–3102, 2016.

[8] Mohammad Safeea, Pedro Neto, and Richard Bearee. Efficient calculation of minimum distance between capsules and its use in robotics. *IEEE Access*, 7:5368–5373, 2019.

[9] Sezgin Secil and Metin Ozkan. Minimum distance calculation using skeletal tracking for safe human-robot interaction. *Robotics and Computer-Integrated Manufacturing*, 73:102253, 2022.

[10] Jing Dong, Mustafa Mukadam, Frank Dellaert, and Byron Boots. Motion Planning as Probabilistic Inference using Gaussian Processes and Factor Graphs. In *Robotics: Science and Systems XII*. Robotics: Science and Systems Foundation, 2016.

[11] Mustafa Mukadam, Jing Dong, Xinyan Yan, Frank Dellaert, and Byron Boots. Continuous-time gaussian process motion planning via probabilistic inference. *The International Journal of Robotics Research*, 37(11):1319–1340, 2018.

[12] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, 2011.

[13] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[14] Luxin Han, Fei Gao, Boyu Zhou, and Shaojie Shen. Fiesta: Fast incremental euclidean distance fields for online motion planning of aerial robots. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4423–4430, 2019.

[15] Joseph Ortiz, Alexander Clegg, Jing Dong, Edgar Sucar, David Novotny, Michael Zollhoefer, and Mustafa Mukadam. isdf: Real-time neural signed distance fields for robot perception. *arXiv preprint arXiv:2204.02296*, 2022.

[16] Mark Nicholas Finean, Wolfgang Merkt, and Ioannis Havoutis. Simultaneous scene reconstruction and whole-body motion planning for safe operation in dynamic environments. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3710–3717, 2021.

[17] Christian Juelg, Andreas Hermann, Arne Roennau, and Rüdiger Dillmann. Fast online collision avoidance for mobile service robots through potential fields on 3d environment data processed on gpus. In *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 921–928, 2017.

[18] Andreas Hermann, Florian Drews, Joerg Bauer, Sebastian Klemm, Arne Roennau, and Ruediger Dillmann. Unified GPU voxel collision detection for mobile manipulation planning. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4154–4160. IEEE, 2014.

[19] Thanh-Tung Cao, Ke Tang, Anis Mohamed, and Tiow-Seng Tan. Parallel banding algorithm to compute exact distance transform with the gpu. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 83–90, 2010.

[20] Puze Liu, Kuo Zhang, Davide Tateo, Snehal Jauhri, Jan Peters, and Georgia Chalvatzaki. Regularized deep signed distance fields for reactive motion generation. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6673–6680, 2022.

[21] Kris Hauser. Semi-infinite programming for trajectory optimization with non-convex obstacles. *The International Journal of Robotics Research*, 40(10-11):1106–1122, 2021.

[22] Shan Xu, Gaofeng Li, and Jingtai Liu. Obstacle Avoidance for Manipulator with Arbitrary Arm Shape Using Signed Distance Function. In *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 343–348. IEEE, 2018.

[23] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and koray kavukcuoglu. Spatial transformer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[24] Andreas Hermann, Sebastian Klemm, Zhixing Xue, Arne Roennau, and Rüdiger Dillmann. Gpu-based real-time collision detection for motion execution in mobile manipulation planning. In *2013 16th International Conference on Advanced Robotics (ICAR)*, pages 1–7, 2013.

[25] Dominik Ernst, Georg Hager, Jonas Thies, and Gerhard Wellein. Performance engineering for real and complex tall & skinny matrix multiplication kernels on gpus. *The International Journal of High Performance Computing Applications*, 35(1):5–19, 2021.

[26] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, 8 2010.

[27] Hung Pham and Quang-Cuong Pham. A new approach to time-optimal path parameterization based on reachability analysis. *IEEE Transactions on Robotics*, 34(3):645–659, 2018.