# GPU-Accelerated Incremental Euclidean Distance Transform for Online Motion Planning of Mobile Robots

Yizhou Chen, Shupeng Lai, Jinqiang Cui, Biao Wang, and Ben M. Chen, *Fellow, IEEE*

***Abstract*—In this letter, we present a volumetric mapping system that effectively calculates Occupancy Grid Maps (OGMs) and Euclidean Distance Transforms (EDTs) with parallel computing. Unlike these mappers for high-precision structural reconstruction, our system incrementally constructs global EDT and outputs high-frequency local distance information for online robot motion planning. The proposed system receives multiple types of sensor inputs and constructs OGM without down-sampling. Using GPU programming techniques, the system quickly computes EDT in parallel within local volume. The new observation is continuously integrated into the global EDT using the parallel wavefront algorithm while preserving the historical observations. Experiments with datasets have shown that our proposed approach outperforms existing state-of-the-art robot mapping systems and is particularly suitable for mapping unexplored areas. In its actual implementations on aerial and ground vehicles, the proposed system achieves real-time performance with limited onboard computational resources.**

***Index Terms*—Mapping, motion and path planning.**

## I. Introduction

An ACCURATE representation of the local environment is critical to the effectiveness of mobile robot navigation [1]–[4]. Especially, the distance information to nearby obstacles is a must in many motion planning algorithms [5]–[7]. To that end, OGMs and EDTs are often used to bridge environment representation and planning. Both the OGM and EDT are voxel maps. In an OGM, each voxel records the probability of being occupied by an obstacle. An EDT records the distance information from each voxel to the nearest obstacle. Typically, a local EDT is required to be updated at high frequency for real-time performance of the local planner, while a global EDT is queried less frequently by the global planner.

Creating these maps is challenging due to limited onboard computational resources and real-time requirements of motion planning. To solve this problem, many researchers (see, e.g., [8]–[11]) choose to sacrifice precision for increased runtime speed. However, their wavefront-based methods can incur large errors in real-world scenarios because the distance values only propagate on observed voxels. Besides, computational resources are highly stretched when updating global EDT with a lightweight onboard computing unit. As the demand for a faster update procedure of global incremental EDT rises, it is natural to explore the possibility of computing OGM and EDT in a parallel fashion using GPU onboard. Unfortunately, most of the methods designed for CPUs reported in the literature have very limited parallelization capability.

We also note that various approaches to distance transformation have been studied in the pattern recognition society (see, e.g., [12]–[14]). These methods are mainly developed for fast distance transformation on a fully known binary map, usually a 2D image. However, in robotic applications, the environment map needs to be incremental as the robot moves, while the standard sensors of mobile robots have a limited sensing range. Moreover, the size of the map cannot be determined in advance for situations in unknown environments.

Motivated by the fast demand in real-time applications for small robots, such as small UAVs and UGVs, we propose in this letter an innovative and effective approach to increase the efficiency and accuracy of the mapping process through massive-scale parallelization. It is designed for the navigation purposes of a mobile robot and allows an incremental map to be constructed with dynamically adjustable map size. The proposed mapping system is versatile enough to construct OGMs from commonly used onboard sensors such as depth cameras and 2D- or 3D-LiDAR. It computes a fast and exact batch EDT with the parallel Euclidean Distance Transform. The batch EDT is then integrated into the global EDT through a parallel wavefront mechanism. With the utilization of parallel computing techniques, both the local and global EDT can be updated in real-time. The proposed system can be deployed on miniature computing devices with GPU, such as Nvidia Xavier NX, commonly used for small

Manuscript received January 23, 2022; accepted May 8, 2022. Date of publication May 25, 2022; date of current version June 8, 2022. This letter was recommended for publication by Associate Editor M. Popovic and Editor J. Civera upon evaluation of the reviewers' comments. This work was supported in part by the Research Grants Council of Hong Kong SAR under Grant 14209020, and in part by the Hong Kong Centre for Logistics Robotics. *(Corresponding author: Shupeng Lai.)*

Yizhou Chen and Ben M. Chen are with the Department of Mechanical and Automation Engineering, Chinese University of Hong Kong, Hong Kong 999077, China (e-mail: 1155141802@link.cuhk.edu.hk; bmchen@cuhk.edu.hk).

Shupeng Lai is with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 117583 (e-mail: shupenglai@gmail.com).

Jinqiang Cui is with the Peng Cheng Laboratory, Shenzhen, Guangdong 518055, China (e-mail: cuijq@pcl.ac.cn).

Biao Wang is with the College of Automation Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu 210095, China, and also with Peng Cheng Laboratory, Shenzhen, Guangdong 518055, China (e-mail: wangbiao@nuaa.edu.cn).

Digital Object Identifier 10.1109/LRA.2022.3177852

robots or unmanned systems. We will make the source code of our system available as an open-source project.[1]

The contributions of this work are: i) the development of an efficient approach that generates OGM and EDT through massive parallelization; ii) the design of an EDT integration method that allows the incremental construction of global EDT with dynamically adjustable map size; and lastly iii) the integration of the proposed algorithm in various ground/aerial mobile robots to demonstrate its effectiveness with onboard motion planning.

The rest of this letter is organized as follows: In Section II, we recall some background materials in fast EDT algorithms and state-of-the-art mappers. In Section III, we present the overview and framework of our proposed mapping system. The technical details of the proposed system are given in Section IV. The experimental and simulation results that demonstrate the effectiveness of our system are presented in Section V, while the result conducted on robotic platforms is given in Section VI. Finally, we draw some concluding remarks in Section VII.

## II. Related Work and Background

In occupancy maps, such as OGM or Octomap, they are usually discretized into voxel grids that store occupancy information. Recently, some researchers (see, e.g., [15], [16]) use occupancy maps to perform surface reconstruction and motion planning at the same time. They have mainly focused on high-resolution space representation rather than fast onboard distance checking. Although the constructed maps can be used in collision checking in corridor-based planners, information on the distance to obstacles is generally not captured.

For many optimization-based motion planning algorithms, it is necessary to query the distance between the trajectory and its nearest obstacle. The demand of multiple queries in each planning iteration dramatically affects the overall efficiency of the planning algorithm. One common approach is to pre-compute an EDT map such that the query process can be executed in $O(1)$ time. For robots, such as miniature aerial vehicles (MAVs), the onboard computers generally have limited computational power due to the very limited payload of the vehicle. The demand on real-time EDT generation has forced to exclude EDT mappers in the process (see, e.g., [8], [17]). Similarly, researchers in [5], [6] only maintain a local and memoryless EDT to represent the environment around the vehicle, whereas obstacles outside the local area are ignored. Also, the works in [10], [11] construct incremental EDT online by careful implementation of dedicated data structures. More specifically, the method, Voxblox in [10] builds EDTs out of the so-called truncated signed distance fields (TSDFs), which contains projected distance information within a truncation radius near the surface. They utilized the distance to propagate throughout the whole space in a quasi-Euclidean metric and finally obtain an EDT map. However, the resulting EDT map has two sources of errors, which would accumulate through the propagation. On the other hand, the approach, FIESTA in [11], uses double-linked lists (DLLs) to record distance information and improve both mapping precision and efficiency. Although their methods have been implemented on mobile robots with vision-based sensors, the computation cost is extremely high when maintaining a global EDT in large areas.

[1][Online]. Available: https://github.com/JINXER000/GIE-mapping

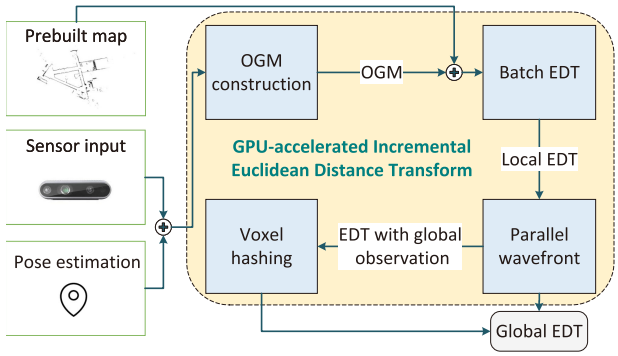



Fig. 1. Framework of the proposed system.

Many studies have been carried out in computational geometry and pattern recognition society in developing fast and exact distance transform algorithms. Saito and Toriwaki [12] devise an algorithm to produce EDT by independently scanning on each dimension. Meijster *et al.* [13] follow the concept of [12] to enhance the overall performance by additionally utilizing properties of parabola intersections. These decomposition-based distance transform algorithms break down the whole mapping procedure into strips or structuring functions, which is suitable for parallel computing. However, these methods compute EDT in a batch of fixed-size memory, which is often not the case in robotics applications. To address these issues, we employ the technique of voxel hashing [18] to store the mapping history and further utilize parallel wavefront to maintain the consistency between the new incoming data and the existing map. Finally, we would like to note that our previous work, i.e., the technique, GPU-LO in [19], only considers the limited observation problem during the mapping process. We propose in this letter more wavefronts in the system proposed to tackle problems associated with real-world mapping tasks.

## III. Overview of the Proposed System Framework

In what follows, we present a framework, depicted in Fig. 1, for generating OGM and EDT through massive parallelization, which can be divided into three stages: OGM construction, batch EDT, and global EDT integration. In the OGM construction stage, the system intakes data from the sensors and builds the OGM. Position and attitude information are usually estimated by localization algorithms, while range measurement can be obtained from onboard range sensors. The range measurement is then integrated into the global OGM as occupancy probabilities by parallel implementations of ray casting and volumetric projection.

In the batch EDT stage, the parallel EDT algorithm transforms the OGM in the euclidean distance metric. Domain knowledge, such as pre-built OGM or pre-defined forbidden zones, can also be incorporated into the EDT process. The algorithm scans each dimension in turn and calculates the distance value for each voxel.

Finally, in the global EDT integration stage, the EDT in the local area is integrated into the hash table that stores global EDT. After identifying the proper sources, the parallel wavefront is utilized to propagate actual distance values over both the local and global EDT. Hence the batch EDT is updated with global observation, and the history can be preserved in the hash table.

## IV. Algorithms and the Pipeline

We present in the following the detailed algorithms for constructing OGM, batch EDT and global EDT integration.

### A. OGM Construction

Two GPU-accelerated algorithms for OGM construction are developed. The first one is a massively-parallel version of the traditional ray casting algorithm [20], while the second one is a parallel volumetric projection method utilizing the sensor model [21]. Each voxel of OGM contains its occupancy probability and its visibility information.

In ray casting, given a pixel at entry $(i, j)$ of a depth image, the algorithm first transforms the pixel to a 3D voxel $\mathcal{V}_{x,y,z}$:

$$\mathcal{V}_{x,y,z} = \mathcal{T}^{-1} K^{-1} \mathcal{D}_{i,j,w}, \tag{1}$$

where $\mathcal{T}$ is the transformation from the world frame to the sensor frame, $K$ is the intrinsic parameters of the sensor, and $\mathcal{D}_{i,j,w}$ is the measurement at $(i, j)$ with depth $w$. Rays are then cast from the sensor to measured obstacle points and update the occupancy probability of each voxel along the line. Denoting the number of points perceived as $n$, the average travel distance as $l$, and the voxel size as $q$, the computational complexity of ray casting process is in the order $O(nl/q)$. One most significant challenge to parallelize ray casting is that multiple rays might try to write to the same voxel with different data, leading to serious synchronization issues of possible memory conflict. We resolve this problem using CUDA-atomic operations, which forces I/O operations to be mutually exclusive.

On the other hand, we adopt the volumetric projection method of [21], which is developed to construct TSDF using optical triangulation scanners, to build OGMs with sensors such as 2D- or 3D-LiDAR. Instead of casting rays from pixels in the depth image to the 3D space, the projection computes the appropriate sensor value of each voxel in the local volume by simulating the acquisition process according to the sensor model. Assuming the range measurement data to be two-dimensional, the projection from the voxel $(x, y, z)$ to the sensor data entry $(i, j)$ can be formulated as:

$$\mathcal{D}_{i,j,\bar{w}} = K \mathcal{T} \mathcal{V}_{x,y,z}, \tag{2}$$

where $\mathcal{D}_{i,j,\bar{w}}$ is the projection onto the sensor with the projected range $\bar{w}$. Thereafter, by comparing the projected range $\bar{w}$ with the measured range $w$, occupancy information can be updated. The detailed procedure can be found in our previous work [19]. In volumetric projection, the computational complexity is proportional to the volume size, i.e., $O(N)$, if there are $N$ voxels within the local volume. As $N$ is generally huge in many applications, such an approach is rarely used in CPU. It can be massively parallelized, however, in GPU without CUDA-atomic operations.

The choice among various OGM construction methods depends heavily on input data. For example, during the construction of point cloud, some sensors choose not to generate a point if the corresponding ray hits no object within the detection range. If a ray casting method is applied directly, the volume covered by such a ray will not be updated as known and obstacle-free. Besides, when the resolution of the range sensor is extremely high, a down-sampling procedure is required to speed up ray casting. As such, we give preference to volumetric projection if intrinsic parameters of the range sensor can be obtained.

**Algorithm 1:** BATCHEDT.

**Input:** OGM $O$, dimension of OGM $dim$
**Output:** Batch EDT $E$ in the local volume and nearest obstacles recording $P$

$d \leftarrow 1$;
$E, P \leftarrow \mathsf{INITSCAN}(O)$;
$E \leftarrow \mathsf{TRANSPOSE}(E, d)$;
**while** $d \leq dim$ **do**
  $d \leftarrow d + 1$;
  $E, P \leftarrow \mathsf{AUXSCAN}(E, P)$;
  $E \leftarrow \mathsf{TRANSPOSE}(E, d)$;
$E \leftarrow \mathsf{CHECKVALID}(E, O)$;
**return** $E, P$

### B. Batch EDT

Unlike the existing EDT construction methods of [9]–[11], which propagate distance from newly observed voxels in an ordered fashion, our method first evaluates EDT in the local volume with a GPU-based scanning technique without introducing propagation error. The parallel batch EDT workflow is given in Algorithm 1, which consists of one initial scan and several auxiliary scans. We extend the scanning method of Meijster *et al.* [13], which performs exact EDT in linear time, to the 3D space. It decomposes memory into independent lines and can therefore be parallelized. In fact, any other EDT algorithm with similar attributes (see, e.g., [14]) can be applied in our framework. In addition to calculating distance values, our algorithm tracks the nearest obstacle of each voxel during scans, which is essential for subsequent wavefront propagation.

In the case of computing a 3D EDT, our objective is to find the nearest obstacle $(i, j, k)$ with respect to each voxel $(x, y, z)$. In the initial scanning INITSCAN(), distance values of all occupied voxels are set as 0, while those of empty voxels are set as *inf*. When generating EDT in 1D, each column is required to be swept forward and backward to compute each grid of its distance to the nearest obstacle within the same column. The sweep always starts with a value of 0 at the obstacle and then increments in empty areas. This procedure can be described as:

$$G(x, y, z) = \min_{j \in rows} \{|j - y| \,|\, G(x, j, z) = 0\}, \tag{3}$$

where $G(x, y, z)$ is the distance value of voxel $\mathcal{V}_{x,y,z}$ when scanning the first dimension.

Fig. 2 illustrates the auxiliary scan on the X-Y-plane. Every thread In the auxiliary scan on the X-Y-plane, every thread

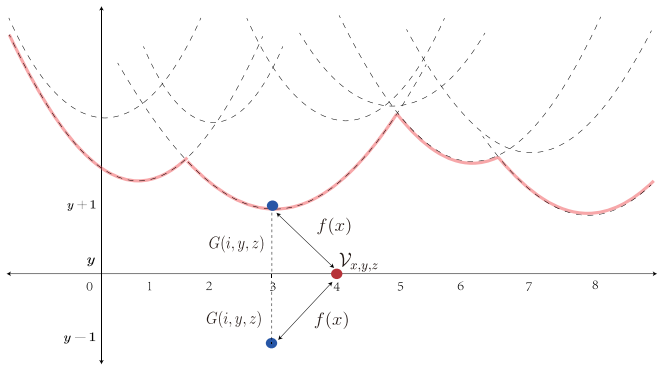



Fig. 2. An illustration of distance functions in AUXSCAN(). When a voxel $\mathcal{V}_{x,y,z}$ (the red dot) is inspected by the current thread, it will be assigned to one parabola segment given the lower envelope (pink curve). The blue dots denote the possible positions of the nearest obstacle of $\mathcal{V}_{x,y,z}$.

executes a sweeping process along X-axis while fixing the value on Y- and Z-axes. Assuming the nearest obstacle of voxel $\mathcal{V}_{x,y,z}$ lies on the column $C_i$, the distance function of its $x$ value can be written as

$$f(x) = (x - i)^2 + G(i, y, z)^2, \tag{4}$$

where $G(i, y, z)$ is regarded as a constant since it is already calculated at INITSCAN(). The distance function of each voxel can be represented as a parabola shifted by its $G$ value (as shown in Fig. 2). Essentially, the calculation of distance values is to find the lower envelope of the union of all parabolas, which can be formulated as

$$F(x, y, z) = \min_{i \in cols} \{(x - i)^2 + G(i, y, z)^2\}, \tag{5}$$

where $cols$ denotes all columns of the map in the X-Y-plane. When traversing the row, parabolas are inserted one by one. If the incoming parabola intersects the current lower envelope, a new parabola segment will be created, and the lower envelope will be updated. We refer readers to [13] for details. After calculating distance values on the X-Y plane, the function AUXSCAN() can then be adopted to progress on Z-axis in a similar fashion. Finally, the nearest obstacles in the entire 3D space are captured.

Lastly, each scan is to be followed by a TRANSPOSE() function. We utilize a tensor transpose library, cuTT [22], to reorder the memory of the local volume so that coalescing memory access can be achieved. After all scans, each voxel of the EDT goes through validation CHECKVALID(). Note that during the OGM construction, voxels are divided into three categories: free, occupied, and unknown. A voxel will be further integrated into the global EDT only if it is ever observed by the robot.

### C. Global EDT Integration

With the batch EDT process, the observable region in the local volume is filled with EDT at each timestamp. However, the mapping accuracy is greatly affected by dynamic obstacles, sensor noise, and more severe, limited observation, which can be illustrated in Fig. 3. More specifically, Fig. 3(a) depicts an MAV flying in 2D grids. When computing batch EDT in the local volume (green dotted-line box), the map only reveals the

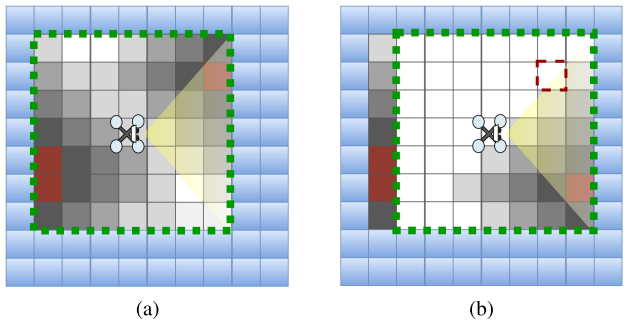



Fig. 3. Limited observation and dynamic obstacle problems. Batch EDT is computed within green dotted line. The obstacles are painted in the dark red grids. The color shade in the observed space represents the distance value. The blue grids are unobserved area.

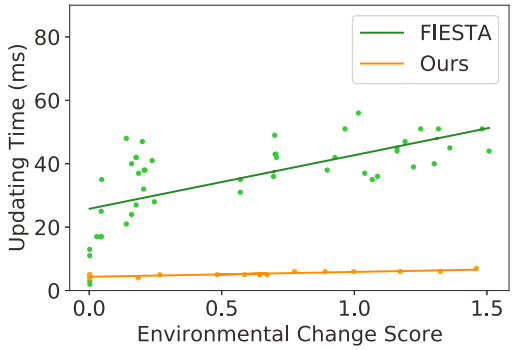



Fig. 4. EDT operation speed with different environmental change score.

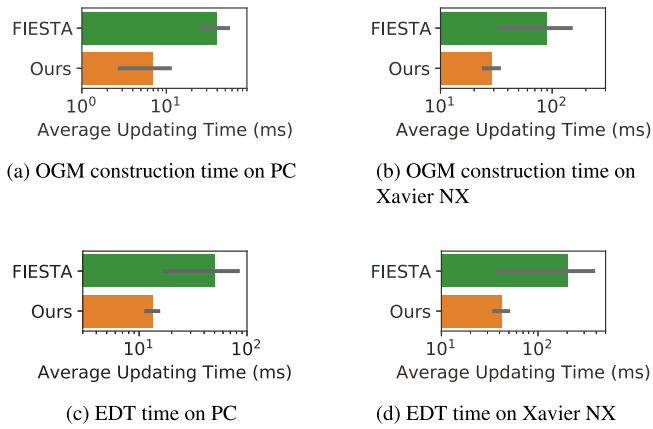
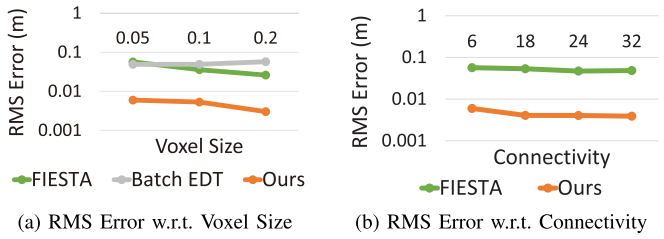


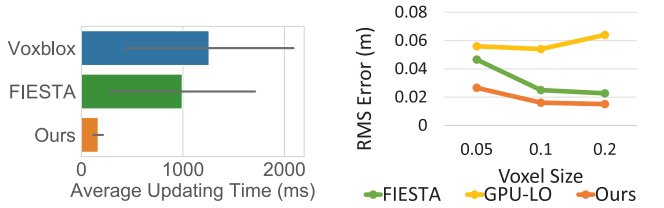

Fig. 5. The mapping results of different mappers on different platforms. The bold line in each center of the bar is the standard deviation of time throughout the run. (a) OGM construction time on PC (b) OGM construction time on Xavier NX (c) EDT time on PC (b) EDT time on Xavier NX.

information inside the local volume and has no knowledge about sensed regions in history. In Fig. 3(b), as the vehicle moves to the right, the obstacles at the left boundary are no longer inside the updating range and are ignored in the batch EDT process. In consequence, inconsistencies between the global EDT and the current local EDT occur. Besides, dynamic obstacles can also pop up (the light salmon grid at lower right) and disappear (the grid with red dashed line at upper right) in the field of view.



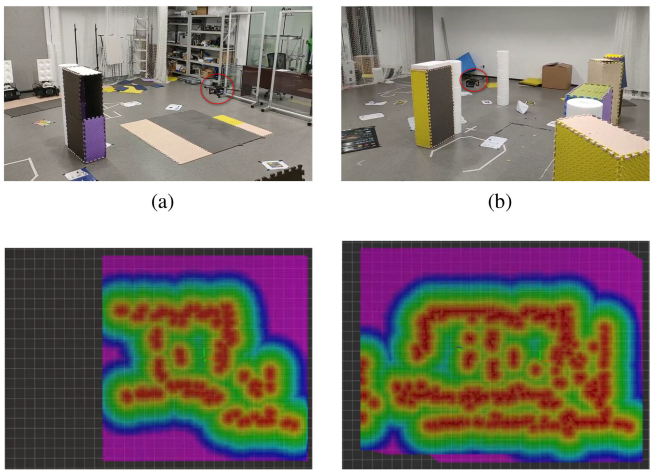

(a) RMS Error w.r.t. Voxel Size (b) RMS Error w.r.t. Connectivity

Fig. 6. The RMS error in different parameter settings. (a) RMS Error w.r.t. Voxel Size (b) RMS Error w.r.t. Connectivity.




(a) Global EDT time on Xavier NX (b) Global RMS Error w.r.t. Voxel Size

Fig. 7. The average speed and RMS error measured in the entire map. (a) Global EDT time on Xavier NX (b) Global RMS Error w.r.t. Voxel Size.



(a) (b) (c) (d)

Fig. 8. The snapshots and the constructed maps when an MAV flying through obstacles. The movement is limited to a 2D plane since only a 2D LiDAR is used.

To tackle the above problems, we combine the batch EDT with our global integration algorithm. Essentially, this algorithm extract frontiers (Algorithm 2), and then perform parallel wavefront (Algorithm 3) separately. Afterwards, the updated EDT in the current local volume is stored in a GPU hash table [18], which aims to make the mapping spatial efficient. The updated portion of hash table on GPU is streamed to CPU; hence the global EDT on both CPU and GPU are continuously constructed in the run time.

The source extraction function GETSOURCE() of Algorithm 2 is explained in detail. Line 2 inspects each voxel *cur* that lies on the boundary of the batch EDT and its neighbor *nbr* that lies outside the local volume. If the nearest obstacle

**Algorithm 2:** GETSOURCE.

**Input:** Current local EDT $E$, global EDT stored in GPU hash table $GHash$
**Output:** Initial frontiers of propagation

1 $frontierA, frontierB, frontierC \leftarrow \varnothing$
2 **forall** $cur \in E.boundary$ **do**
3  **forall** $nbr \in \{x | x \in cur.nbrs \wedge x \notin E.volume\}$ **do**
4  **if** $||cur.parent - nbr||_2 > \mathsf{DIST}(GHash, nbr) \wedge$ $nbr.parent \in E.volume$ **then**
5  $nbr.raise \leftarrow$ true
6  $frontierA \leftarrow frontierA \cup \{nbr\}$
7  **else if** $||cur.parent - nbr||_2 < \mathsf{DIST}(GHash, nbr)$ **then**
8  $frontierB \leftarrow frontierB \cup \{nbr\}$
9  **if** $||nbr.parent - cur||_2 < \mathsf{DIST}(E, cur) \wedge$ $nbr.parent \notin E.volume$ **then**
10  $frontierC \leftarrow frontierC \cup \{cur\}$
11 **return** $frontierA, frontierB, frontierC$

**Algorithm 3:** PARWAVEFRONT (raise).

**Input:** Initial frontiers $frontierA$, $frontiersB$, current local EDT $E$, and global EDT in GPU hash table $GHash$
**Output:** Updated local EDT and global EDT

1 **while** $frontierA.size \neq 0$ **do**
2  CHOOSELEVEL($frontierA$)
3  **forall** $cur \in frontierA$ **do**
4  $frontierA \leftarrow frontierA \backslash \{cur\}$
5  **forall** $nbr \in \{x | x \in cur.nbrs \wedge x.raise =$ false$\}$ **do**
6  **if** $nbr.parent \in$ $E.volume \wedge \mathsf{DIST}(E, nbr.parent) \neq 0$ **then**
7  $\mathsf{DIST}(GHash, nbr) \leftarrow ||cur.parent - nbr||_2$
8  $nbr.parent \leftarrow cur.parent$
9  $nbr.raise \leftarrow$ true
10  $frontierA \leftarrow frontierA \cup \{nbr\}$
11  **else if** $\mathsf{DIST}(GHash, cur) > ||nbr.parent - cur||_2$ **then**
12  $frontierB \leftarrow frontierB \cup \{cur\}$
13 **return** $E, GHash$

*nbr.parent* is inside the local volume, but the distance value of the neighbor in GPU hash table DIST($GHash, nbr$) can be raised by one obstacle inside the local volume, it implies that *nbr.parent* disappears in the robot's field of view(Line 6). Otherwise, if an obstacle suddenly pops up in the new observation, it will lower the distance value of *nbr* (Line 8). If the nearest obstacle *nbr.parent*, which is outside the local volume, can lower the distance value inside the local volume $\mathsf{DIST}(E, cur)$, then *cur* is appended to the frontier corresponding to the limited observation situation (Line 10).

Subsequently, the parallel wavefront function PARWAVEFRONT() in Algorithm 3 propagates the accurate distance from the frontiers to both inside and outside the local volume. The essential problem of conducting propagation in parallel is that GPU implementations of wavefront algorithms [23], [24] are inherently computationally inefficient and can even be slower than traditional CPU-based algorithms. We tackle this problem by extending hierarchical frontiers [25], which originally solves single-source shortest path problems, to our wavefront context, where a voxel can be visited more than once in one or more iterations. Algorithm 3 shows the details of the parallel wavefront propagation to raise the distance value caused

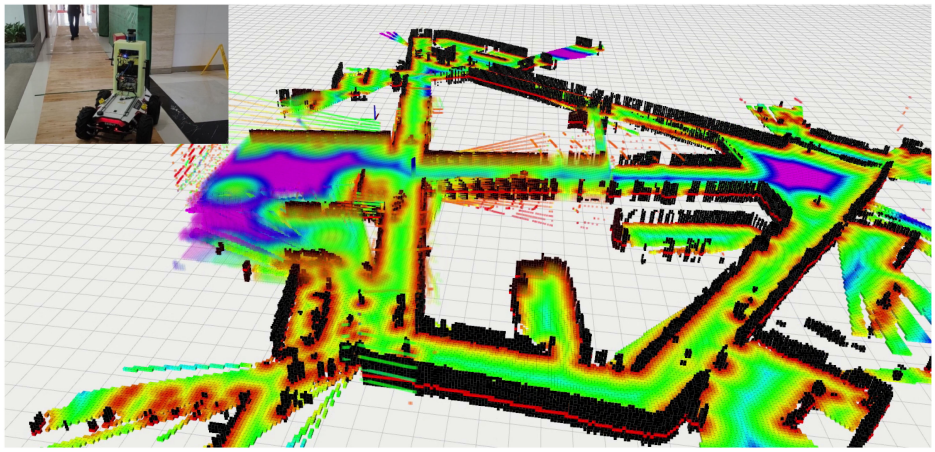


Fig. 9. OGM and EDT constructed by a UGV in narrow corridors.

by vanished obstacles. In the function CHOOSELEVEL(), the frontier levels are changed in accordance with the number of elements of the frontier. If it exceeds the maximum size of a GPU thread-block, the frontier is stored on the global memory for multiple thread-blocks to access. Otherwise, they are placed in the shared memory of one thread-block, which is faster than on the global memory. In Line 4, each cell *cur* of the frontier is placed on a single thread. Starting at Line 5, we enumerate all the neighbors of *cur* whose distance value is not raised. If the nearest obstacle of *nbr* is found disappeared inside the local volume, *nbr* will be propagated by *cur* and be inserted back into *frontierA*. Otherwise, in Line 11, if the current voxel encounters a neighbor that has a closer obstacle to *cur*, the algorithm adds *cur* to another frontier *frontierB* for a lowering wavefront. The algorithm for lowering wavefront is similar to Algorithm 3 except that it propagates the minimum distance value to the neighborhood. Following the wavefront processing *frontierB* outside the local volume, the wavefront of *frontierC* begins to propagate global observation into the local volume. We omit this algorithm in the paper because of page limitation. Interested readers are referred to the detailed algorithm appended in the supplementary material.[2] Finally, the propagation ends once all frontiers become empty.

## V. Experimental Results

In this section, we conduct a series of thorough tests on three datasets: the UAV-pillar dataset, the UGV-Corridor dataset, and the Cow-and-Lady dataset. The UAV-Pillar dataset is synthesized in the Gazebo simulator, where an MAV is flying through pillars at a constant speed, gathering range information with an onboard 3D LiDAR. The UGV-Corridor dataset is recorded in a real-world environment with long and narrow corridors, where a UGV is equipped with a 16-line Velodyne LiDAR and drives through pedestrians. The Cow-and-Lady dataset is presented with the work Voxblox [10], and its data is collected using a depth camera in a small room.

[2][Online]. Available: https://github.com/JINXER000/GIE-mapping/blob/main/doc/sup-mat.pdf

### A. Simulation in an Unexplored Environment

While exploring an unknown space, the robot updates its map with continuous sensor data stream. The degree of change in the sensed environment also affects the computational efficiency of the mapping process. We demonstrate that the speed of our method is stable while exploring an unknown environment.

To quantify the difference between two sensed environments, we define the change score $V$ as:

$$V(t_1, t_2) = \mathop{\mathbb{E}}_{i \in S}(|E_{t_1}(i) - E_{t_2}(i)|), \tag{6}$$

where $S$ is the set of all grids in the map, $i$ is the index of the grid, and $E_{t_1}, E_{t_2}$ are respectively the ground truth EDTs at two distinct timestamps $t_1, t_2$ over the global range. The unknown volume is treated as empty during ground truth EDT calculation. If there is no difference between the sensed environment at $t_1$ and $t_2$, the score will be 0. The more the sensed environment changes, the higher the score is. The EDT updating speed versus the environment change score can be found in Fig. 4, where the resolution is set as 0.2 m and the local update range is 20 m $\times$ 20 m $\times$ 6 m. The figure shows that the EDT construction time continues to increase as the environmental change score increases. However, the construction speed of our mapper remains almost constant. This is because the traditional mappers construct EDT purely with wave propagation and rely more on existing EDT.

### B. Experiments on Real-World Datasets

We test each stage of our system separately to show how differences in techniques that affect performance. First, we show the necessity of transpose operation in the batch EDT stage. The coalescing pattern is realized by forcing each thread to access adjacent memory, and scanning is significantly accelerated. In this experiment, we test the UGV-Corridor dataset on an Xavier NX. Table I shows the difference between the approaches with and without the coalesced memory access. It is observed that the parallel EDT operation is much faster in the coalescing case.

TABLE I
BATCH EDT TIME WITH AND WITHOUT COALESCED MEMORY ACCESS

| Method | Average Updating Time (ms) |
|---|---|
| Original | 46 |
| Coalescing | **19** |

TABLE II
GLOBAL EDT INTEGRATION TIME WITH DIFFERENT PROPAGATION ALGORITHMS

| Method | Average Updating Time (ms) |
|---|---|
| CPU bucket heap | 790 |
| GPU bucket heap | 2035 |
| GPU priority queue | 3020 |
| GPU Bellman-Ford | 250 |
| GPU Prefix Sum BFS | 22 |
| Ours | **10** |

Apart from the global EDT integration technique proposed in Section IV-C, we evaluate the methods on the Cow-and-Lady dataset with different graph traversal algorithms. The experiments are done on a PC with Intel i5-8300H CPU and a GTX1060 GPU, mapping 6 m $\times$ 6 m $\times$ 3 m local range in 0.05 m resolution. In our method, the GPU thread-block limit is set as 512, and 6-connectivity is used. The results of average updating time are shown in Table II. The baseline solution using dynamic brushfire on CPU with the bucket-heap data structure is adopted, which is similar to the implementation by Lau *et al.* [9]. This cache-oblivious data structure minimizes the I/O operations in theory. Following the work of [26], we have also implemented a GPU version[3] of the bucket heap, aiming to speed up the computation. However, it turns out that this data structure is not suitable for GPU since its performance is even worse than that of the CPU bucket heap. In contrast, our proposed method is cache-aware and we have the freedom to decide the parameters catering to different computing platforms. Other technqiues included for comparison with our works are the parallel Bellman-Ford method [27], the BFS (breadth-first search) method accelerated by prefix-sum,[4] and the algorithm with GPU pripority queue.[5]

The following experiments are conducted to show that our method outperforms the state-of-the-art EDT mapper, FIESTA [11]. All comparisons are done with the UGV-Corridor dataset, and the local volume is set to 10 m $\times$ 10 m $\times$ 1.2 m. For results on the Cow-and-Lady dataset, we refer interested readers to the supplemented video[6] [19].

First, we evaluate the performance of mappers on different computing platforms, i.e., a personal computer (with Intel i5-8300H CPU and a GTX1060 GPU) and an Nvidia Xavier NX. Since high-frequency local planning is usually limited within a local volume, the CPU-GPU stream is turned off in our method, while the accurate EDT inside the local volume is copied to CPU and broadcast by ROS (although the global EDT in our method can be directly accessed by a GPU-based motion planner [28]). In FIESTA, we simply disable the *global_update* option to update its EDT only within the local volume, which makes the condition work in its favor. The frequency of updating the EDT is 2 Hz, and the resolution is 0.05 m. The elapsed time of constructing OGM and EDT are measured separately, and the results are plotted in bar charts as shown in Fig. 5. Obviously, our method outperforms FIESTA in speed on both platforms. Furthermore, as discussed in Section V-A, the average EDT time of our method is expected to be less volatile.

The second set of experiments is performed to investigate the extent to which hyperparameters, such as voxel size and connectivity, affect the accuracy of the EDT within the local volume. The results are shown in Fig. 6. The mappers in this experiment take measurements from the UGV-Corridor dataset, construct their own OGM and EDT, and then calculate the RMS error of the EDT against ground truth. In the ground truth calculation, a K-D tree of the current OGM constructed by each mapper is first created. The nearest neighbor of each voxel is then found and the least distance to the obstacles is computed. Note that we only focus on the error induced by the EDT computation and do not include the error caused by the discretization of OGM. In Fig. 6(a), all samples are tested with 6-connected in propagation. Since FIESTA is more accurate than Voxblox [10] according to Han *et al.* [11], we only compare our method with FIESTA. In both FIESTA and our method, global update is enabled while only the error in the local volume is taken into consideration. Also, the RMS error curve of the batch EDT version of our mapper is plotted to show the necessity of global EDT integration. We can see that the resolution and the RMS error are negatively correlated. The mapping accuracy with respect to connectivity is shown in Fig. 6(b), where the resolution is fixed as 0.05 m. As the connectivity increases, the RMS error drops. From similar results in FIESTA, a likely explanation is that the error is mainly incurred by wavefront propagation. When the connectivity remains constant, as the number of voxels increases, error cases in the propagation has a larger chance to occur. On the other hand, as more neighbors are considered, the propagation error could be reduced. We refer interested readers to [29] for a more descriptive explanation. Compared to FIESTA, our mapper is more accurate in any situation because the exactness of our batch EDT method helps to reduce the error.

In addition to the experiments in the local volume, we compared the mapping performance with Voxblox [10], FIESTA [11], and GPU-LO [19] in terms of global EDT. From Fig 7(a), although maintaining the global EDT on both GPU and CPU causes extra overhead in our method, it is negligible compared to the drastically increasing load in FIESTA. Similarly, as shown in Fig 7(b), when measuring RMS error over the whole map, the newly proposed technique has outperformed FIESTA and GPU-LO. The results assure us that our system suffices to provide global EDT for online global motion planners.

## VI. IMPLEMENTATION ON ACTUAL ROBOTIC PLATFORMS

To verify that the entire mapping system can run on lightweight onboard computers, we conduct an experiment with

[3][Online]. Available: https://github.com/JINXER000/parHeap
[4][Online]. Available: https://github.com/rafalk342/bfs-cuda
[5][Online]. Available: https://github.com/crosetto/cupq
[6][Online]. Available: https://github.com/JINXER000/GIE-mapping

an MAV navigating in an unexplored and cluttered indoor environment. The motion planner is taken directly from [30], which requires distance information from EDT. The range measurement comes from a 2D LiDAR mounted on the top of the MAV. An Intel Real-sense T-265 tracking camera is used to estimate the current pose of the MAV. All computations are done on an onboard Nvidia Xavier NX. In Fig. 8, the MAV flies to an unmapped region while avoiding obstacles. The mapper updates EDT at 10 Hz within 20 m $\times$ 20 m with 0.1 m voxel width, allowing the motion planner to obtain the required distance information in real-time.

Lastly, Fig. 9 depicts a global EDT map constructed using a UGV equipped with a 16-line LiDAR. The experiment was carried out in an indoor environment with narrow corridors. We set the resolution as 0.1 m while the local volume is 20 m $\times$ 20 m$\times$ 1.2 m. Using a computer with Intel i5 9300H CPU and Nvidia 1660Ti GPU, the vehicle navigates fully autonomously through walking pedestrians. The results show that our proposed technique is efficient and can be readily adopted for real applications.

## VII. Conclusion

In this letter, we have proposed a novel robotic volumetric mapping framework with parallel computing. the proposed system is capable of generating real-time OGMs and EDTs that can be used for rapid motion planning. The effectiveness and the efficiency of our proposed technique has been proved by various experiments in simulations, datasets, and the real world problems. With the deployment on the onboard computing devices of both MAV and UGV, we have demonstrated that our system can efficiently build incremental EDT for mobile robots to navigate in unexplored environments.

## References


[1] M. Lan, S. Lai, T. H. Lee, and B. M. Chen, "A survey of motion and task planning techniques for unmanned multicopter systems," *Unmanned Syst.*, vol. 9, no. 2, pp. 165–198, 2021.

[2] M. Mondal and S. Poslavskiy, "Offline navigation (homing) of aerial vehicles (quadcopters) in GPS denied environments," *Unmanned Syst.*, vol. 9, no. 2, pp. 119–127, 2021.

[3] S. Yuan, H. Wang, and L. Xie, "Survey on localization systems and algorithms for unmanned systems," *Unmanned Syst.*, vol. 9, no. 2, pp. 129–163, 2021.

[4] J. Q. Cui *et al.*, "Search and rescue using multiple drones in post-disaster situation," *Unmanned Syst.*, vol. 4, no. 1, pp. 83–96, 2016.

[5] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2009, pp. 489–494.

[6] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robot. Automat. Lett.*, vol. 4, no. 4, pp. 3529–3536, Oct. 2019.

[7] P. Arora and C. Papachristos, "Environment reconfiguration planning for autonomous robotic manipulation to overcome mobility constraints," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 2352–2358.

[8] R. Javis, "Distance transform based path planning for robot navigation," in *Recent Trends in Mobile Robots*, Y. F. Zureik, Ed., Singapore: World Scientific, 1994, pp. 3–31.

[9] B. Lau, C. Sprunk, and W. Burgard, "Improved updating of Euclidean distance maps and Voronoi diagrams," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2010, pp. 281–286.

[10] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3D Euclidean signed distance fields for on-board MAV planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 1366–1373.

[11] L. Han, F. Gao, B. Zhou, and S. Shen, "FIESTA: Fast incremental Euclidean distance fields for online motion planning of aerial robots," in *Proc. IEEE Int. Conf. Intell. Robots Syst. (IROS)*, Macau, China, 2019, pp. 4423–4430.

[12] T. Saito and J.-I. Toriwaki, "New algorithms for Euclidean distance transformation of an n-dimensional digitized picture with applications," *Pattern Recognit.*, vol. 27, no. 11, pp. 1551–1565, 1994.

[13] A. Meijster, J. B. Roerdink, and W. H. Hesselink, "A general algorithm for computing distance transforms in linear time," in *Mathematical Morphology and Its Applications to Image and Signal Processing*, Berlin, Germany: Springer, 2002, pp. 331–340.

[14] T. T. Cao, K. Tang, A. Mohamed, and T. S. Tan, "Parallel banding algorithm to compute exact distance transform with the GPU," in *Proc. ACM SIGGRAPH Symp. Interactive 3D Graph. Games*, 2010, pp. 83–90.

[15] E. Vespa, N. Nikolov, M. Grimm, L. Nardi, P. H. Kelly, and S. Leutenegger, "Efficient octree-based volumetric SLAM supporting signed-distance and occupancy mapping," *IEEE Robot. Automat. Lett.*, vol. 3, no. 2, pp. 1144–1151, Apr. 2018.

[16] N. Funk, J. Tarrio, S. Papatheodorou, M. Popović, P. F. Alcantarilla, and S. Leutenegger, "Multi-resolution 3D mapping with explicit free space representation for fast and accurate mobile robot motion planning," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 3553–3560, Apr. 2021.

[17] L. Wu, K. M. B. Lee, L. Liu, and T. Vidal-Calleja, "Faithful Euclidean distance field from log-Gaussian process implicit surfaces," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 2461–2468, Apr. 2021.

[18] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3D reconstruction at scale using voxel hashing," *ACM Trans. Graph.*, vol. 32, no. 6, pp. 1–11, 2013.

[19] Y. Chen, S. Lai, B. Wang, F. Lin, and B. M. Chen, "A GPU mapping system for real-time robot motion planning," in *Proc. IEEE Int. Conf. Real-time Comput. Robot.*, 2021, pp. 762–768.

[20] J. Amanatides *et al.*, "A fast voxel traversal algorithm for ray tracing," *Eurographics*, vol. 87, no. 3, pp. 3–10, 1987.

[21] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proc. 23rd Annu. Conf. Comput. Graph. Interactive Techn.*, 1996, pp. 303–312.

[22] A.-P. Hynninen and D. I. Lyakh, "cuTT: A high-performance tensor transpose library for CUDA compatible GPUs," 2017, *arXiv:1705.01598*.

[23] P. Harish and P. J. Narayanan, "Accelerating large graph algorithms on the GPU using CUDA," in *Proc. Int. Conf. High Perform. Comput.*, 2007, pp. 197–208.

[24] Y. Deng, B. D. Wang, and S. Mu, "Taming irregular EDA applications on GPUs," in *Proc. Int. Conf. Comput. Aided Des.*, 2009, pp. 539–546.

[25] L. Luo, M. Wong, and W.-m. Hwu, "An effective GPU implementation of breadth-first search," in *Proc. IEEE Des. Automat. Conf.*, 2010, pp. 52–55.

[26] J. Iacono, B. Karsin, and N. Sitchinava, "A parallel priority queue with fast updates for GPU architectures," 2019, *arXiv:1908.09378*.

[27] F. Busato and N. Bombieri, "An efficient implementation of the Bellman-Ford algorithm for Kepler GPU architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 8, pp. 2222–2233, Aug. 2016.

[28] H. Lu, Q. Zong, S. Lai, B. Tian, and L. Xie, "Flight with limited field of view: A parallel and gradient-free strategy for micro aerial vehicle," *IEEE Trans. Ind. Electron.*, vol. 69, no. 9, pp. 9258–9267, Sep. 2022.

[29] O. Cuisenaire and B. Macq, "Fast Euclidean distance transformation by propagation using multiple neighborhoods," *Comput. Vis. Image Understanding*, vol. 76, no. 2, pp. 163–172, 1999.

[30] S. Lai, M. Lan, and B. M. Chen, "Model predictive local motion planning with boundary state constrained primitives," *IEEE Robot. Automat. Lett.*, vol. 4, no. 4, pp. 3577–3584, Oct. 2019.