Motion planning with diffusion maps

Motion Planning with Diffusion Maps

Yu Fan Chen, Shih-Yuan Liu, Miao Liu, Justin Miller, and Jonathan P. How

***Abstract*— Many robotic applications require repeated, on-demand motion planning in mapped environments. In addition, the presence of other dynamic agents, such as people, often induces frequent, dynamic changes in the environment. Having a potential function that encodes pairwise cost-to-go can be useful for improving the computational speed of finding feasible paths, and for guiding local searches around dynamic obstacles. However, since storing pairwise potential can be impractical given the $O(|V|^2)$ memory requirement, existing work often needs to compute a potential function for each query to a new goal, which would require a substantial online computation. This work addresses the problem by using diffusion maps, a machine learning algorithm, to learn the map's geometry and develop a memory-efficient parametrization $(O(|V|))$ of pairwise potentials. Specially, each state in the map is transformed to a diffusion coordinate, in which pairwise Euclidean distance is shown to be a meaningful similarity metric. We develop diffusion-based motion planning algorithms and, through extensive numerical evaluation, show that the proposed algorithms find feasible paths of similar quality with orders of magnitude improvement in computational speed compared with single-query methods. The proposed algorithms are implemented on hardware to enable real-time autonomous navigation in an indoor environment with frequent interactions with pedestrians.**

I. Introduction

Planning feasible, collision-free paths in a previously mapped environment is fundamental to many robotic applications, such as indoor service robots [1], campus-wide shuttle systems [2], and robotic manipulators [3]. Although a map is often provided, there can be frequent changes in the environment due to the presence of other mobile agents. For example, in office buildings, furniture is often shifted due to people's daily activities. To navigate efficiently and safely in such environments, a robot needs to leverage the knowledge of the map to find good paths quickly, and to replan locally when changes are detected.

A common strategy is to use single-query motion planning algorithms for finding feasible paths (e.g. A*), and then adapt to local changes in the dynamic environment (e.g. D*) [2], [4]–[6]. However, planning from scratch for each new query can be inefficient for applications that require repeated, on-demand motion planning. Multi-query methods seek to improve computational speed by doing pre-computation on the map or learning from experience. For instance, sampling-based roadmap methods [7], [8] pre-compute a topological graph for a more concise representation of the map. In addition, using previously planned paths, experience-based methods can pre-compute a better heuristic [9], [10] or construct a path library [3] to solve for new queries more efficiently. However, these methods tend to bias toward previously sampled points or explored paths, and do not explicitly consider uncertainties in the environment.

Laboratory of Information and Decision Systems, Massachusetts Institute of Technology, 77 Massachusetts Ave., Cambridge, MA, USA {chenyuf2, syliu, miaoliu, justinm, jhow}@mit.edu

Fig. 1: A robotic vehicle navigates autonomously in a dynamic indoor environment. The vehicle uses a SICK Lidar for localization, and a Velodyne for obstacle detection.

An alternative to experience-based methods is to learn from the map a potential function that estimates pairwise cost-to-go. A good cost-to-go estimate can serve as a heuristic for speeding up single-query search algorithms. While efficient to query, classical potential field methods can suffer from having many local minima [7], thus limiting their usefulness for motion planing in complex environments. Researchers have proposed potential functions that do not have local minima [11], such as by solving a steady-state heat equation on the map [12]. However, these potential functions need to be solved individually for each query to a different goal. Computing a potential function is often much more expensive than finding a feasible path, because the former solves for every state with respect to a goal. Yet, a good cost-to-go function can be valuable for purposes beyond just finding a feasible path. For instance, a cost-to-go function can help guide a safe local exploration strategy [13] in an unknown environment. More generally, motion planning in the vicinity (i.e. perception range) of a robot often requires more sophisticated models, such as accounting for interaction with pedestrians [14] and modulating the robot's speed [2], [15]. Having a potential function could be useful for guiding such local search methods.

This work seeks to pre-compute and store a potential function by learning the map's geometry. A naive approach would be to solve the all-pairs-shortest-path problem (e.g. with the Floyd-Warshall algorithm [16]). However, doing so would require $O(|V|^2)$ storage, which could be formidable for complex environments. For instance, a map with a million

states would require more than 1TB of storage space. This work finds a novel application of diffusion maps [17] to address the issue. Rather than directly encoding pairwise potentials, the diffusion map method transforms each state into a new coordinate, and retrieves a meaningful pairwise similarity measure by calculating the Euclidean distances in the new coordinate space. Furthermore, a diffusion map can be pre-computed in a reasonable amount of time, and stored efficiently in memory (on the same order of magnitude as the map).

The main contributions of this work are 1) development of an efficient motion planning algorithm based on a novel application of diffusion maps, 2) integration of the diffusion potential with a local planner to handle dynamic changes in the environment, 3) empirical results that show significant improvement in computational speed for finding good-quality feasible paths, and 4) hardware demonstrations of a robot navigating autonomously in a dynamic environment.

II. Preliminaries

This section reviews the construction of diffusion maps and summarizes key properties of the mapping in relation to motion planning. A diffusion map is a graph-based nonlinear dimensionality reduction technique [18] that has many applications in data processing, such as clustering [17] and image inpainting [19]. For a high dimensional dataset, a diffusion map seeks to find a lower dimensional manifold that captures major structures in the dataset. For instance, in [18], the diffusion map approach finds a two dimensional parametrization that captures rotational motion on a high dimensional (in pixel count) image dataset.

For clarity, the following notations are used in the exposition below. For a given graph $\mathcal{G}$, each node n_i is associated with a coordinate $\mathbf{x}_i$ in the original space $\mathcal{X}$. Diffusion map finds an alternative parametrization $\mathbf{x}_i \mapsto \mathbf{y}_i$ in a diffusion space $\mathcal{Y}$. Let $d_{\mathcal{X}}$, $d_{\mathcal{Y}}$, $d_{\mathcal{G}}$ denote Euclidean distance in the original space, Euclidean distance in the diffusion space, and geodesic distance on the graph, respectively.

The key idea of diffusion maps is that local distances ($d_{\mathcal{X}}$) captured by edge weights in the graph are reliable, but large distances in the original coordinates ($d_{\mathcal{X}}$) can be misleading as a similarity metric. For instance, little can be inferred about the relationship between two images – either the same object viewed at different angles or two distinct objects – if every pair of pixels differs substantially. This work is motivated by the insight that, for motion planning, local distances (e.g. $d_{\mathcal{X}}$ between adjacent nodes) are reliable and easy to compute, but large distances ($d_{\mathcal{X}}$) between non-neighboring nodes are often uninformative. For example, a pair of non-neighboring nodes with a small $d_{\mathcal{X}}$ could be separated by a wall, thus having a large geodesic distance $d_{\mathcal{G}}$. The diffusion map algorithm addresses this problem by solving for a diffusion process to learn the geometry of the graph in the original space $\mathcal{X}$ [20]. In particular, the following exposition will show that diffusion distance $d_{\mathcal{Y}}$ is a meaningful similarity metric relating to geodesic distance $d_{\mathcal{G}}$.

Algorithm 1: Diffusion maps

1 **Input:** distance matrix $\mathbf{W}$, time step t, truncation k
2 **Output:** diffusion matrix $\mathbf{Y}$
3 similarity matrix $\mathbf{A} \leftarrow \ker(\mathbf{W})$
4 diagonal degree matrix $\mathbf{D} \leftarrow \text{rowSum}(\mathbf{A})$
5 $\mathbf{A}_1 \leftarrow \frac{1}{2}\mathbf{A} + \frac{1}{2}\mathbf{D}$ // lazy Markov chain
6 $\mathbf{A}_2 \leftarrow \mathbf{D}^{-1}\mathbf{A}_1\mathbf{D}^{-1}$ // anisotropic scaling
7 $\mathbf{D}_2 \leftarrow \text{rowSum}(\mathbf{A}_2)$, $\mathbf{M}_2 \leftarrow \mathbf{D}_2^{-1}\mathbf{A}_2$
8 $(\mathbf{V}, \mathbf{\Lambda}) \leftarrow \text{eigs}(\mathbf{D}_2^{-\frac{1}{2}}\mathbf{A}_2\mathbf{D}_2^{-\frac{1}{2}}, k+1)$
9 $\mathbf{\Phi} \leftarrow \mathbf{D}_2^{-\frac{1}{2}}\mathbf{V}$
10 $\mathbf{Y} \leftarrow [\lambda_2^t\phi_2, \lambda_3^t\phi_3, \ldots, \lambda_{k+1}^t\phi_{k+1}]$
11 **return** $\mathbf{Y}$

The construction of a diffusion map for motion planning is outlined in Algorithm 1. Given the map of an environment represented as a graph $\mathcal{G}$, the first step is to construct a sparse distance matrix $\mathbf{W}$ (line 1), where W_{ij} is the Euclidean distance between node i and j if they are adjacent, and infinity otherwise. Then, a similarity matrix $\mathbf{A}$ is computed by applying a kernel (line 3) that maps each entry in the distance matrix to a similarity score. This work uses the Gaussian kernel $g(x) = exp(-\frac{x^2}{2w})$, where w is a length scale parameter [1]. The similarity matrix can be interpreted as the transition probabilities of a Markov chain. In particular, A_{ij}/D_{ii} is the transition probability from node i to j, where $D_{ii} = \sum_j A_{ij}$ is the degree of node i (line 4). To avoid potential issues with aperiodicity, the Markov chain is made *lazy* in line 5. The transition probabilities are scaled using the degree, D_{ii}, of each node for reasons to be explained later this section. The last step computes a transition matrix $\mathbf{M}_2$ (line 7), its $k+1$ largest eigenvalues $\{\lambda_1, \ldots, \lambda_{k+1}\}$, and the associated right eigenvectors $\{\phi_1, \ldots, \phi_{k+1}\}$, in lines 8-9. Diffusion map associates the i-th node in the original graph with the i-th entry of each eigenvector; more precisely, it defines the following mapping (line 10) [2]

$$\mathbf{x_i} \mapsto \mathbf{y_i} = [\lambda_2^t\phi_2(i), \lambda_3^t\phi_3(i), \ldots, \lambda_{k+1}^t\phi_{k+1}(i)]. \quad (1)$$

The new space $\mathcal{Y}$ is called the k-truncated diffusion space, and the distance $||\mathbf{y}_i - \mathbf{y}_j||_2$ is known as the k-truncated diffusion distance [17]. The choice of parameter t in (1) will be explored in greater details in Section III-A.

We make a few remarks regarding the process of computing this mapping. First, coordinate $\mathbf{x}_i$ in the original space is not needed in computing the diffusion map. The procedure finds structures (e.g. eigenvectors) in the graph only using the distance matrix $\mathbf{W}$, which captures local distances and connectivities. Second, for motion planning problems, $\mathbf{W}$ is typically sparse since each node in the graph has only a few neighbors. The sparse structure is maintained throughout Algorithm 1, thus making it memory efficient.

[1] For example, if the graph is constructed from a discretized gridmap, w can be set to the width of a grid.

[2] ϕ_1 is a constant vector, so it cannot differentiate between different nodes [17].

Third, if $\mathbf{W} \in \mathbb{R}^{n \times n}$ is symmetric, the full set of eigenvectors $\{v_i\}$ (line 8 Algorithm 1) would form an orthonormal basis of $\mathbb{R}^n$ [17], and the eigenvalues λ_i's are shown to be in $[0, 1]$. Fourth, a graph with r components would have precisely r eigenvalues equal to one, and the corresponding eigenvectors would reveal the r connected components. For simplicity, the graph is assumed to be connected.

The following facts illustrate that diffusion distance is a meaningful similarity metric.

Fact 1 [17]: Consider a random walk on a graph $\mathcal{G}$ with n nodes. Let $s_t \in \{1, \ldots, n\}$ denote the state of the walk after t iterations, and $deg(h)$ denote the degree of node h in $\mathcal{G}$. When keeping the full set of eigenvectors ($k = n - 1$ in Algorithm 1), the diffusion distance is equal to a weighted distance between two probability clouds after t steps. More precisely, let $P^t_{h_i}$ denote $\mathbb{P}(s_t = h | s_0 = i)$, it is shown

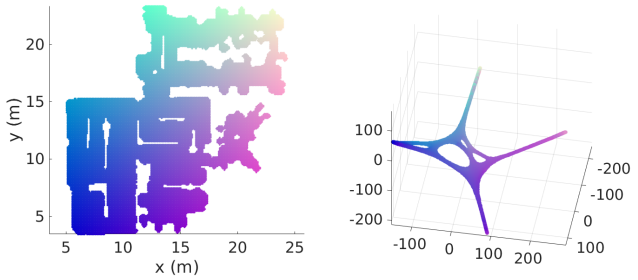
$$||\mathbf{y}_i - \mathbf{y}_j||_2^2 = \sum_h \frac{1}{deg(h)} \left[P^t_{h_i} - P^t_{h_j} \right]^2. \qquad (2)$$

Fact 2 [17]: The k-truncated diffusion map is the best k-dimensional approximation of the full diffusion map, in the sense of the average approximation error of diffusion distance (2).

Intuitively, nodes with smaller geodesic distance on the graph are expected to have more overlap in densities after a t-step diffusion process. Facts 1 and 2 indicate that diffusion distance is an appropriate measure of the similarity between the probability densities.

In light of the algebraic form of λ_i^t in (1) and the fact that $\lambda_i \in [0, 1]$, the diffusion distance (2) can be approximated by keeping the first k eigenvectors, where $k \ll n$. Further, there often exists a spectral gap for real world problems [21], so a small value of k is typically sufficient to achieve a good approximation of (2). It is found empirically that $k \approx 10$ is sufficient for the complex planning domains considered in Section IV. An interesting perspective is that finding the full diffusion coordinate (i.e. $k = n - 1$) is as expensive as solving the all-pairs-shortest-path problem, because finding all n eigenvectors requires $O(|V|^3)$ computation and $O(|V|^2)$ storage. Yet, a diffusion map's explicit eigenvalue formulation permits an efficient approximation that leads to substantial savings on both computational time and memory usage, because finding the first k eigenvectors requires $O(k|V|^2)$ computation and $O(k|V|)$ storage.

Finally, it can be shown that the eigenvectors are related to geometric structures in the original domain. Without the anisotropic scaling step (line 6 Algorithm 1), $\{v_i\}'s$ would also be the eigenvectors of the graph Laplacian matrix [18], which can be seen as a discrete approximation of Laplace-Beltrami operator plus a density term [18], [22]. The density term is undesirable because the diffusion process is biased toward regions of high connectivity (i.e. nodes with a large degree). The anisotropic scaling step removes the effect of the density term [18]. Hence, finding diffusion maps is related to solving the heat equation. For instance, the $\{v_i\}'s$ of a line graph resemble sinusoidal functions, which are the eigenfunctions of 1D heat equation. Thus, the first few eigenvectors capture the low frequency components, which can be interpreted as long range structures in the original domain [20]. Researchers have investigated using the eigenvectors $\{v_i\}'s$ as basis functions for solving reinforcement learning problems [23], but have not used pairwise relationship as revealed in diffusion maps for motion planning.



(a) 2D gridmap (b) diffusion space

Fig. 2: Diffusion map as coordinate transformation. Each node is associated with a unique color showing the correspondence between the 2D space and the diffusion space. Euclidean distance between every pair of states in the diffusion space is called the diffusion distance, and is shown to be a meaningful similarity measure.

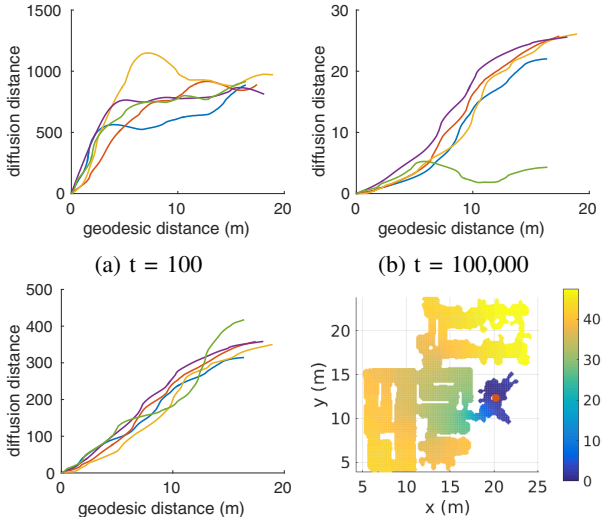
In short, diffusion coordinates reveal geometric structures of the original domain, and diffusion distance $d_{\mathcal{Y}}$ is a meaningful similarity metric that relates to geodesic distance $d_{\mathcal{G}}$. Furthermore, diffusion coordinates can be computed and stored efficiently. It will be shown in Section III that diffusion distance can be used to improve computational speed in motion planning applications.

III. Approach

The following explores the effects of varying the time scale parameter t in (1) on diffusion distance for motion planning. Further, diffusion-based motion planning algorithms are developed for both static and dynamic environments. A gridmap of an office space is created using the ROS *gmapping* package, as shown in Fig. 2a. The width of each grid cell is set to be 0.1m, and grid cells within 0.25m are considered neighbors if a line connecting their centers does not intersect any obstacle. When converted to a graph representation, this domain contains 20,666 nodes, and an average of 18.7 connections per node. This work uses a gridmap representation to help elucidate coordinate transformation using diffusion maps; yet, the proposed approach is applicable to any graph structure (see input to Algorithm 1), such as sparse graphs constructed by roadmap-based methods. This domain is used as a running example to illustrate various aspects of the proposed algorithms.

The first three dimensions of the diffusion coordinates[3] (2) are shown in Fig. 2b. The color shows the correspondence between the original 2D space and the diffusion space. For example, the lime color shows that the top right region in the 2D space is being mapped to the top middle region in the diffusion space. This diffusion map captures long range

[3]For numerical reasons, the diffusion coordinates are scaled by a factor of n (number of vertices in the graph) in this work.



(a) t = 100 (b) t = 100,000

(c) t = 10,000 (d) diffusion dist. at t = 10,000

Fig. 3: Diffusion distance as a function of time scale parameter t. Subfigures a-c show geodesic distance versus diffusion distance to goal on the optimal paths for a set of randomly generated test cases. Subfigures a,b,c show when the time scale parameter t is set too small, too big, or well tuned, respectively. Subfigure d shows the diffusion distance from every node to the red node at (20.1, 12.5).

structures – the corner rooms are being mapped to tips of a roughly tetrahedron-shaped object in the diffusion space.

A. Diffusion Distance

Recall that the time scale parameter t is the number of steps of a random walk as shown in (2). Given the form of scaling term λ_i^t in (1), when t is set too small, approximation of diffusion distance would be poor because the truncated coordinates have non-negligible weight; when t is set too large, a lot of information would be lost because the diffusion space would collapse to the first dimension. To empirically determine a good value of t, we find the optimal paths for several randomly generated test cases (e.g. use A*), and plot geodesic distance versus diffusion distance to goal (each path shown in a different color in Fig. 3a-c). Ideally, there would be a monotonic linear relationship, because diffusion distance decreases as the vehicle moves closer to the goal. Figures 3a and 3b show when t is set too small or too big, respectively. It is found empirically that setting $t \approx 50 w_{\mathcal{G}}$ (Fig. 3c), where $w_{\mathcal{G}}$ is the width of a graph, leads to good performance. Figure 3d visualizes the diffusion distance with respect to the node at (20.1, 12.5). The tuning procedure does not lead to a substantial increase in computational time, because the full diffusion map does not need to be computed repeatedly. In particular, only the scaling step (line 10 of Algorithm 1) needs to be run for several times, which is inexpensive compared with computing the eigenvectors (line 8).

B. Motion Planning in Static Environments

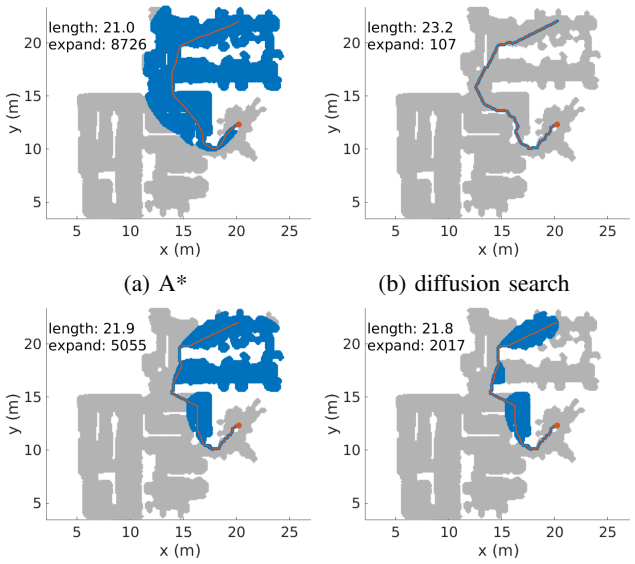
Diffusion search is developed in Algorithm 2, which finds paths by steepest descent on diffusion distance surface (e.g. Fig. 3d). Using terminology common to graph-based search algorithms, a priority queue $OPEN$ is used for choosing the next state to be expanded, and a list $CLOSED$ is used for keeping track of the expanded states (line 3). The $OPEN$ queue is sorted by diffusion distance to goal (line 4). The main loop is shown in lines 5-10, which proceeds by repeatedly expanding the node with minimum diffusion distance to goal (line 6) and adding its unexplored neighbors to the queue (lines 7-9). As explained in Section III-A, a problem with keeping only the first k coordinates in computing diffusion distance is that short scale structures would be lost. It is found that the diffusion distance surface (e.g. Fig. 3d) is typically very flat in the vicinity of the goal, leading the diffusion search algorithm to visit many more states than necessary near the goal state. This behavior can be observed in Fig. 3c, in which the lines becomes more flat when diffusion distance is below 20. In practice, this problem can be handled by noting that the vehicle is typically very close to the goal (e.g. less than 2m) when diffusion distance falls below a certain threshold. At this point, a simple analytical solution, such as computing a Dubins curve, is usually sufficient to reach the goal. For a fair comparison with other graph-based search algorithms, the diffusion search algorithm switches to A* after the diffusion distance falls below a threshold η, as shown in lines 11-13.

Algorithm 2: Diffusion search

1 **Input:** diffusion matrix $\mathbf{Y}$, diffusion threshold η
2 **Output:** path $\mathbf{p}$ from s_{start} to s_{goal}
3 $OPEN \leftarrow \emptyset, \quad CLOSED \leftarrow \emptyset$
4 insert s_{start} into $OPEN$ with value $||\mathbf{y}_{start} - \mathbf{y}_{goal}||_2$
5 **while** s_{goal} *not expanded* **do**
6 $s \leftarrow \text{pop}(OPEN)$
7 **for** *each neighbor* s' *of* s **do**
8 **if** s' *not in* $CLOSED$ **then**
9 insert s' into $OPEN$ with value $||\mathbf{y}_{s'} - \mathbf{y}_{goal}||_2$
10 insert s into $CLOSED$
11 **if** $||\mathbf{y}_s - \mathbf{y}_{goal}||_2 < \eta$ **then**
12 $\mathbf{p}' \leftarrow \text{retrievePath}(s_{start}, s)$
13 **return** $\mathbf{p} \leftarrow \mathbf{p}' \cup \text{findPath}(s, s_{goal})$

An example of diffusion search is shown in Fig. 4b. Compared with the A* in Fig. 4a, diffusion search finds a path that is 10.2% longer than optimal but expands many fewer states ($\leq 2\%$ the number in this case). Since computational time is linear in the number of expanded states, diffusion search is more than an order of magnitude faster than A*. More detailed performance comparison is presented in Section IV. A major source of sub-optimality is choosing the next state greedily (line 9 Algorithm 2), which is analogous to the steepest descent algorithm. It is possible to improve the solution quality by using post-processing methods [24], [25]. A method to address this issue is developed in Section III-C.

Further, diffusion distance can be used to prune states that are unlikely to be on the optimal path. Thereby, diffusion



(a) A* (b) diffusion search

(c) Weighted A* (d) Weighted A* with diffusion

Fig. 4: Path planning in a static environment. Four algorithms are applied to find a path from (20, 22) to (20, 12). Red curve shows the planned path with the red dot being the goal state. Blue shows the set of expanded states. Diffusion maps can be either used by itself for finding a feasible path (subfigure b), or be integrated with an existing algorithm (subfigure d).

search can be easily integrated into an existing graph-based motion planner to improve computational speed. In particular, Weighted A* [5] is considered. Using an inadmissible heuristic $h(\mathbf{x}_1, \mathbf{x}_2) = c \cdot ||\mathbf{x}_1 - \mathbf{x}_2||_2$ (for $c > 1$), Weighted A* achieves better computational speed at the expense of losing optimality[4]. An example of using Weighted A* for path planning is shown in Fig. 4c. The problem with Weighted A* is that similar to A*, a substantial amount of computation can be spent on exploring dead-ends. This problem is addressed by using diffusion distance to prune states whose diffusion distance to goal are higher than that of their parents. In particular, nodes being pushed onto the $OPEN$ queue are penalized (i.e. adding a large constant to their cost) if their diffusion distance is larger than that of their parent. An example of Weighted A* with diffusion metric is shown in Fig. 4d. Note that Weighted A* with diffusion metric attains a slightly better path while expands half as many nodes.

C. Motion Planning in Dynamic Environments

In many real world applications, there can be frequent small changes in the environment. For instance, for indoor navigation, obstacles' position (e.g. chairs) might be shifted in relation to the map and there might be other dynamic agents in the environment. Assuming no major changes to the underlying geometry (e.g. blocking an entire corridor), the pre-computed diffusion map can be used to guide a local motion planner. This enables the planning system to focus the computational resources within the robot's perception range, without needing to compute a complete feasible path to goal.

[4]Weighted A* reduces to A* with $c = 1$. This work uses $c = 3$.

Algorithm 3: Diffusion search with motion primitives

1 motion primitives $\mathbf{MP} \leftarrow$ computeMP()
2 **while** $\mathbf{x}_{cur}$ *not sufficiently close to goal* **do**
3 $(\mathbf{x}_{cur}, Obs) \leftarrow$ sensorUpdate()
4 $\mathbf{MP}' \leftarrow$ pruneWithObs($\mathbf{x}_{cur} + \mathbf{MP}$, Obs)
5 // find motion primitive segments that reduce diffusion distance to goal
6 $\mathbf{MP}'' \leftarrow$ pruneWithDiffusionDist($\mathbf{MP}', \mathbf{Y}$)
7 $\mathbf{p}'' \leftarrow$ chooseBestLocalPath($\mathbf{MP}''$)
8 move along $\mathbf{p}''$

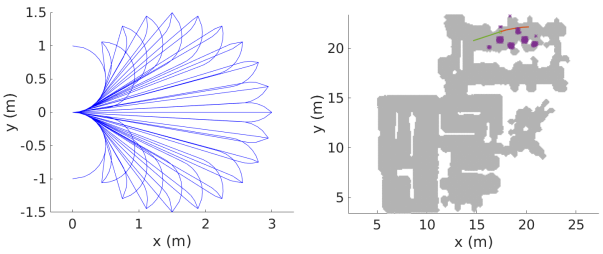
Algorithm 3 proposes a motion planner that combines diffusion search with motion primitives. A set of motion primitives, shown in Fig. 5a, is pre-computed based on the Dubins car model [13] (line 1). The main loop is repeated until the vehicle reaches the goal (line 2). At every time step, the vehicle updates its pose and obstacle information within its perception range (line 3). Then, the set of motion primitives is translated and rotated with respect to the vehicle's pose to form a set of local paths. Each local path is pruned up to the first point that it collides with any obstacle (line 4). Each path is further pruned up to the point with the smallest diffusion distance to goal (line 5-6). At this stage, what remains is a set of traversable paths that lead to positions closer to the goal than the vehicle's current position. The last step is to choose the best (i.e., leading to lowest diffusion distance to goal) local path within this set (line 7), and the vehicle is then presumed to move along the chosen path until the next sensor update (line 8). An example is shown in Fig. 5b-d. New obstacles shown in purple are generated at random to simulate dynamic changes in the environment. The vehicle is assumed to have a perception radius of 3m, within which it can observe the newly added obstacles. Following diffusion distance pre-computed from the static map, and planning locally using the pre-computed motion primitives, the vehicle finds a path that is within 2% of the optimal.

IV. Results

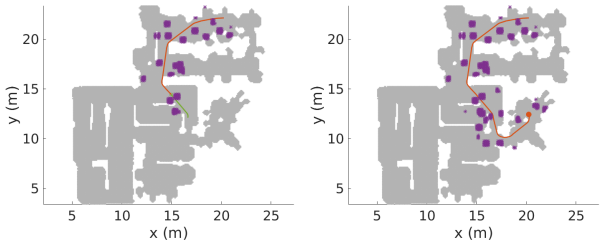
The following presents simulation and hardware results on path planning with diffusion maps in two additional domains. The performance of the diffusion-based algorithms is assessed on randomly generated test cases.

A. Path Planning for a Nonholonomic Vehicle

Consider motion planning for a nonholonomic vehicle in the map shown in Fig. 2a. Following [24], a heading angle dimension discretized at 5° increments is included in addition to the 2D position to account for vehicle dynamics. When converted to a graph representation, this domain contains $\approx$1.4 million nodes. The connectivity (number of neighbors per node) of this domain is lower than that of the 2D map because the vehicle is constrained by a 0.5m minimum turning radius. An example is shown in Fig. 6 in which diffusion search finds a feasible path within 15% of the optimal path



(a) motion primitives (b) step 20



(c) step 120 (d) step 212

Fig. 5: Path planning in a dynamic environment. Subfigure a shows a set of motion primitives based on the Dubins car model. Subfigures b-d show path planning using Algorithm 3. Grey shows the map, purple shows the newly added obstacles that are visible within vehicle's perception range. Red shows the path traveled, and green shows the chosen local path as described in line 7 Algorithm 3.

TABLE I: Computing diffusion maps with $k = 10$.

	2D	3D-nh	mlt-flr
number of states	20,666	1,485,912	311,890
avg. number neighbors per node	18.7	11.9	19.1
computational time (sec)	1.7	181	381

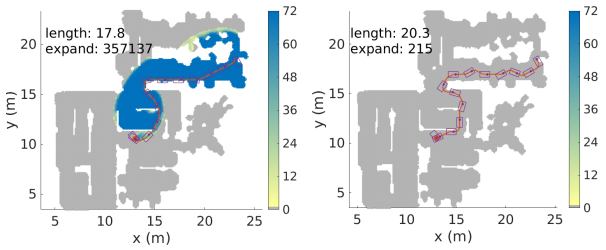
but explored three orders of magnitude fewer nodes than A* (using the Euclidean distance heuristic).

B. Path Planning on Multiple Floor Levels

Next consider motion planning on three floors connected by multiple elevators, as shown in Fig. 8. Each floor is mapped using the ROS *gmapping* package and discretized to build a gridmap with the same resolution as Fig. 2a. When converted to a graph representation, the three floors combined have more than 300,000 states. Another challenge with this domain is that there are often multiple routes connecting nodes on different floors. An example is shown in Fig. 7, in which diffusion search finds a feasible path within 12% of the optimal path but explored two orders of magnitude fewer nodes than A* (with the Euclidean distance heuristic). Connections between floor levels and diffusion distance with respect to the goal are shown in Fig. 8.

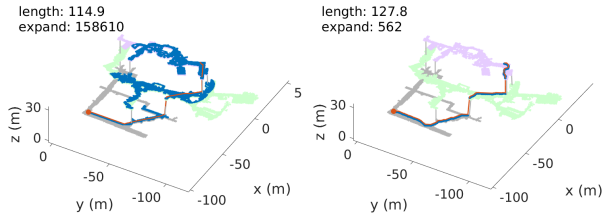
C. Performance Comparison

The domains illustrated in Fig. 2a, Fig. 6, and Fig. 7 are denoted as 2D gridmap (2D), 3D gridmap with nonholonomic constraints (3D-nh), and multiple connected floor levels (mlt-flr), respectively. The complexity of each domain in graph representation and the corresponding time required for computing diffusion maps is summarized in Table I. The



(a) A* (b) diffusion search

Fig. 6: Path planning for a nonholonomic vehicle. The shaded colors (yellow to blue) show the number of expanded states at each 2D position. Recall this is a 3D domain, so the heading angle dimension is collapsed for plotting the number of expanded states.



(a) A* (b) diffusion search

Fig. 7: Path planning on multiple floors. Grey, pink and lime green show three floor levels, and gray vertical lines are connecting elevators. Blue denotes expanded states; red shows planned path.

algorithms are run on a computer with an Intel i7-2600K CPU and 16GB of memory.

Diffusion-based motion planning algorithms are being evaluated on 100 randomly generated test cases on each domain. In particular, Fig. 9 shows a comparison of solution quality as measured by path length and run time as measured by the number of expanded states. Run time is also computed and shown in Table II. To account for variability among different test cases, we normalized the performance statistics by that of A*. The average performance statistics is summarized in Table II.

Diffusion search finds paths that are about 20% longer than the optimal paths. A major cause is that greedy search leads to some local oscillations along the path, as shown in Figs. 6b and 7b. In practice, the use of a larger look-ahead distance, as in Algorithm 3, mitigates this issue to a large extent. Diffusion search often finds a feasible path two orders of magnitude faster than A* and one order of magnitude faster than Weighted A*. Moreover, diffusion distance can be integrated into A*-based algorithms for pruning states that are unlikely to be on the optimal path. In particular, a factor of two speedup (and slight improvement in solution quality) was achieved by augmenting Weighted A* with diffusion distance, as shown in the bottom two rows of Table II. In addition, while Table II compares the run time to find a feasible path, the availability of a pre-computed diffusion map allows the vehicle to move towards its goal without needing to first find a feasible path, such as described in Algorithm 3.

D. Hardware Experiment

An autonomous vehicle is developed using the *Pioneer 3-AT* rover platform Fig. 1. The vehicle is equipped with a SICK

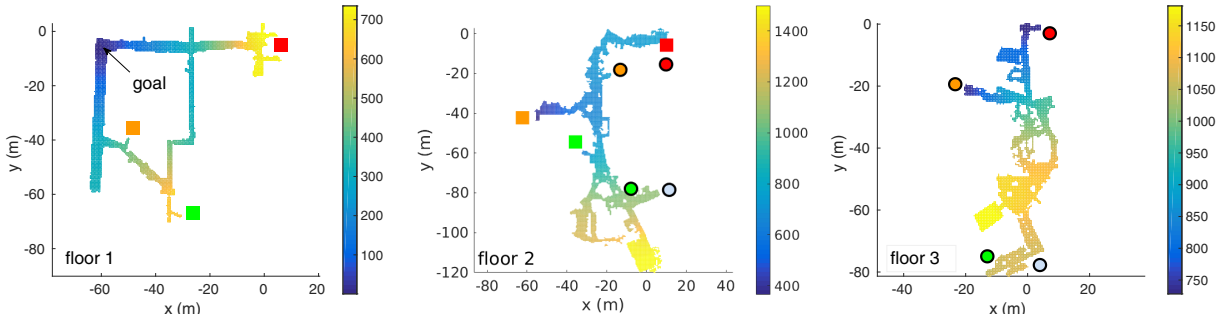
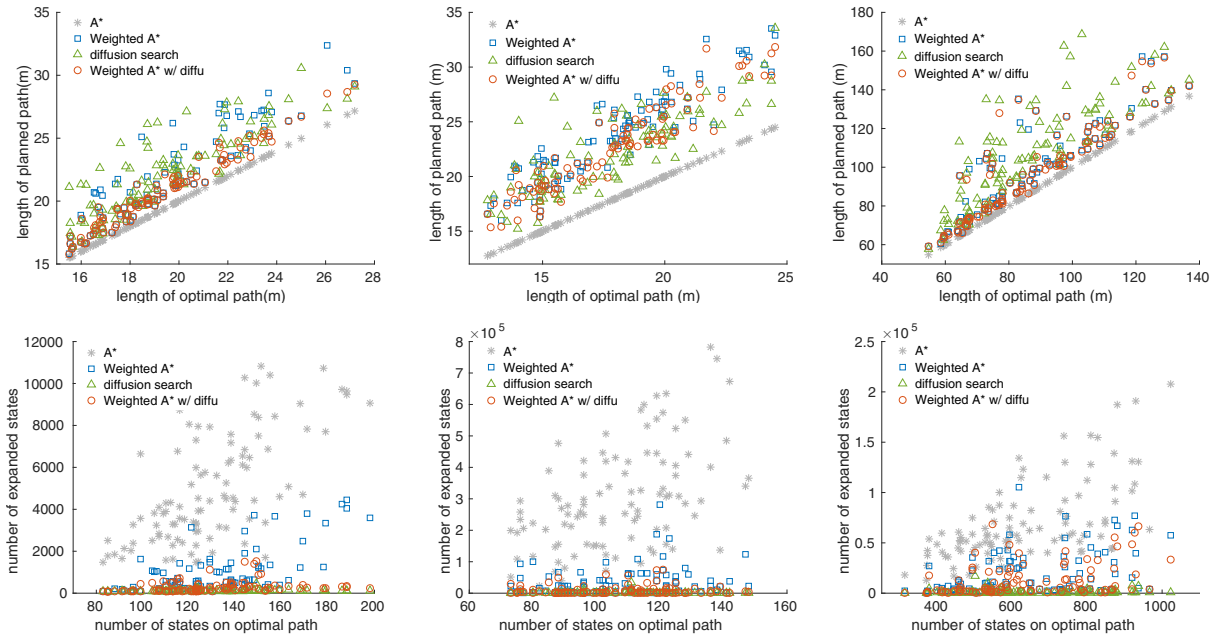


Fig. 8: Multi-floor domain and diffusion distance with respect to goal position (-60.5, -7) on floor 1. Connections between floor 1 and 2 are shown in squares, and connections between floor 2 and 3 are shown in circular disks. Diffusion distance is computed similarly as in Fig. 3d. A path planning example using this diffusion map is shown in Fig. 7.



(a) 2D (b) 3D-nh (c) mlt-flr

Fig. 9: Performance statistics on a set of randomly generated test cases. A* always finds the optimal solution but needs to explore a large number of states. Diffusion search finds feasible paths that are about 20% longer than the optimal paths, but explores two orders of magnitude fewer states than A*. Further, augmenting Weighted A* with diffusion distance metric leads to a 3% improvement in solution quality and two fold reduction in the number of expanded states.

Lidar for mapping and localization, and a Velodyne sensor for obstacle detection. The vehicle navigated fully autonomously in a dynamic environment (Fig. 8) using Algorithm 3. A hardware demonstration video is available at https://www.youtube.com/watch?v=3s9gswbhWSU.

Figure 10a illustrates the vehicle navigating through a busy indoor environment that results in frequent interactions with pedestrians – an average of 11.3 people came within 2m of the vehicle per minute. The ROS *costmap_2d* package is used to process Velodyne point clouds for detecting obstacles within a 5m $\times$ 5m region around the vehicle. The size of the region is chosen to achieve a 10Hz update rate for enabling fast local replanning in Algorithm 3. Figure 10b shows the vehicle detecting and navigating around a person. Despite the person standing in the direct path to the goal, the vehicle quickly finds a safe local path (green) from its motion primitives library (line 4-7 Algorithm 3), and navigates behind the person. In short, our experiment shows that pre-computing a diffusion map is an effective method for reducing computational time. Recall in Algorithm 3, when assigned a new goal, a vehicle does not need to wait for a feasible path to be found.

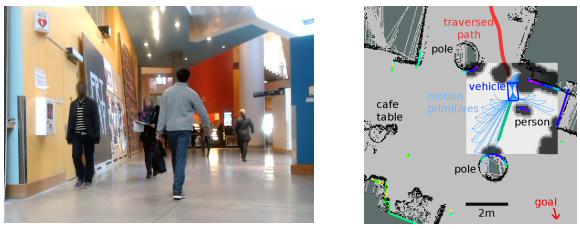
V. Conclusion

This paper developed a multi-query motion planning algorithm for dynamic environments. In particular, given a previously generated map, the proposed algorithm would learn the map's geometry and find a parametrization that encodes pairwise similarity. This similarity metric, called diffusion distance, was used as a potential function to guide local searches. The proposed algorithms were applied to three real-world domains, and achieved orders of magnitude improvement in computational speed compared with single-query methods. Moreover, a hardware experiment demon-

[5]In this work, grids within 0.25m are considered neighbors if the line connecting their centers does not intersect any obstacle. Thus, a feasible path can have fewer states on its path than that of the optimal path.

TABLE II: Average performance on the random test cases shown in Fig. 9. Path quality is measured by the ratio of (path length)/(optimal path length). Number of visited states is normalized by the number of states on the optimal path. Run time is normalized by that of A*. Second row shows that diffusion search finds feasible paths of comparable quality to other approaches while expanding many fewer nodes. Third and fourth rows show augmenting diffusion distance to a graph-based method leads to more than two fold reduction in the number of expanded states and run time.

	Path Length Ratio			Expanded States Ratio			Run Time Ratio		
	2D	3D-nh	mlt-flr	2D	3D-nh	mlt-flr	2D	3D-nh	mlt-flr
A*	1	1	1	36.8	2844.5	108.6	1	1	1
diffusion search	1.14	1.26	1.25	0.87[5]	13.2	2.09	0.038	0.011	0.019
Weighted A*	1.11	1.33	1.14	6.27	286.4	27.5	0.172	0.102	0.231
Weighted A* w/ diffusion	1.07	1.27	1.12	2.23	89.4	18.83	0.070	0.035	0.162



(a) on-board camera view (b) navigating around obs.

Fig. 10: Autonomous navigation in a dynamic environment using Algorithm 3. Subfigure a shows the vehicle comes in frequent interaction with pedestrians. Subfigure b illustrates the vehicle detecting obstacles (black regions) within its perception range (white patch). A set of motion primitives (blue) is pruned by the detected obstacles. The feasible motion primitive (green) that leads to the smallest diffusion distance to goal is chosen.

strated autonomous navigation in an indoor environment that requires frequent interaction with pedestrians. Future work will consider integrating more sophisticated local motion planners, such as accounting for interaction with pedestrians [14], with diffusion distance.

ACKNOWLEDGMENT

This work is supported by Ford Motor Company.

REFERENCES

[1] T. Kanda, M. Shiomi, Z. Miyashita, H. Ishiguro, and N. Hagita, "A communication robot in a shopping mall," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 897–913, 2010.

[2] H. Bai, S. Cai, N. Ye, D. Hsu, and W. S. Lee, "Intention-aware online POMDP planning for autonomous driving in a crowd," in *Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 454–460.

[3] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 3671–3678.

[4] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354–363, 2005.

[5] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems*, 2003.

[6] M. Otte and E. Frazzoli, "RRT^X: Real-time motion planning/replanning for environments with unpredictable obstacles," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 461–478.

[7] S. M. LaValle, *Planning algorithms*. Cambridge University Press, 2006.

[8] J. van den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, May 2006, pp. 2366–2371.

[9] M. Phillips, B. J. Cohen, S. Chitta, and M. Likhachev, "E-Graphs: Bootstrapping planning with experience graphs." in *Robotics: Science and Systems*, 2012.

[10] M. Phillips and M. Likhachev, "Speeding up heuristic computation in planning with experience graphs," in *Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 893–899.

[11] A. Valero-Gomez, J. V. Gomez, S. Garrido, and L. Moreno, "Fast marching methods in path planning," *IEEE Robotics and Automation Magazine*, 2013.

[12] Y. Wang and G. S. Chirikjian, "A new potential field method for robot path planning," in *Proceedings of the 2000 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, 2000, pp. 977–982.

[13] C. Richter, J. Ware, and N. Roy, "High-speed autonomous navigation of unknown environments using learned probabilities of collision," in *Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 6114–6121.

[14] P. Trautman, J. Ma, R. M. Murray, and A. Krause, "Robot navigation in dense human crowds: the case for cooperation," in *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 2153–2160.

[15] T. Gu, J. Atwood, C. Dong, J. M. Dolan, and J.-W. Lee, "Tunable and stable real-time trajectory planning for urban autonomous driving," in *Proceedings of the 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 250–256.

[16] T. H. Cormen, *Introduction to algorithms*. MIT press, 2009.

[17] S. Lafon and A. B. Lee, "Diffusion maps and coarse-graining: A unified framework for dimensionality reduction, graph partitioning, and data set parameterization," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, no. 9, pp. 1393–1403, 2006.

[18] R. R. Coifman and S. Lafon, "Diffusion maps," *Applied and computational harmonic analysis*, vol. 21, no. 1, pp. 5–30, 2006.

[19] S. Gepshtein and Y. Keller, "Image completion by diffusion maps and spectral relaxation," *IEEE Transactions on Image Processing*, vol. 22, no. 8, pp. 2983–2994, 2013.

[20] R. R. Coifman, S. Lafon, A. B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. W. Zucker, "Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 21, pp. 7426–7431, 2005.

[21] B. Nadler, S. Lafon, R. R. Coifman, and I. G. Kevrekidis, "Diffusion maps, spectral clustering and eigenfunctions of Fokker-Planck operators," *arXiv preprint math/0506090*, 2005.

[22] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[23] S. Mahadevan and M. Maggioni, "Proto-value functions: A Laplacian framework for learning representation and control in Markov Decision Processes." *Journal of Machine Learning Research*, vol. 8, no. 2169-2231, p. 16, 2007.

[24] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," in *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08)*. Chicago, USA: AAAI, June 2008.

[25] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA)*, 2009, pp. 489–494.