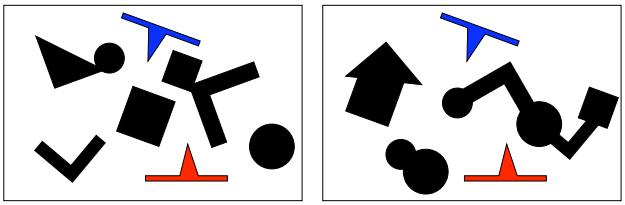
Planning Motion in Environments with Similar Obstacles

Jyh-Ming Lien and Yanyan Lu
George Mason University, Fairfax, Virginia 22030
{jmlien, ylu4}@gmu.edu

***Abstract*— In this work, we investigate solutions to the following question: Given two motion planning problems $\mathcal{W}_1$ and $\mathcal{W}_2$ with the same robot and similar obstacles, can we reuse the computation from $\mathcal{W}_1$ to solve $\mathcal{W}_2$ more efficiently? While the answer to this question can find many practical applications, all current motion planners ignore the correspondences between similar environments. Our study shows that by carefully storing and reusing the computation we can gain significant efficiency.**

I. Introduction

In our everyday life, we face motion planning problems. We know how to navigate in our environment from the experiences learned since our childhood. The learning process may be complex but one of the reasons that we can learn such tasks is that most objects we encounter today are identical or similar to the objects we encountered yesterday or even years ago. That is, we as human beings, remember how to navigate around or manipulate similar objects using similar strategies.



(a) (b)

Fig. 1. Two similar workspaces sharing the same robot and several similar obstacles. The red and blue objects indicate the start and goal configurations. Similar workspaces are usually viewed as completely different problems.

In this work, we propose a method that mimics this simple observation. The goal of our work is to solve a motion planning problem, e.g., the problem in Fig. 1(b), more efficiently by reusing the computation from other similar problems, e.g., the problem in Fig. 1(a). It is important to note that the similarity in this paper is defined by the shapes of the (C-space) obstacles, not by the similarity of the free C-space.

Similar environments are common. For example, desks and chairs in a classroom or in an office may be moved around from one place to another frequently, but new items are seldom introduced. Even different environments, such as two apartments or a manufacturing factory and an airport garage, may share many similar items. The main differences are usually the arrangements. Similar environments can also be found in simulated reality, e.g., in different levels of a video game or in the different regions of a VR world, where many objects are intentionally duplicated to reduce the (e.g., modeling and rendering) complexity. A planner that exploits the similarity between its workspaces, such as the one proposed in this paper, can provide significant efficiency.

This paper also attempts to address a closely related but slightly different question: how much pre-processing can be done to solve a family of motion planning problems with similar environments? If a significant portion of the computation can be done offline, we can pre-process a very large set of geometric models (e.g., all the public 3-d models on the internet) and store the computation in a database for fast retrieval. An important consequence of this is that almost all (either known or unknown) motion planning problems can be solved more efficiently.

In this paper, we propose a motion planner (ReUse-based PRM or simply RU-PRM) that constructs local roadmaps around geometric models and stores the local roadmaps in a database. When an environment is given, the planner will match the obstacles in the environment to the existing models in the database. Then the roadmap associated with the matched models is transformed and merged into a global roadmap. In essence, our planning method solves the motion problems by reconstructing the given environment from its "memory" (i.e., the existing computation in the database).

Although the proposed method is simple and extends from the existing PRM-based frameworks, RU-PRM is unique in several ways. The most critical distinction is that much "reusable" computation is ignored by the existing planners. Given two workspaces with identical models (obstacles) but with different arrangements of these obstacles (e.g., Fig. 1), all existing planners treat these two workspaces as two distinct problems and completely ignore the correspondences between them. More detailed comparison to the related work is in Section II. In addition to the RU-PRM planner, we also propose and develop a new shape matching method (in Section V-D and the Appendix) that allows RU-PRM to perform sub-part matching and roadmap transformation more easily.

To the best of our knowledge, this is the first work dealing with similar environments. Although we consider this paper as preliminary work that provides a proof of concept in this new research direction, our experimental results are very encouraging. Our experimental results are very encouraging and show significant efficiency improvement (by 1$\sim$3 orders of magnitude faster in most of the studied environments) over the existing PRM planners.

II. Related Work

We are not aware of any planners that consider similarities among motion planning problems. There exist PRM planners that identify and learn *features* in C-free but their goal is to determine sampling strategies [1, 2, 3, 4, 5] using machine learning techniques. Given similar environments, these methods still build the roadmaps from scratch. There are also methods that pre-compute and reuse configurations for highly constrained systems (e.g., closed-chain [6]) in which feasible configurations are usually difficult to obtain. These methods do not consider similarities among motion planning problems.

Our method can also be viewed as a type of "self-improving" algorithm [7, 8]. Existing self-improving planners consider the performance for a single environment with multiple queries, e.g., [9, 10], but do not consider the performance improvement across different environments.

In the rest of this section, we will review related work in motion planning. We classify these work into methods that deal with static and dynamic environments, respectively. In our discussion, we will focus on the difficulty of applying the existing work directly to efficiently plan motions in similar environments. From our reviews, we notice that all existing methods cannot efficiently handle the problem of similar environments. In similar environments, even though the robot and the obstacles may remain the same, the arrangement of the obstacles can be totally different, all existing methods essentially treat them as distinct problems.

A. Static Environments

In this paper, we focus on the problems with static environments. The problem of motion planning in static environments has been studied extensively. It is well known that any *complete motion planners* [11], which always return a solution as long as there is a path or no if there is not, are unlikely to be practical. During the two last decades, researchers have focused on *probabilistically complete* planners, e.g., PRM [12, 13, 14], EST [15], and RRT [16], which are easy to implement and are capable of solving challenging problems. A complete review of these methods can be found in [17, 18].

Briefly, the classic PRM works by randomly generating collision-free configurations and then connecting them using local planners to form a roadmap in C-space [12]. Due to the uniform sampling strategy used by the classic PRM, the roadmap may not capture the connectivity of C-free. This is known as the "narrow passage problem." Therefore, many of the PRM variants that deal with static environments focus on problems with narrow passages. Strategies have been proposed to increase the chance of creating samples inside the narrow passage [13, 19, 20, 21]. Difficult problems, such as the alpha puzzle, become the center of study in these works. Nevertheless, these difficult problems are usually created artificially for testing purposes, and in many real-life problems, such as planning motion in a factory or in a virtual world, are usually easier and do not contain very narrow passages. Therefore, instead of focusing on these difficult but rare problems, our work attempts to increase the planner efficiency for the easier but more commonly seen problems. Our method is based on extending the existing work of PRMs with the functionality of reusing computations among similar environments.

B. Changing Environments

In changing environments, obstacles become movable [22] or even deformable [23]. The problem of changing environments can be considered as a 'continuous' version of the similar environments. If we take snapshots of the changing environments, the consecutive shots are a set of similar environments with small changes.

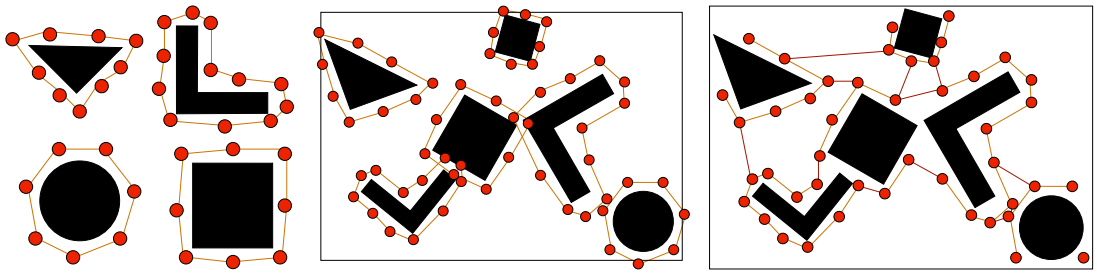
Many methods have been proposed to handle changing environments. These methods usually combine known techniques to capture both globally static and locally dynamic connectivity of C-free and can be categorized into two main frameworks: state-time space and hybrid methods. The state-time space framework is (probabilistically) complete but requires perfect knowledge of obstacle velocities and trajectory. On the other hand, the hybrid methods are more suitable for on-line planning. For example, Fiorini and Shiller [24] combine the idea of C-space with velocity obstacle and Petti and Fraichard [25] incorporate the idea of inevitable collision zone. More recent work usually uses PRMs to compute the global roadmap. In order to quickly reflect the dynamic changes in the global roadmap, Jaillet and Simeon [26] incorporate RRT to connect disconnected regions, van den Berg et al. [27] propose the idea of D*, and Leven and Hutchinson [28] construct a regular grid in workspace that maps each of its cell to the roadmap nodes and edges. Once obstacles move, [28] quickly checks occupied cells and invalidates the associated nodes and edges. Vannoy and Xiao [29] keep a set of paths and update the fitness values of the paths when obstacles move. Our method is closer to the Elastic roadmap proposed by Yang and Brock [30], where the configurations are sampled around obstacles and are moved along with the obstacles.

Most of these methods, either explicitly or implicitly, depend on the idea that the moving obstacles do not significantly affect the roadmap connectivity (at least for a short period of time), thus the planner should be able to quickly repair and replan. This assumption becomes unrealistic when obstacles are totally rearranged.

III. Our Method

In either static or dynamic environments, all existing methods compute the entire roadmap from scratch once the robot is placed in a new workspace. This is because, even though similar environments are composed of similar obstacles, a roadmap capturing the connectivity of a particular environments is unlikely to be reusable in other environments.

On the contrary, RU-PRM represents its "mental image" as a roadmap constructed for each obstacle (called ob-map) independently. For an unknown environment, RU-PRM also provides a mechanism to *transform* the ob-maps to reflect the new arrangement of the obstacles. Finally, RU-PRM solves motion planning problems by *composing* a global roadmap from all ob-maps in the new environment.



(a) a tiny ob-map database (b) transformed ob-maps (c) composed global roadmap

Fig. 2. (a) A database of pre-computed ob-maps. (b) Transformed ob-maps in a new environment. (c) A global roadmap is composed from the ob-maps.

More specifically, given a motion planning problem (O,R,Q), where O is a set of obstacles, R is the robot, and Q is a query, RU-PRM first looks for an existing computation (ob-map) for each obstacle $O_i \in O$. If the same or a similar obstacle O_i' for O is found in the database, we reuse the computation by transforming the ob-map constructed for O_i'. If there is no such an obstacle found in the database, we compute an ob-map for O_i and store it in the database. To construct ob-maps, we use the idea of obstacle-based PRMs [19, 13], which sample configurations on or near the surface of the C-space obstacles (C-obst). Once all the ob-maps are either loaded and transformed or constructed for all the obstacles in the workspace, these ob-maps are merged into a global roadmap, which then is used to answer the query Q. Fig. 2 illustrates how RU-PRM works. Algorithm III.1 summarizes the main steps of RU-PRM.

Algorithm III.1: RU-PRM(O,R,Q)

comment: Obstacles O, Robot R and query Q

for each $O_i \in O$

do
- **if** $\nexists M_{O_i}$, an ob-map of O_i
  - **then**
    - Create(M_{O_i})
    - Store(M_{O_i})
- Read(M_{O_i})
- Transform(M_{O_i})

$M_O \leftarrow$Merge$(\cup_i\{M_{O_i}\})$

Query(M_O, Q)

In Algorithm III.1, the sub-routines Store$(\cdot)$, Read$(\cdot)$, and Query$(\cdot)$ are straightforward. The other three main sub-routines: Create$(\cdot)$, Transform$(\cdot)$ and Merge$(\cdot)$ will be discussed in detail in Section IV.

For the rest of this section, we will provide a short discussion on the necessary conditions and assumptions that make RU-PRM efficient. When a given problem does not satisfy these assumptions, RU-PRM degrades to the traditional PRM.

A. Assumptions

The benefits provided by our method is based on the following assumptions and conditions.

1) The robot remains the same in the similar environments.
2) The unknown workspace $\mathscr{W}_i$ has high correspondences to other known workspaces $\mathscr{W}_1 \cdots \mathscr{W}_{i-1}$.
3) A large storage space is available to store all ob-maps.
4) Pre-processing time is available.

These assumptions are general enough to cover many practical situations. As we have pointed out earlier, the first two assumptions are from the observation that many motion planning problems in real life and in virtual worlds share many similar items in their workspaces. Moreover, the type and the number of the robots used in these environments, e.g., characters in a game or robot arms in a factory, are usually limited and do not change often.

Other assumptions are also supported by the current technologies. Most off-the-shelf hard-drive disc with Tera-byte capacity can be obtained for just a few hundred US dollars. Multi-core processors are becoming cheaper and allow more background computation for creating a database of ob-maps. In addition, due to the advances of modeling software and digitizing techniques, constructing complex geometric models becomes easier than ever. Several 3-d geometric databases, e.g., [31], that contain thousands of geometric models, are also available to the public and provide us a set of common objects, e.g., desks, tables, and chairs, to bootstrap the proposed planner.

RU-PRM combines all these ingredients and reuses the computation from the pre-processed geometric models. We envision that our method will allow these computation to be shipped with a new robot or a new character (e.g., in a video game). As far as we know this is the first work dealing with this types of problems. In this paper, we present a preliminary work to provide a proof of the concept in this research direction.

IV. CREATE, TRANSFORM AND MERGE OB-MAPS

In this section, we present three main sub-routines for RU-PRM in their basic forms. Advanced techniques for optimizing the efficiency of these operations will be discussed in the next section (Section V).

A. Create Ob-maps

Creating roadmaps around C-obst allows us to reuse the computation when the same or similar obstacles are given. We call these roadmaps ob-maps. There exist several planners

creating ob-maps, e.g., OBPRM [13] and Gaussian PRM [19]. Although RU-PRM can work with any of these planners, in the experiment shown in Section VI, we use both Gaussian PRM and the Minkowski sum based approach (called MSUM-PRM) [32]. Because Gaussian PRM is well known, we will briefly describe the idea of MSUM-PRM.

MSUM-PRM samples configurations by computing the *Minkowski sums* of the robot and the obstacles. It is known that the contact space of a *translational* robot is the boundary of Minkowski sum of the obstacles and the negated copy of the robot [33]. Although it is difficult to compute the exact Minkowski sums of polyhedra [34], we have shown that sampling points from the Minkowski sum boundary without explicitly generating its mesh boundary, can be done much more easily and efficiently [35]. To handle the case of non-translational robots, we simply draw samples from a set of Minkowski sums, each of which is constructed by randomly assigning different orientations and joint angles to the robot.

In [32], we have shown that MSUM-PRM generates a free configuration significantly faster than OBPRM and Gaussian PRM. We have also demonstrated that these configurations can be connected into roadmap using more powerful local planners based on the geometric properties of the Minkowski sum. We refer the interested reader to [32] for details regarding this sampling strategy.

MSUM-PRM provides an important advantage for RU-PRM that is missing from the other obstacle-based samplers. A configuration created by MSUM-PRM can be transformed easily because the configuration is represented by a pair of points from the robot and an obstacle. Details of the transformation operation will be discussed in the next section.

B. Transform Ob-maps

When an unknown obstacle X is matched to an existing model O by translating, rotating, and scaling O, we consider how these transformations can affect O's ob-map (denoted as M_O) for free-flying robots. More precisely, we seek methods that transform M_O to approximate the ob-map of X.

We let CO be the C-obst of O. We further let T, R, and S be the translation, rotation and uniform scale applied to O, respectively. Given a configuration c from ∂CO, our goal is to obtain the corresponding configuration c' of c so that c' is in ∂CO of O transformed using T, R and S.

To ease our discussion, we assume that (1) when we generate M_O, the center of O is placed at the origin of the coordinate system and (2) the robot is a free-flying articulated robot. The configuration c is composed of three components (c_T, c_{BR}, c_{JR}), where c_T and c_{BR} are the values of the translational and rotational degrees of freedom (dof) for the base, respectively. and c_{JR} represents the rotational dof of the joints.

When we consider only the translation T and rotation R, obtaining the corresponding c' is straightforward, i.e.,

$$c' = (T + R \cdot c_T, R \cdot c_{BR}, c_{JR}) \ . \tag{1}$$

Note that T and R have no effect on c_{JR}.

When we consider the scale S, the only component of c affected by S is c_T. Therefore, c' will have the form of (c'_T, c_{BR}, c_{JR}). However, it is not always possible to obtain c'_T that can place c' in ∂CO after O is scaled. In fact, this is only possible when (1) both the robot and O are *convex* or (2) O is convex and S shrinks O.

Because the contact space of the robot can be represented by the Minkowski sum operation, we can always decompose c_T so that $c_T = r + o$, where r and o are points from the boundary of $-R$ and O, respectively. Here, R is the robot and $-R = \{-x \mid x \in R\}$. When R, O and S satisfy the aforementioned requirements, c' is in the contact space by letting

$$c'_T = r + S \cdot o \ . \tag{2}$$

If O and R are not convex or if O is convex but S enlarges O, we can work around this problem using convex decomposition. However, convex decomposition can slow down the computation significantly and can generate a lot of components. Therefore, instead of decomposing exactly, we use approximate convex decomposition [36] and represent the model using the convex hulls of the components in the decomposition. Details of this approach is discussed in the Appendix when we describe our shape matching method.

Alternatively, we can apply Eq. 2 to non-convex shapes. The consequence of this is that c' may make the robot collide with $S(O)$. Therefore, collision detection is used to check the feasibility of every transformed configuration.

C. Merge Ob-maps

So far, we have treated each obstacle independently. After the pre-processing step that either loads or generates ob-maps, we proceed to compose a global roadmap.

The configurations in a (transformed) ob-map, although are near the surface of the associated obstacle, may be outside the bounding box or colliding with other obstacles. We validate each configuration in an ob-map using a collison detector. For edges connecting two collision-free configurations, we evaluated them in a lazy manner [37], i.e., we only check the feasibility of the edges in the extracted paths during the query phase.

After all configurations are verified, we merge the ob-maps pairwisely. For every pair of obstacles O_i and O_j in the workspace, we connect their ob-maps by simply adding edges for k pairs of the closest configurations. Each pair consists of a configuration from the ob-map of O_i and a configuration from the ob-map of O_j. Again, we do not check the feasibility of these edges, although some edges may collide with the obstacles.

Once the global roadmap is constructed, we iteratively extract and evaluate new paths and delete invalid edges from the roadmap until a collision-free path is found, otherwise "no solution" will be reported.

V. PLANNER OPTIMIZATION

In the previous section, the basic framework of RU-PRM is described. In this section, we will present several optimization

techniques to improve the efficiency of the framework. We also present a new shape matching method using approximate convex decomposition.

A. Create Ob-maps with Quality Control

Like most PRM-based methods, the quality of the roadmap generated by MSUM-PRM also depends on the number n of configurations sampled. The value of n is usually provided by the user based on the "difficulty of the problem." However, when we consider similar workspaces, their difficulties may not be known at the ob-map creation time. Therefore, requiring users to specify the value of n not only puts too much burden on the users, but it is also unrealistic and impractical.

Xie et al. [38] have attempted to remove this limitation from PRMs by incrementally increasing the resolution of a roadmap. Their method adds and connects additional nodes to the roadmap until certain criteria are met. We follow the same approach to avoid specifying the value of n. In each iteration of our map generation process, we add and connect n_i new configurations to the existing ob-map (which is initially empty) using MSUM-PRM. Then, we check the number n_{CC} of connected components that comprise $m\%$ of the configurations in the map. If n_{CC} is decreasing for the last z consecutive iterations, we stop. Note that, although it seems that we introduce more parameters (n_i, m and z) by constructing ob-maps incrementally, these parameters are easier to set and usually remain fixed regardless the workspace.

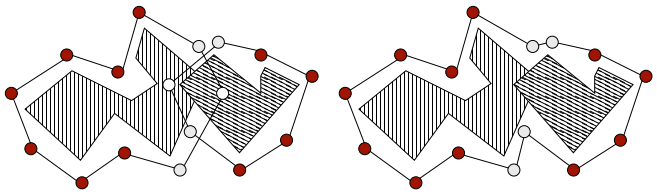
We realize that this simple heuristic method may not handle all possible scenarios properly. However, from the results of our experiments, the ob-maps generated by this method cover the obstacles well, and, in most examples, ob-maps usually contain a single connected component.

B. Transform Ob-maps Hierarchically

An ob-map may still contain many nodes. Since we do not know in advance if all these nodes are necessary. For example, in an open workspace, only few of these nodes are needed to solve the problem. Transforming and validating all the nodes in the ob-map significantly slows down the computation. However, in a cluttered workspace, it is necessary to keep all the nodes in order to capture the global connectivity.

To address this issue, we organize the roadmap nodes into a hierarchy and process these nodes in the same manner. More specifically, we compute the core of an ob-map, which is a sub-graph that maintains the same connectivity as the ob-map. In a new workspace, all the nodes in the core will be transformed and their feasibility will be checked. When the core does not have the desired connectivity, we repair the core by transforming additional nodes from the ob-map.

The core is constructed by determining and removing the redundant nodes in the ob-map. Given a pair of adjacent nodes u and v in the ob-map, we say that v is redundant if the roadmap contains the same number of connected components after (1) removing v and (2) connecting u to v's adjacent nodes without colliding with the associated obstacle. Finally, the core is simply a sub-graph of the ob-map without redundant nodes. We store the core with the ob-map in the database.



(a) (b)

Fig. 3. Merge the ob-maps of two overlapping obstacles. (a) Before merging. Free, collision, and boundary nodes are shown in dark gray, white and light gray, respectively. (b) After merging, the boundary nodes are connected.

In a new workspace, the core is loaded from the database first. After transformation and evaluation, the core may be split into many connected components due to the removal of in-collision nodes. When this happens, more nodes are added to the core from the original ob-map. Let CC_1 and CC_2 be two connected components in the core, we improve the core's connectivity by finding a path in the ob-map that connects a pair of nodes from CC_1 and CC_2. This process repeats until both the core and the ob-map have the same number of connected components or when no path in the ob-map can be found.

C. Merge Ob-maps Using Boundary Nodes

The merging operation can also be optimized by classifying the relationships of the C-space obstacles. Let O_i and O_j be a pair of obstacles in the workspace, we can classify their relationships *in C-space* based on the feasibility of the configurations in their ob-maps, M_{O_i} and M_{O_j}. More specifically, we identify the *boundary nodes* in the ob-maps. Let $B_{\{O_i,O_j\}}$ be a set of boundary nodes in M_{O_i} w.r.t. O_j. For each configuration $c \in M_{O_i}$, we say $c \in B_{\{O_i,O_j\}}$ if an adjacent node of c makes the robot collide with O_j. Fig. 3 shows an example of the boundary nodes. Note that $B_{\{O_i,O_j\}} \neq B_{\{O_j,O_i\}}$ unless they are both empty. We use the boundary nodes to classify and connect the ob-maps. There are three cases to consider:

1) M_{O_i} and M_{O_j} are far away from each other, i.e., $B_{\{O_i,O_j\}} = B_{\{O_j,O_i\}} = \emptyset$.
2) M_{O_i} and M_{O_j} overlap, i.e., $B_{\{O_i,O_j\}} \neq \emptyset$ and $B_{\{O_j,O_i\}} \neq \emptyset$.
3) M_{O_i} and M_{O_j} are near each other, i.e., $B_{\{O_i,O_j\}} \neq \emptyset$ and $B_{\{O_j,O_i\}} = \emptyset$ or vice versa.

Case 1. When the C-obsts of O_i and O_j are far away from each other, we connect their ob-maps using the method described in Section IV-C, which simply connects the k-closest pairs between the ob-maps.

Case 2. This is the situation that the C-obsts of O_i and O_j overlap. The ob-maps are connected by adding edges between $B_{\{O_i,O_j\}}$ and $B_{\{O_j,O_i\}}$. Fig. 3 shows an example in this case.

Case 3. This situation requires us to combine the techniques for **Case 1** and **Case 2**. For $B_{\{O_i,O_j\}} \neq \emptyset$ and $B_{\{O_j,O_i\}} = \emptyset$, ob-maps are connected by adding edges between $c_i \in B_{\{O_i,O_j\}}$ and its k closest nodes $c_j \in M_{O_j}$. For $B_{\{O_i,O_j\}} = \emptyset$ and $B_{\{O_j,O_i\}} \neq \emptyset$, ob-maps are merged in the same manner.

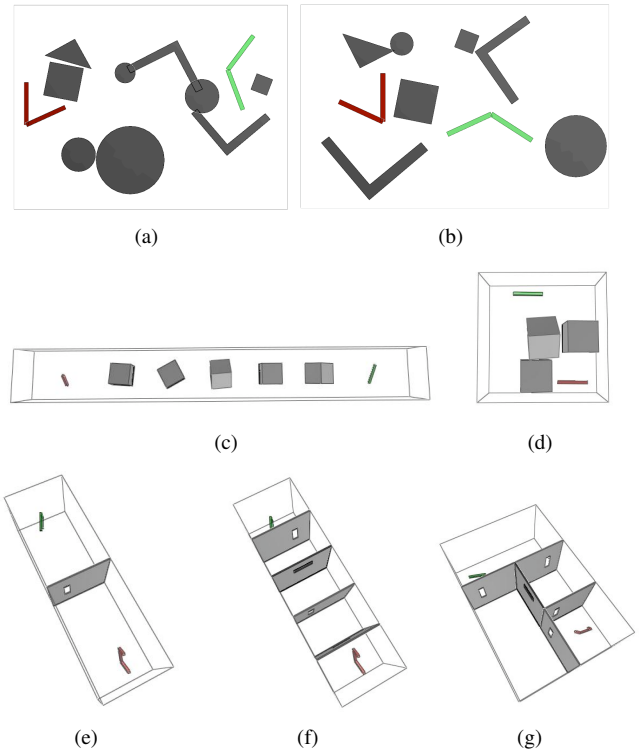


Fig. 4. Environments used in the experiments. (a, b) 2-d workspaces with an articulated robot. (c, d) Similar workspaces with a rigid robot and cubes. (e, f, g) Similar workspaces with a U-shaped rigid robot and walls.

D. Shape Matching using Approximate Convex Decomposition

So far, we have assumed that the same obstacle can be found in the database. To reuse ob-maps for different but similar obstacles, we propose a new shape matching method. Essentially, our matching method estimates the dissimilarity of two objects by measuring their morphing distance. The proposed matching method is composed of two main steps: (1) decomposing shapes using approximate convex decomposition (ACD), and (2) computing dissimilarity using morphing and bipartite matching. Details of the shape matching method and some preliminary results are discussed in the Appendix.

VI. Implementation and Experimental Results

In this section, we show experimental results. All the experiments are performed on a PC with two Intel Core 2 CPU at 2.13 GHz with 4 GB RAM. Our implementation is coded in C++. Our current implementation only supports free-flying robots and 2-d shape matching.

We test RU-PRM in three sets of similar environments (see Fig. 4). We assume that the robots and the obstacles in these environments are known, therefore we can pre-process them and store their ob-maps in a database. We use two types of RU-PRMs in our experiments: (1) RU-PRM with Gaussian PRM and (2) RU-PRM with MSUM-PRM. During the pre-processing step, RU-PRM with Gaussian PRM takes about 4.9, 96, and 382 seconds to create and store the ob-maps for the obstacles in Env. (a-b), Env. (c-d), and Env. (e-g), respectively. RU-PRM with MSUM-PRM takes about 35 and 131 seconds to create and store the ob-maps for the obstacles in Env. (c-d) and Env. (e-g), respectively. Because currently MSUM-PRM has not been implemented to handle 2-d workspaces, RU-PRM with MSUM-PRM is not used in Env. (a-b). In these experiments, we compare RU-PRMs against several well-known planners: MSUM-PRM [32], Uniform [12], Gaussian [19] and Visibility PRMs [39].

RU-PRM is significantly faster The running times for all three environments are shown in Fig. 5. Notice that the *y*-axes of the charts in Fig. 5 are in logarithmic scale. The running time for RU-PRMs includes the time for reading the ob-maps from the hard disk drive, as well as the time for transforming, evaluating and merging the ob-maps and the time for performing the query in the global roadmap.

From the results, RU-PRM shows significant efficiency improvement in *all* studied environments. Env. (a-b) and Env. (c-d) are simple environments where the Uniform PRM usually solves the problems with a few hundred nodes and outperforms Gaussian and Visibility PRMs and MSUM-PRM. In these simple environments, RU-PRM with MSUM-PRM or Gaussian PRM still provides noticeable improvements (up to 100 times faster) over the Uniform PRM (and therefore over Gaussian PRM only and MSUM-PRM only). This evidence demonstrates the strength of reusing computation. These plots also shows that combining RU-PRM with either MSUM-PRM or Gaussian PRM does not seem to affect its performance in simple environment, however, the difference becomes more noticeable in more difficult environments.

In more difficult environments, e.g., Env. (e-g) that contains narrow passages, RU-PRM with MSUM-PRM or Gaussian is significantly faster (by 5$\sim$8 orders of magnitude) than Uniform and Gaussian PRMs and is still significantly faster than using MSUM-PRM or Gaussian only (by 2$\sim$5 orders of magnitude). A possible source of the improvement is from MSUM-PRM, which has been shown to be better than the classic PRMs [32], however, our results also show that combining RU-PRM with MSUM-PRM further improves MSUM-PRM. The main reason for this performance improvement is obvious. Obtaining connectivity in the free C-space is usually the most time-consuming step in PRMs, RU-PRM saves significant amount of time by reusing the connectivity provided by the ob-maps.

VII. Conclusion

In this paper, we proposed the first work considering motion planning in similar environments. We developed a new method called RU-PRM that reuses the computation from the previously solved problems. Essentially, RU-PRM stores the local roadmap built around each C-obst. When a new environment is given, RU-PRM matches the obstacles and loads the matched roadmaps. These roadmaps are transformed, evaluated and finally merged to solve the queries in the new environments. We discussed several optimization techniques that improve the efficiency of the basic RU-PRM framework. We also proposed a new shape matching method that allows RU-PRM to do sub-part matching and roadmap transformation more easily.

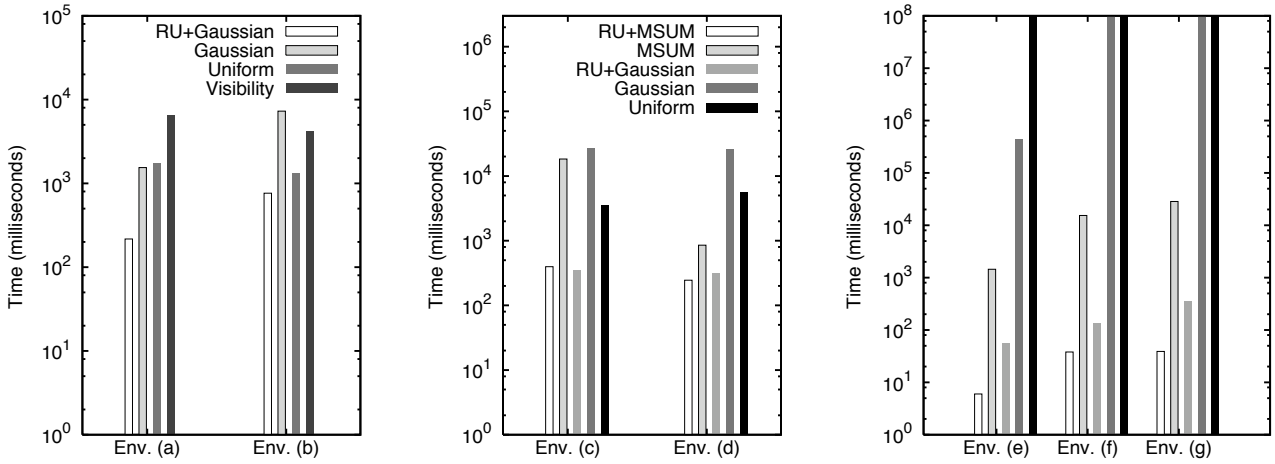


Fig. 5. Experimental results for the environments in Fig.4. Notice that the y-axes are all in logarithmic scale and Envs. (c)-(g) share the same legend. We stop the planner when it takes more than 10^8 milliseconds. The running times for all PRMs are collected over 30 runs.

Although we consider this paper as a preliminary work that provides a proof of the concept in this new research direction, our experimental results are very encouraging and show significant efficiency improvement (by 5∼8 orders of magnitude faster) over the classic PRM planners (including Uniform, Gaussian, and Visibility PRMs) and is faster then MSUM-PRM by 3∼5 orders of magnitude.

Limitations and Future work. Many aspects in the proposed work can be improved. One of the most critical bottlenecks of our current implementation is the efficiency of the shape matching method. We also plan to investigate better criteria for evaluating the connectivity of the ob-map and its core.

REFERENCES

[1] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato, "A machine learning approach for feature-sensitive motion planning," in *Proc. Int. Workshop Alg. Found. Robot.(WAFR)*, Utrecht/Zeist, The Netherlands, July 2004, pp. 361–376.

[2] B. Burns and O. Brock, "Toward optimal configuration space sampling," in *Proc. Robotics: Sci. Sys. (RSS)*, 2005.

[3] A. Yershova, L. Jaillet, T. Simeon, and S. M. Lavalle, "C-space subdivision and integration in feature-sensitive motion planning," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, April 2005, pp. 3867–3872.

[4] S. Rodriguez, S. Thomas, R. Pearce, and N. M. Amato., "(resampl): A region-sensitive adaptive motion planner," in *Proc. Int. Workshop Alg. Found. Robot.(WAFR)*, July 2007.

[5] S. Finney, L. Kaelbling, and T. Lozano-Perez, "Predicting partial paths from planning problem parameters," in *Proceedings of Robotics: Science and Systems*, Atlanta, GA, USA, June 2007.

[6] L. Han and N. M. Amato, "A kinematics-based probabilistic roadmap method for closed chain systems," in *Robotics:New Directions*. Natick, MA: A K Peters, 2000, pp. 233–246, book containts the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), Dartmouth, March 2000.

[7] N. Ailon, B. Chazelle, S. Comandur, and D. Liu, "Self-improving algorithms," in *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*. New York, NY, USA: ACM, 2006, pp. 261–270.

[8] K. L. Clarkson and C. Seshadhri, "Self-improving algorithms for delaunay triangulations," in *SCG '08: Proceedings of the twenty-fourth annual symposium on Computational geometry*. New York, NY, USA: ACM, 2008, pp. 148–155.

[9] T.-Y. Li and Y.-C. Shie, "An incremental learning approach to motion planning with roadmap management," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, 2002, pp. 3411–3416.

[10] R. Gayle, K. R. Klingler, and P. G. Xavier, "Lazy reconfiguration forest (LRF): An approach for motion planning with mulitple tasks in dynamic environments," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, 2007, pp. 1316–1323.

[11] J. F. Canny, *The Complexity of Robot Motion Planning*. Cambridge, MA: MIT Press, 1988.

[12] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, August 1996.

[13] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo, "OBPRM: An obstacle-based PRM for 3D workspaces," in *Robotics: The Algorithmic Perspective*. Natick, MA: A.K. Peters, 1998, pp. 155–168, proc. Third Workshop on Algorithmic Foundations of Robotics (WAFR), Houston, TX, 1998.

[14] N. M. Amato and Y. Wu, "A randomized roadmap method for path and manipulation planning," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1996, pp. 113–120.

[15] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," *Int. J. Comput. Geom. & Appl.*, pp. 2719–2726, 1997.

[16] S. M. LaValle and J. J. Kuffner, "Rapidly-Exploring Random Trees: Progress and Prospects," in *Proc. Int. Workshop Alg. Found. Robot.(WAFR)*, 2000, pp. SA45–SA59.

[17] R. Geraerts and M. H. Overmars, "Sampling techniques for probabilistic roadmap planners," in *Intelligent Autonomous Systems (IAS)*, 2004, pp. 600–609.

[18] S. M. LaValle, *Planning Algorithms*, 6th ed. Cambridge University Press, 2006.

[19] V. Boor, M. H. Overmars, and A. F. van der Stappen, "The Gaussian sampling strategy for probabilistic roadmap planners," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, vol. 2, May 1999, pp. 1018–1023.

[20] S. A. Wilmarth, N. M. Amato, and P. F. Stiller, "MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, vol. 2, 1999, pp. 1024–1031.

[21] D. Hsu, T. Jiang, J. Reif, and Z. Sun, "Bridge test for sampling narrow passages with proabilistic roadmap planners," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, 2003, pp. 4420–4426.

[22] J. van den Berg and M. Overmas, "Roadmap-based motion planning in dynamic environments," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 2004, pp. 1598–1605.

[23] S. Rodriguez, J.-M. Lien, and N. M. Amato, "Planning motion in

completely deformable environments," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, May 2006, pp. 2466–2471.
[24] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *Int. Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, 1998.
[25] S. Petti and T. Fraichard, "Safe motion planning in dynamic environments," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 2005, pp. 2210–2215.
[26] L. Jaillet and T. Simeon, "A prm-based motion planner for dynamically changing environments," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 2004, pp. 1606–1611.
[27] J. van den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2006, pp. 2366 – 2371.
[28] P. J. Leven and S. Hutchinson, "A framework for real-time path planning in changing environments," *Int. Journal of Robotics Research*, vol. 21, no. 12, pp. 999–1030, 2002.
[29] J. Vannoy and J. Xiao, "Real-time planning of mobile manipulation in dynamic environments of unknown changes," in *Proc. Robotics: Sci. Sys. (RSS)*, 2006.
[30] Y. Yan and O. Brock, "Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation in dynamic environments," in *Proc. Robotics: Sci. Sys. (RSS)*, 2006.
[31] M. K. Philip Shilane, Patrick Min and T. Funkhouser, "The princeton shape benchmark," in *Proceedings of the Shape Modeling International*, 2004.
[32] J.-M. Lien, "Hybrid motion planning using Minkowski sums," in *Proc. Robotics: Sci. Sys. (RSS)*, Zurich, Switzerland, 2008.
[33] T. Lozano-Pérez, "Spatial planning: A configuration space approach," *IEEE Trans. Comput.*, vol. C-32, pp. 108–120, 1983.
[34] G. Varadhan and D. Manocha, "Accurate Minkowski sum approximation of polyhedral models," *Graph. Models*, vol. 68, no. 4, pp. 343–355, 2006.
[35] J.-M. Lien, "Point-based minkowski sum boundary," in *PG '07: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 261–270.
[36] J.-M. Lien and N. M. Amato, "Approximate convex decomposition," in *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*. New York, NY, USA: ACM Press, 2004, pp. 457–458, video Abstract.
[37] R. Bohlin and L. E. Kavraki, "A randomized algorithm for robot path planning based on lazy evaluation," in *Handbook on Randomized Computing*, P. Pardalos, S. Rajasekaran, and J. Rolim, Eds. Kluwer Academic Publishers, 2001, pp. 221–249.
[38] D. Xie, M. A. Morales, R. Pearce, S. Thomas, J.-M. Lien, and N. M. Amato, "Incremental map generation (IMG)," in *Proc. Int. Workshop Alg. Found. Robot.(WAFR)*, July 2006.
[39] C. Nissoux, T. Simeon, and J.-P. Laumond, "Visibility based probabilistic roadmaps," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 1999, pp. 1316–1321.
[40] R. C. Veltkamp and M. Hagedoorn, "State of the art in shape matching," *Principles of visual information retrieval*, pp. 87–119, 2001.

APPENDIX

Shape approximation using ACD In the first step, we decompose the polygon P into several approximately convex pieces [36]. A component C in the decomposition is approximately convex if the concavity of C is less than a user pre-defined tolerance. It has been shown that ACD contains much fewer components compared to the exact decomposition and, more importantly, it maintains the key structural features. Because of these advantages, we can represent P using the convex hulls of the components in its ACD. An example of ACD is shown in Fig. 6(a) and (b).

We first compute the dissimilarity between a pair of *convex* components by estimating their *morphing* distance. Let $\{C_i\}$ and $\{D_j\}$ be two sets of convex components obtained from the ACDs of the polygons P and Q. Given two convex components C_i with m vertices (colored in green in Fig. 6(d)) and D_j with n vertices (colored in blue in Fig. 6(d)), we compute the morphing distance as the follows:

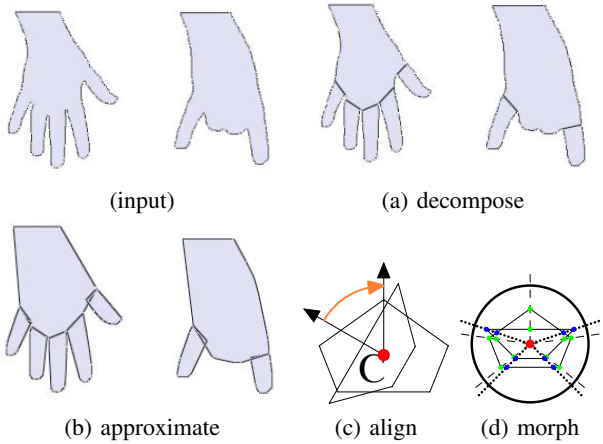


Fig. 6. An example of the proposed shape matching method.

1) Align C_i and D_j by overlapping their centers and their principal axes, (see Fig. 6(c)).
2) Draw m rays from the common center to the vertices of C_i and n rays to those of D_j, (see Fig. 6(d)).
3) Compute the intersections of the rays with vertices C_i and D_j. For each ray, a line segment between the intersection and the vertex is identified.
4) Let the morphing distance of C_i and D_j be the sum of the lengths of all $m+n$ segments.

Once the morphing distances between all pairs of components in $\{C_i\}$ and $\{D_i\}$ are computed, we arrange the components $\{C_i\}$ and $\{D_i\}$ into a bipartite graph and solve the minimum bipartite matching problem.

Note that although we only demonstrate using polygons in this section, the proposed matching method can be extended to handle polyhedra without modification.

Although there exist many shape matching methods (e.g., see survey [40]), this new method provides unique functionalities for RU-PRM. In particular, it represents shape using a set of convex objects. This not only allows the scale transformation discussed in Section IV-B, but also allows subpart matching. For example, the ob-map for a hand gesture can be "deformed" to fit around another gesture by transforming the fingers even if the gestures are very different. Due to the space limitation, we will skip the technical details of the proposed method.

The proposed method correctly returns its best match for every test case using 10 randomly selected polygons (shown in Fig.7). Moreover, it takes 70 ms on average for a polygon to find the best match. Therefore, even when we consider the time spent on shape matching, RU-PRM still outperforms the other three planners.

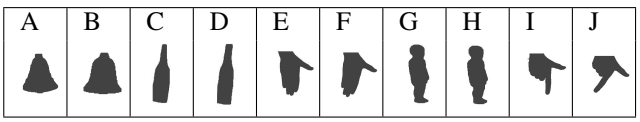


Fig. 7. Matching Results. 10 models used in this shape matching experiments.