Proposal Abstract: Reusable Methods for Handling Uncertainty in Robot Manipulation Planning

Christopher Vo

1 Background

One of the fundamental goals in robotics is to make it possible for robots to interact with objects in the real world. Such tasks usually require the solution to a manipulation planning problem—simply put, the problem of how to tell a robot to move an object from configuration $A$ to configuration $B$. In manipulation planning, it is common to assume perfect knowledge of the geometry and dynamics of world, and the configuration of the objects within. These assumptions are reasonable when robot operates in a highly controlled environment, has perfect sensors, or exists only in a simulation. However, these assumptions are not valid in reality. For example, many current manipulation planning methods make use of approximate models of the geometry and dynamics of the world, whose accuracy is unknown—subject to modeling uncertainty. The movement of the robot and objects it manipulates is imperfect, introducing action uncertainty. This can also lead to uncertainty in the configuration of the objects relative to the robot, or object pose uncertainty. Some planning methods use information gathered from a sensor to decide what to do about unexpected situations during execution of a movement, but this information is also susceptible to sensor uncertainty. These various kinds of uncertainties can make it difficult or perhaps impossible to guarantee successful completion of the given task. My proposed research is to develop techniques to increase the effectiveness of manipulation planning successfully when faced with various types of uncertainty.

In general, the difficulty in estimating the configuration of the system in manipulation planning has led many researchers towards methods that attempt to greatly restrict the way the robot interacts with the obstacles. For example, to reduce the burden of estimating the object’s state, many algorithms require a robot to grasp an object before moving it. Planning grasps is a difficult problem in itself, and there are many works that have considered manipulation via grasping, including under uncertainty. For example, Lozano-Pérez, Mason, and Taylor have developed the preimage planning framework for performing manipulation planning under uncertainty [7, 26]. A preimage is a region in space such that if the robot executes a motion within
that region, it is always guaranteed to reach a given goal region regardless of any source of uncertainty. This leads to the idea of preimage backchaining, where the planner starts by finding preimages of the goal, then preimages of those preimages, and so on recursively, until the initial region is covered. There are also other variations of the grasping problem. For example, Platt et al. also considered the problem of Simultaneous Localization and Grasping, where the robot must attempt to estimate the position of the object while a grasp is occurring [30].

More recently, probabilistic reasoning has been introduced for sensor-based grasping, such as the use of particle filters to estimate the changing probability distributions of the pose of the object and find good grasps [22]. In related work, Feiten et al. in [10] presented how to use a convex combination of Projected Gaussians in lieu of multivariate Gaussian distributions to represent the probability density function (pdf) of a 6D pose consisting of a 3D translation component and an orientation. Such a representation permits easier and more efficient calculation of compositions of rigid motions from the pdfs of individual rigid motions.

While grasping is a popular way to manipulate objects is also very restrictive, and in fact may not be possible for particularly heavy or unwieldy objects. There are many other natural ways to manipulate objects, such as by pushing, shaking, sliding, or throwing them. Therefore, it is also of interest to perform nonprehensile manipulation, or manipulation without grasping. Recently, Dogar and Srinivasa introduced a framework for planning in clutter using a library of actions derived analytically from the mechanics of pushing [6]. There have also been several works focused on estimation of state despite the lack of reliable sensor information. For example, Erdmann and Mason have explored sensorless manipulation [8], where by dropping an object into a tray and performing a sequence of tilting operations, the number of possible orientations is reduced; sometimes leaving the orientation of the object completely determined. Similarly, Goldberg and Mason [11, 12, 13] have explored manipulation of objects by squeezing. Because of the way that the object complies to the squeezing operations, it may be possible to design a sequence of squeezing operations that leads to a known orientation of the object.

There also exist many techniques for developing more robust plans in traditional motion planning problems. Perhaps the most common approach for dealing with uncertainty is to reduce the planning horizon so that replanning can be performed quickly during execution. Along the same lines, some researchers have focused on implicitly dealing with uncertainty of future states through using a concept known as feedback motion planning [23, 40]. Unlike traditional motion planners, which typically generate a path and execute it in an open-loop fashion, feedback planners use information about the state space to generate a navigation function that maps every state of the system to an action. These mappings often resemble vector fields, compositions
of funnels, and gradient maps. To execute a task, the robot must repeatedly estimate the state of the system, and execute the action that the navigation function prescribes for that state. By covering all possible states, feedback planners implicitly offer a recovery plan for unexpected situations that may arise by errors in the motion of the robot.

In some planning methods, uncertainty is handled through explicit modeling or by taking advantage of a-priori knowledge of the sensors and controller that will be used to execute the task. This kind of modeling typically allows the planner to measure the quality of a path, and select a path from a set of candidate paths with respect to that measure. For example, [1, 34] presents an approach for assessing the quality of a path. By assuming a linear-quadratic controller (LQG-controller) and Gaussian models of motion and sensing uncertainty, they derive the probability distributions for the states and control inputs along a given path, and this information can be used to determine the probability that collisions will be avoided, or the probability that the robot will arrive at the goal.

Another popular approach to handling uncertainty in motion planning is to model the problems as Partially Observable Markov Decision Processes (POMDPs). POMDPs model the problem as one of an agent moving through a state space by taking actions. The uncertainty of the end state of actions, and the uncertainty of observation are explicitly modeled using conditional probability functions. At each step, the agent receives a reward. The solution of these POMDPs is an optimal policy that maximizes the expected total reward. Due to the curse-of-dimensionality and the general difficulty in representing such probability distributions, POMDPs are notoriously difficult to solve exactly. However, over the years, many approximation algorithms [15] and point-based algorithms such as HSVI2 [32] and SARSOP [21] have been shown to handle up to 100,000 states in reasonable time [20].

Another general approach to handle uncertainty is to use a hybrid (or “hierarchical”) motion planning approach, such as in [18, 5]. This approach is similar to three-layer architectures [31], where control is separated into layers with different levels of precision and length of planning horizon. For example, it is typical to construct a planner that uses local reactive control to handle immediate geometric constraints, and a global symbolic planner that handles high-level task specifications. One interesting method that uses this technique is temporal logic motion planning [9]. There, the planner obtains a discrete, topological abstraction using a projection—for example, through a space partitioning or cell decomposition method. The planner then computes a high-level plan in this discrete space using automata theory, and then implements each step of the plan in the continuous space using local feedback control. The hierarchy of using a high-level planner with low-level controls allows such a planner to handle complicated high level temporal logic constraints.
while also providing robust feedback control for the robot to move safely between space partitions.

An interesting representation for uncertainty in motion planning is the use of uncertainty roadmaps, which are graph data structures that capture the probability of successfully transitioning from one valid configuration (or set of configurations) to another. With this information stored in a roadmap, discrete path finding algorithms such as A* can be used to query paths that balance cost and uncertainty, such as in [3].

While there are many techniques for dealing with uncertainties, the overhead of computing robust plans is high. Planners that rely on explicitly modeling uncertainty are often complex and difficult to generalize. On the other hand, planning methods that implicitly model uncertainty, such as feedback motion planners, must deal with the issue of coverage—that is, how to offer the best action for each state. In my proposed research, I intend to develop more efficient methods that harness a hierarchical approach along with uncertainty roadmaps to improve the success rate of manipulation planning under pose uncertainty. This research is motivated by the notion that sampling-based models are generally less complex to design than analytical models, and are often amenable to speedup via parallelization. The sampled data from manipulating objects can be used in a many ways, from building uncertainty roadmaps, to guiding sampling and increasing the effectiveness of search.

2 Overview of Completed Work

In my preliminary research, I have experimented with several manipulation planning problems that each involve different approaches and models for robustness.

2.1 Group Motion Control

In [35], we experimented with the group motion control problem. This problem involves moving one group of agents (or shepherds) to control or guide another group of agents (or flock) through a continuous, obstacle-filled environment. An example scenario is illustrated in Figure 1. This problem is difficult for many reasons. First, the system is high-dimensional, as it involves many independently-moving agents. The flock is also indirectly controlled, so the system is considered underactuated. The motion of each agent affects the motion of its surrounding agents, so the system is also considered to be differentially constrained. For this problem, we developed several hybrid approaches that combine the high-level sampling motion planners EST [17], RRT [24], and PRM [19] with reactive local shepherding behaviors. We found our approach to be more effective than simply using the medial axis as a guide for the local behaviors. In this work, we also introduced
a method based on the concept of a meta-graph. A meta-graph is a type of uncertainty roadmap. It attempts to capture the uncertainty in the pose of the flock members by considering each node of the graph as a set of similar configurations (hence, it is called a meta-node). Considering sets of similar configurations rather than single configurations reduces the sensitivity of the roadmap to small perturbations or uncertainties in the configuration of the flock. The directed edges of the meta-graph are weighted to represent the probability of successfully herding the flock from a configuration conforming to the source meta-node, to a configuration conforming to the destination meta-node. These probabilities are computed through sampling repeated simulations.

Figure 1: Illustration of the shepherding problem. The shepherds (green) must manipulate the flock (blue) around obstacles (gold and purple) to a goal position (red).

In [14], we extended the work of [35] further by representing the large group of agents with a deformable shape. This abstraction is motivated by potentially being more efficient for modeling large crowds than a direct representation; while being a more accurate abstraction of the flock, more amenable to control than simpler representations such as spheres or central moments; and being less sensitive to random perturbations in the behavior of individual flock members than previous methods. We indeed found this method to be more scalable—allowing for the shepherding of hundreds of agents using only a few shepherds—and surprisingly robust against compared to previous approaches, against increasing uncertainty or lack of coherence in the motion of the flock members. An example of the blob model and a screenshot of a simulation involving many agents is shown in Figure 2. In general, this work highlighted the usefulness of geometric abstractions in
improving the robustness and scalability of existing methods.

In general, we found that manipulation in the face of uncertainty benefited from use of a hybrid planners and geometric abstractions. However, the preliminary results from these shepherding experiments were not necessarily consistent across a wide range of environments, and it is not clear exactly what geometric features of the environments make the difference. Furthermore, these techniques rely heavily on a good local planner to be effective. Furthermore, meta-graph techniques require much more sampling than the tree-based motion planners, making it inefficient to perform such planning in large scale simulations. Therefore, more work is still needed to improve the efficiency and consistent performance of these techniques.

3 Tracking

In shepherding, we always assumed that the shepherd knows the positions of all of the flock members. However, in reality, this may not be the case. The environment may occlude some or all of the flock from view of its sensors. Therefore, we were motivated to consider visibility. That is, in [36, 38, 39, 37] we explored the problem of moving a mobile sensor to search for and track groups of targets with unpredictable trajectories as they move through environments filled with obstacles.

This is a variant of the pursuit-evasion problem, where the goal is to maintain visibility of as many of
the wandering agents as possible. Here, we developed and compared several techniques for tracking the
the targets through the environment: we called those methods Reactive, IO, VAR, and MTR. The baseline
Reactive approach is simple: attempt to follow the targets when they are in view. However, we found that it is
easy for for the mobile sensor to lose track of the targets if the targets quickly turn a corner or if the sensor
moves more slowly than the targets. Therefore, the next approach, IO, used sampling to predict candidate
locations for the targets and sensor, and then select the sensor movement that maximizes its coverage of the
predicted target locations. We also developed a method called VAR (loosely based on the method in [27]),
which partitioned the free space into a set of overlapping hyperspheres. This network of overlapping spheres
form a roadmap. We use offline sampling to compute a visibility score between each pair of spheres, and this
score is stored in the edges of the roadmap. Then, in real time, it is used to make quick decisions about where
the sensor should move to optimize coverage of potential visibility risks. Finally, we developed an approach
called MTR, which partitioned the free space into a set of tunnel-like regions, which we called monotonic
tracking regions (MTRs). These regions satisfy the property that the sensor can monotonically maintain the
visibility of a target in the region by moving forwards or backwards along a trajectory that supports the tunnel.
This offline computation of the MTRs and their supporting trajectory allows the online problem of camera
following to be reduced to a fast linear programming problem.

Figure 3: A camera (shown in blue with motion trail) tracking a group of agents (shown in red). Figure 3a
shows the results using a simple reactive tracker. The reactive tracker simply attempts to follow the flock,
it does not attempt to predict occlusion or maximize the overall visibility of members. Figure 3b shows the
results of tracking using the IO algorithm. By sampling, the camera position is optimized at each time step to
maximize visibility of the flock.
3.1 Collaborative Foraging and Stigmergy

In [16], we looked at a different kind of planning problem—the collaborative foraging problem, wherein the goal is for a set of robots to collaboratively to find food in one location and deliver it to another location. The approach we used mimics the stigmergic use of pheromone trails by ants, but with a twist: instead of storing the pheromones continuously in the environment or on a dense grid, the stigmergic information is stored in a sparse set of beacons distributed through the space. These beacons can be placed by robots, moved around freely, and their value can be updated by the robots continuously. An illustration of the approach appears in Figure 4. The approach is stigmergic, meaning that the agents store information in the environment and act only locally. This feature allows the approach to be fully distributed, and also allows for quick response to dynamic changes in the environment despite using only local information. While the model we used explicitly placed physical beacons in the environment, these desirable properties also lend themselves to methods involving virtual trails of beacons. We intend to harness this approach with the meta-graph so that meta-nodes are represented as virtual beacons. Using similar update rules, it may be possible to continuously optimize paths and account for dynamic obstacles by relocating and updating simple stigmergic information in each meta-node.
4 Proposed Work In Detail

In my preliminary work, I have already examined several ways to achieve better robustness in the face of manipulation uncertainty. First, in the shepherding planning scenario, I found that through geometric abstractions such as meta-nodes, deformable shapes, and space partitioning, it was possible to make more intelligent decisions and produce coherent plans despite significant uncertainty in the pose of agents. Second, in the tracking scenario, we found ways to use offline computation on the geometry of the environment to greatly increase the effectiveness of online planning decisions, despite uncertainty about the future trajectory of agents. However, there are still many open problems and concerns with this work. I will address them in the following sections.

4.1 Robust Tracking With Visibility and Environmental Uncertainties in 3D

To manipulate an object, it is important to be able to track its position. In [36, 38, 39, 37], we had experimented with different methods to solve a group tracking problem. Our two most successful techniques, VAR and MTR, involved partitioning the free space of the environment into topologically connected regions which provide our planner a view of both mobility in the space as well as the visibility characteristics of its regions. However, there is much room for improvement. In my proposed work, I seek to create a space partitioning method that can be performed on 3D environments, and is less sensitive to small discontinuities or noise in the environment model. Another limitation of the basic tracking problem that I would like to consider is that existing methods require that the mobile sensor maintains visibility of its target at all times. In reality, if the environment model is partially known, it may be possible to localize a moving target without maintaining constant visibility. For example, If a target is observed entering a partition of the space which is known to have only a single exit, it is not necessary to follow the target into the partition to localize the target. Some examples of environments which illustrate these types of situations are shown in Figure 5. To achieve this, I propose to use the above space partitioning method to discover areas that represent loss-of-localization risks and create paths that optimize the probability of future visibility or rendezvous with the target.

For the basic tracking problem, we have used a method named VAR, which was motivated by the visibility-aware roadmap concept [27]. The roadmap in VAR is constructed based on partitioning the free space in to a network of overlapping hyperspheres, and computing the visibility between those spheres. In VAR, sphere centers are selected by picking free configurations from a fixed grid, and the maximum radius of the spheres
Figure 5: Examples of tracking without constant visibility. The sensor is depicted by a green circle, the target is depicted by a yellow star. The free space is the white area, and the obstacles are dark gray. In Figure 5a, the target must be either in an aisle, or behind an obstacle. Determining which part of the space the target is in is only a matter of walking back and forth along the bottom of the aisle. In Figure 5b, it is not necessary for a target to enter the room; the room is ‘covered’ simply by waiting outside of it. In Figure 5c, the target is obstructed by the column. However, the camera still covers the area adequately.

generated from those configurations is given as input by the user. We found that this method has several limitations. First, to ensure coverage of narrow passages of the space, it is required to have a fine resolution grid, which greatly increases the number of partitions and the sampling required to compute the weights of the roadmap. Furthermore, it is important that each sphere in the roadmap accurately represents points of the space that have similar visibility properties. By sampling on a rigid grid, it is common to end up with spheres containing points with very different visibility characteristics. However, this method is amenable to improvement—the partitioning is easy to compute in 3D, and the visibility-graph representation that it produces is quite useful for more advanced tracking objectives.

Therefore, to address the issues in VAR, we propose to compute a visibility-aware roadmap based on a combination of sampling and clustering. That is, each node in the roadmap represents a set of points that have similar visibility. This is done by first sampling \( n \) points \( S \) from the 3D space. For each point \( p \in S \), we determine two types of connections: a visibility connection \( v(p, q) \) if two points \( p \) and \( q \) are visible from each other, and a neighborhood connection \( n(p, q) \) if two points \( p \) and \( q \) are neighbors.

We then cluster the points. To do so, we must first define a relevant metric for clustering. Here, we define the visibility integrity, which measures how accurately a set of points are indeed visible by a cluster of points. We compute and update visibility integrity to estimate the loss of visibility that occurs after merging two clusters. Let \( C \) be a cluster of points. If \( C \) contains only a single point \( p \), its visibility \( V(C) \) is simply all points visible from \( p \). If \( C \) contains multiple points, then \( V(C) \) is the union of all points visible from the points in \( C \),
i.e., $V(C) = \bigcup_{p \in C} V(p)$. For each point $q$ in $V(C)$, we measure the visibility integrity of $q$ with respect to $C$. We denote this measure as a function $vi(q, C)$ that maps a point $q$ and a cluster $C$ to a scalar value between 0 and 1. When $vi(q, C)$ is 1, then all points in $C$ can see $q$. Likewise, if $vi(q, C)$ is 0, $q$ is invisible from any point in $C$. Then, we define the visibility integrity of $C$ as the average of visibility integrity of all points in $V(C)$, i.e.,

$$vi(C) = \frac{\sum_{p \in V(C)} vi(q, C)}{|V(C)|}.$$ 

Intuitively, when visibility integrity is low, that means there are regions that are visible by some points in the cluster but are not visible from most of the points. Therefore, a cluster with high visibility integrity is more desirable. Note that the visibility integrity is always 1 when $C$ contains only one point.

Then, for each neighborhood connection $e = \{U, V\}$, we assign a weight to $e$ by computing the visibility integrity $vi(U \cup V)$. We cluster points based on the order of their visibility integrity. Our approach maintains neighborhood connections in a max-heap and iteratively collapses neighborhood connections with the highest visibility integrity. This process repeats until the highest visibility integrity is lower than a user specified value. We believe that this value is intuitive enough to be specified by a naïve user.

Each point $s$ in the original sample $S$ now has the information about the clusters that are likely to be visible from $s$. Moreover, the cluster that has the highest visibility integrity to $s$ is usually the cluster that contains $s$. When the camera is tracking a single target $t$, our planner simply finds the $k$ closest points around $t$ and identifies the clusters that are commonly associated with these $k$ points. The camera then determines its next target configurations by selecting a cluster and a configuration in the cluster which minimizes the travel distance and maximizes the visibility integrity. The motion of the camera is simply determined by a smooth and collision free path that brings the camera to the target. When the camera is tracking multiple targets $T$, the planner will select $K$ points that are closest to $T$. From these $K$ points in the roadmap, the camera decides its next position based on the same strategy.

One way to make such decisions is to use the computed visibility meta-graph computed from the clustering approach to estimate which parts of the space represent risks not just in loss-of-visibility but also loss-of-localization. This information permits our camera sensor to localize objects in known environments without maintaining constant visibility, and potentially reduce the amount of movement necessary by the sensor to keep track of a target. These abilities are important for robustness because even if the visibility of the target is lost (leading to pose uncertainty). It ensures that sensor still has retained enough information to make decisions about where the mobile sensor can move, and optimize the probability of rendezvous with the target and predict the likelihood of escape. The idea of finding areas of the space that represent visibility risks is
similar to the idea of finding *relocalization zones*, a concept used in [28] to plan paths that reduce localization uncertainty by maximizing visibility of important landmarks in the environment.

### 4.2 Reusable Kinodynamic Manipulation Planning Under Uncertainty

Previously, in [14] and [35], we experimented lightly with meta-graph techniques to improve the robustness of shepherding, a very difficult manipulation problem. To improve our understanding of how to harness the meta-graph approach, I propose to study how to apply this technique to the more general manipulation problem of posing objects in the plane through push interactions. The most robust approaches to this problem usually involve grasping [22, 30, 4], caging, using special manipulators [2], or require the design of rigorous analytical models for each new object [6]. Instead, I seek ways to perform this manipulation using **nonprehensile manipulation** (only pushing), considering dynamics, and through sampling-based methods rather than of analytical models.

Furthermore, current planners for solving problems with kinodynamic constraints are usually tree-based planners (e.g., RRT [24], EST [17], PDST [33], and DSLX [29]). The general strategy behind such approaches is to grow an exploring tree rooted at an initial state until one of the goal states is (approximately) reached. However, if the initial or an intermediate state is inaccurate, or shifted a bit (e.g., due to various kinds of uncertainty and simulation and model inaccuracies), then the plan usually becomes invalid and a new tree will be built from scratch. Unlike these approaches, I seek to create an approach that is able to **reuse prior computation** to enable fast replanning. In addition, I hope to be able to reuse this data in tracking problem (Section 4.1) so that **tracking and manipulation can be performed simultaneously** (such as in the Simultaneous Localization and Grasping work in [30]) using the same data.

To achieve robust planning for this scenario, I propose to use a graph-based planner based on the meta-graph approach we have used in [35]. This kind of planner generates a reusable uncertainty roadmap that can be used to perform multiple queries, for both short and long horizon planning. The meta-graph approach prescribes encoding the configuration of the target object in a fuzzy manner—that is, instead of a single configuration, each node of the roadmap acts as a *metanode*, representing a set of configurations.

We have begun some preliminary research to determine the feasibility of the approach. More specifically, our instance of the polygon pushing problem poses the question of how to plan the motion for a cylindrical robot \( R \) to move a polygonal target object \( P \) from a given initial pose to a given goal pose using only push interactions. The workspace may be filled with known obstacles that the robot and the object are permitted to touch, but not penetrate. We assume that \( R \) is a holonomic cylinder, and that the \( P \) that is an extruded
polygonal shape with known geometry. We also assume that a black-box kinodynamic simulator is provided, containing the obstacles, the robot, and the object. The simulation provides the resulting trajectories of $P$ and $R$ given control inputs for $R$.

A configuration of the robot $R$ can be represented as point $(x, y) \in \mathbb{R}^2$. Likewise, a configuration for $P$ is represented as a point $(x, y, \theta) \in \mathbb{R}^2 \times S^1$, and the set of all such points forms the configuration space $C$ of the polygonal target object. As such, the set of all configurations where the object is colliding with an obstacle is referred to as $C_{\text{obst}}$, and the free space is referred to as $C_{\text{free}} = C - C_{\text{obst}}$.

Our method consists of several phases. First, we create a local planning function $LP$ that given a starting configuration for $P$ ($p_{\text{start}}$), and a desired destination configuration for $P$ ($p_{\text{dest}}$), outputs a push manipulation that supports moving $P$ from $P_{\text{start}}$ to $P_{\text{dest}}$. To do so, we sample information about how $P$ will move when it is pushed by $R$ from various angles. We store this information as points in a KD-tree so that when we are given the inputs $p_{\text{start}}$ and $p_{\text{dest}}$, we can quickly look up the closest sampled push manipulation that supports pushing $P$ to $p_{\text{dest}}$. Furthermore, for each manipulation, we also sample many times with random perturbations, to simulate uncertainty that may arise from imperfectly executing pushes. This information will be used later in roadmap generation to compute probabilities.

Next, we build the roadmap using this data. In general, we first seek to generate metanodes which capture sets of states rather than single states, and then we connect them with edges whose weights reflect the probability of successful transit from one metanode to the next. We begin by sampling from the free space. Many sampling techniques can be used; we use the sampling technique from Gaussian PRM [25], mixed with uniform sampling. We convert each configuration $s$ from the set of samples into a metanode by building a sphere, centered at $s$, with pre-defined radius. Configurations that lie within the sphere defined for a configuration $s$ are thus considered to be conforming to the metanode for $s$. An example environment with sampled metanodes is shown in Figure 6a. This sampling process may lead to many overlapping metanodes, so we also attempt reduce the number of metanodes in the roadmap by removing metanodes that have significant overlap.

Finally, to complete the roadmap, we connect meta-nodes that are in close proximity. The edge between two nodes, $s_a$ and $s_b$ is given a weight that is computed based on sampling the number of feasible, successful pushes between uniformly random configurations conforming to $s_a$, and any configuration conforming to $s_b$. The number of successful pushes versus the number of attempted pushes at an edge provides us with a metric that can be used to estimate the probability of successfully pushing an object from some state conforming to node $s_a$ to some state conforming to $s_b$, in the presence of the obstacles. Other metrics can be used, such as
Figure 6: An example polygon-pushing scene. The robot is depicted as a red cylinder, and the target object is the gray bar. In 6a, white circles depict the meta-nodes sampled from the configuration space after overlap reduction is performed. In 6b, white lines depict the connections in the roadmap between metanodes. In 6c, a safe corridor is computed for the target object to move inside the room. The corridor is depicted by a series of connected circles (metanodes).

the average probability of success. Figure 6b shows our example environment with a constructed roadmap.

Given the weighted roadmap, we compute an all-pairs shortest path. This information is stored so that upon any query, a sequence of meta-nodes can be quickly obtained that has the greatest probability of successful manipulation. We call this sequence of meta-nodes a corridor, representing many possible paths for pushing $P$. Figure 6c shows an example corridor extracted from a query in the example scene. This corridor represents a robust path and can be used online quickly and efficiently to generate a final trajectory. Since the corridor is a significantly reduced subset of the free space, it can be used to focus search in another simple planning algorithm such as A-star, PRM, or RRT. Another interesting possibility is to apply the previous foraging work mentioned in Section 3.1 to continuously improve the meta-graph. Metanodes in the graph take the place of beacons, and can be continually re-arranged and assigned updated fitness as new paths through the space are sought and evaluated.

References


