Real-Time Adaptive Motion Planning (RAMP) of Mobile Manipulators in Dynamic Environments With Unforeseen Changes

John Vannoy and Jing Xiao, *Senior Member, IEEE*

***Abstract*—This paper introduces a novel and general real-time adaptive motion planning (RAMP) approach suitable for planning trajectories of high-DOF or redundant robots, such as mobile manipulators, in dynamic environments with moving obstacles of unknown trajectories. The RAMP approach enables simultaneous path and trajectory planning and simultaneous planning and execution of motion in real time. It facilitates real-time optimization of trajectories under various optimization criteria, such as minimizing energy and time and maximizing manipulability. It also accommodates partially specified task goals of robots easily. The approach exploits redundancy in redundant robots (such as locomotion versus manipulation in a mobile manipulator) through loose coupling of robot configuration variables to best achieve obstacle avoidance and optimization objectives. The RAMP approach has been implemented and tested in simulation over a diverse set of task environments, including environments with multiple mobile manipulators. The results (and also the accompanying video) show that the RAMP planner, with its high efficiency and flexibility, not only handles a single mobile manipulator well in dynamic environments with various obstacles of unknown motions in addition to static obstacles, but can also readily and effectively plan motions for each mobile manipulator in an environment shared by multiple mobile manipulators and other moving obstacles.**

***Index Terms*—Adaptive, dynamic obstacles of unknown motion, loose coupling, mobile manipulators, partially specified goal, real time, redundant robots, trajectory optimization.**

I. Introduction

MOTION PLANNING is a fundamental problem in robotics [1], [2] concerned with devising a desirable motion for a robot to reach a goal. Motion planning for high-DOF articulated manipulators or mobile manipulators is more challenging than for mobile robots because the high-dimensional configuration space of a robot has little or no resemblance to the physical space that the robot works in, and how to construct a configuration space higher than three dimensions efficiently remains a largely unsolved problem.

A. Related Research on Motion Planning

Randomized algorithms, such as the popular probabilistic roadmap (PRM) method [3] and rapidly exploring random tree (RRT) method [4], are found to be very effective in finding a collision-free path for a robot with high DOFs offline because such algorithms avoid building the robot's configuration space explicitly by sampling the configuration space. The PRM method has inspired considerable work on improving sampling and roadmap construction [2], including a recent paper [5] on producing compact roadmaps to better capture the different homotopic path groups. By building a tree rather than a graph, the RRT method is more suitable for generating a path in one shot or generating a trajectory directly and thus more suitable for online operation [6]. Both methods have seen many variants [2].

There are also methods for path planning based on genetic algorithms (GAs), or more broadly, evolutionary computation [7], [8], which are general frameworks of randomized search subject to user-defined optimization criteria. Such optimization techniques have been used widely and successfully in many application domains [8], [9] to tackle NP-hard optimization problems. There are two major ways of applications. One straightforward way is to map a problem into the form suitable for a standard, off-the-shelf GA, solve it by running the GA, and then, map the results back to the application domain. This one-size-fit-all approach is often not effective because it forces artificial transformation of a problem into something else that is confined in the format of a standard GA but may lose certain important nature of the original problem. Some GA-based path planning methods [10], [11] adopt such an approach, where C-space is discretized into a grid, and a path is in terms of a fixed-length sequence of grid points. As the standard GA operates on fixed-length bit strings, search is often very slow.

A more effective approach is to adopt the general idea of evolutionary computation to solve a problem in a more natural and suitable representation. The path planning methods reported in [12]–[14] belong to such a customized approach. A real-time path planning method is reported in [12] for 2 DOF point mobile robots, which is extended in [13] for 3 DOF point flying robots with specific constraints. A multiresolution path representation is proposed in [14] for path planning. However, all evolutionary algorithms have a number of parameters that must be set appropriately, which is often not a trivial task.

Manuscript received May 16, 2007; revised December 13, 2007 and March 5, 2008. First published October 10, 2008; current version published October 31, 2008. This paper was recommended for publication by Associate Editor K. Yamane and Editor L. Parker upon evaluation of the reviewers' comments. A preliminary part of this paper was presented at the IEEE International Conference on Intelligent Robots and Systems, Sendai, Japan, 2004.

The authors are with the Intelligent, Multimedia and Interactive Systems (IMI) Laboratory, Department of Computer Science, University of North Carolina at Charlotte, Charlotte, NC 28223 USA (e-mail: jmvannoy@gmail.com; xiao@uncc.edu).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors: a video showing the real-time planning and execution of mobile manipulator motion by our RAMP algorithm. This video is 14 MB in size.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2008.2003277

Unlike path planning, motion planning has to produce an executable trajectory for a robot in configuration×time space, or *CT-space*, and not merely a geometrical path. A common approach is to conduct trajectory planning on the basis of a path generated by a path planner. A notable framework is the elastic strip method [15], which can deform a trajectory for a robot locally to avoid moving obstacles inside a collision-free "tunnel" that connects the initial and goal locations of the robot in a 3-D workspace. Such a "tunnel" is generated from a decomposition-based path planning strategy [16]. The other approach is to conduct path and trajectory planning simultaneously. However, most effort in this category is focused on offline algorithms assuming that the environment is completely known beforehand, i.e., static objects are known, and moving objects are known with known trajectories [17]–[20]. As for dealing with unknown moving obstacles, only recently some methods were introduced for mobile robots [21], [22].

The combination of mobility and manipulation capability makes a mobile manipulator applicable to a much wider range of tasks than a fixed-base manipulator or a mobile robot. For a mobile manipulator, a task goal state is often partially specified as either a configuration of the end-effector, which we call a *place-to-place* task, or a desired path (or trajectory) of the end-effector, which we call a *contour-following* task, and the target location/path of the base is often unspecified.

Here, a major issue of motion planning is the coordination of the mobile base and the manipulator. This issue, as it involves redundancy resolution, presents both challenges and opportunities. There exists a rich literature addressing this issue from many aspects. Some researchers treat the manipulator and the mobile base together as a redundant robot in planning its path for place-to-place tasks [23]–[25]. Some focused on planning a sequence of "commutation configurations" for the mobile base when the robot was to perform a sequence of tasks [26], [27] subject to various constraints and optimization criteria. Others focused on coordinating the control of the mobile base and the manipulator in a contour-following task [28], [29] by trying to position the mobile base to maximize manipulability. Many considered nonholonomic constraints.

While most of the existing work assumes known environments with known obstacles for a mobile manipulator, a few researchers considered local collision avoidance of unknown, moving obstacles online. One method [30] used RRT as a local planner to update a roadmap originally generated by PRM to deal with moving obstacles. For contour-following tasks, an efficient method [31] allows the base to adjust its path to avoid a moving obstacle if possible while keeping the end-effector following a contour, such as a straight line. Another method [29] allowed the base to pause in order to let an unexpected obstacle pass while the arm continued its contour-following motion under an event-based control scheme. Other methods include one based on potential field [32] to avoid unknown obstacles and one based on a neuro-fuzzy controller [33] to modify the base motion locally to avoid a moving obstacle stably. There is also an online planner for the special purpose of planning the motions of two robot arms getting parts from a conveyer belt [34].

However, we are not aware of any existing work that can plan motions of high-DOF robots globally among *many* unknown dynamic obstacles.

B. Our Problem and Approach

Planning high-DOF robot motion in such an environment of many unknown dynamic obstacles poses special challenges. First, planning has to be done in real time, cannot be done offline, and cannot be based on a certain prebuilt map because the environment is constantly changing in unforeseen ways, i.e., the configuration space obstacles are unknown and changing. Examples of such environments include a large public square full of people moving in different ways, a warehouse full of busy-moving robots and human workers, and so on. Such an environment is very different from static or largely static environments or known dynamic environments (i.e., with other object trajectories known), where motion planning can reasonably rely on exploring C-space (for known static environments) or CT-space (for known dynamic environments) offline (such as by PRM). The elastic strip method provides the flexibility to make small adjustments of a robot motion to avoid unknown motions of obstacles, if the underlying topology of the C-space does not change. For an environment with changing C-space topology in unknown ways, a planned path/trajectory can be invalidated completely at any time, and thus, real-time adaptive *global* planning capability is required for making drastic changes of robot motion. Planning and execution of motion should be simultaneous and based on sensing so that planning has to be very fast and always able to adapt to changes of the environment.

By nature, to tackle motion planning in an unknown dynamic environment cannot result in a complete planning algorithm. That is, no algorithm can guarantee success in such an unknown environment. We can only strive for a *rational* algorithm that serves as the "best driver" of a high-DOF robot, but even the best driver cannot guarantee to be accident-free if other things in the environment are not under his/her control.

This paper addresses the problem of real-time simultaneous path and trajectory planning of high-DOF robots, such as mobile manipulators, performing general place-to-place tasks in a dynamic environment with obstacle motions unknown. The obstacle motions can obstruct either the base or the arm or both of a mobile manipulator. We introduce a unique and general real-time adaptive motion planning (RAMP) approach. Our RAMP approach is built upon both the idea of randomized planning and that of the *anytime*, *parallel*, and *optimized* planning of evolutionary computation, while avoiding the drawbacks. The result is a unique and original approach effective for the concerned problem.

The RAMP approach has the following characteristics.

1) Whole trajectories are represented at once in CT-space and constantly improved during simultaneous planning and execution, unlike algorithms that build a path/trajectory sequentially (or incrementally) so that a whole path/trajectory can become available only at the end of the planning process. Our anytime planner can provide a valid trajectory quickly and continue to produce better

trajectories at any later time to suit the need of real-time *global* planning.

2) Different optimization criteria (such as minimizing energy and time and optimizing manipulability) can be accommodated flexibly and easily in a seamless fashion. Optimization is done directly in the original, continuous CT-space rather than being confined to a certain limited graph or roadmap. Trajectories are planned and optimized directly rather than conditional to the results of path planning.
3) Our planner is intrinsically parallel with multiple diverse trajectories present all the time to allow instant, and if necessary, drastic adjustment of robot motion to adapt to newly sensed changes in the environment. This is different from planners capable of only local trajectory adjustment based on a known set of homotopic paths. It is also different from sequential planners, such as anytime A* search [35], which also requires building a discrete state–space for search—a limitation that our planner does not have.
4) Trajectory search and evaluation (of its optimality) are constantly adaptive to changes but built upon the results of previous search (i.e., knowledge accumulated) to be efficient for real-time processing.
5) As planning and execution (i.e., robot motion following the planned result so far) are simultaneous, partially feasible trajectories are allowed, and the robot may follow the feasible part of such a trajectory (if it is the current best) and switch to a better trajectory to avoid the infeasible part.
6) With multiple trajectories from our planner, each trajectory can end at a different goal location in a goal region, i.e., partially specified goals, rather than a single goal configuration.
7) Our planner represents a trajectory for a redundant robot, such as a mobile manipulator, as *loosely coupled* trajectories of redundant variables to take advantage of the redundancy in order to best achieve obstacle avoidance and various optimization objectives.

The rest of the paper is organized as follows. Section II provides an overview of our RAMP approach; Sections III and IV describe problem representation and initialization; Section V outlines our optimization criteria for trajectory evaluation and describes the strategies for evaluation. Sections VI and VII describe the strategies to alter trajectories to produce better ones. Section VIII describes how the RAMP planner can create and preserve a diverse set of trajectories. Section IX provides implementation and experimentation results and discusses performance of the planner. Section X concludes the paper.

II. Overview of the RAMP Approach

One basic premise of our approach is that the planning process and the execution of motion are interweaving to enable simultaneous robot motion planning and execution. This is achieved through our anytime planning algorithm that always maintains a set of complete trajectories in the CT-space of the robot called a *population*. The feasibility and optimality of each trajectory, called *fitness*, is evaluated through an *evaluation function* coding the optimization criteria. Feasibility refers to collision-free and singularity-free. Both infeasible and feasible trajectories are allowed in a population. Feasible trajectories are considered fitter than infeasible trajectories. Within each type, trajectories are compared for optimality in fitness.

The initial population is a combination of randomly generated and deliberately seeded trajectories. Deliberately seeded trajectories include ones constructed to represent distinct subpopulations in order to achieve certain diversity in the population. If the environment contains known static obstacles, trajectories based on preplanned feasible paths with respect to the known static obstacles can also be included. See Section IV for more details.

Once the initial population is formed, it is then improved to a fitter population through iterations of improvements, called *generations*. In each generation, a trajectory is randomly selected and altered by a randomly selected modification operator among a number of different modification operators, and the resulting trajectory may be used to replace a trajectory that is not the fittest to form a new generation. The fittest trajectory is always kept in the population and can only improve from generation to generation. Each generation is also called a *planning cycle*.

To improve the fitness of the initial population, a number of initial planning cycles may be run based on the initial sensing information of the environment before the robot begins executing the fittest trajectory. The robot need not wait for a feasible trajectory to emerge; if no feasible trajectory is available, the robot will begin moving along the fittest in feasible trajectory while continuing the search for a fitter, and hopefully will locate a feasible trajectory before it comes within a distance threshold D of the first predicted collision or singularity of the executed trajectory. This strategy makes sense because: 1) the presently predicted infeasible trajectory may become feasible later and *vice versa*; 2) as to be described later, our planner makes the robot switch to a better trajectory if one is available, and thus, before the infeasible part of the currently followed trajectory is encountered, the robot may already switch to a better trajectory; 3) the strategy allows limited sensing, in which the robot may not sense an obstacle until getting closer; and 4) it provides a measure of safety in trajectory evaluation (see Section V).

As the robot moves, planning continues to improve the population of trajectories until the next *control cycle*, when the robot can switch to a fitter trajectory so that it always follows the best trajectory. For that purpose, each trajectory is always updated to start from the current robot configuration with the current velocity when a new control cycle begins. For the trajectory that is being followed, this means that the executed portion of the trajectory is dropped from the trajectory, while for every other trajectory, it means that only the starting configuration and velocity are changed—the rest of the knot points on the trajectory (see Section III) remain intact. Note that each control cycle here does not necessarily have to be a servo cycle of the low-level controller. Our control cycle, which is high level for controlling the rate of adaptation, can be longer than a servo cycle to ensure that within a control cycle, there can be more than one planning cycle. This is because adaptation is guided by planning.

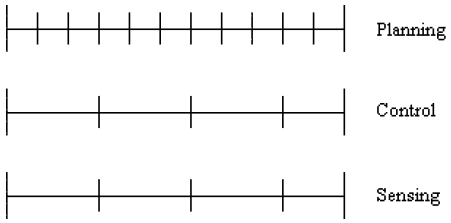

Fig. 1. Relationship among planning, control, and sensing cycles.

Changes in a dynamic environment are sensed and fed to the planner in each sensing cycle, which lead to updated fitness values of trajectories in the subsequent planning cycles, and unknown motions of moving obstacles are predicted in fitness evaluation of robot trajectories. The presence of a diverse population of ever-improving trajectories enables the robot to quickly adapt to changes in the environment. It does so by following the fittest trajectory under each circumstance: when the current trajectory that the robot follows becomes worse or can no longer be followed due to imminent collision (i.e., the threshold D is reached), the robot may not need to stop its motion and replan from scratch; rather the planner often merely needs to switch the robot to a feasible or better trajectory in the population swiftly in a seamless fashion. The chosen trajectory can be of a very different homotopic group from the previous one to deal with drastic and large changes.

In the case when the robot reaches D of the current trajectory but finds no better trajectory to switch to, it will stop its motion at D, which is called a *forced stop*. However, the RAMP planner (i.e., the robot's "thinking" process) never stops, and it continues to plan and search for a better trajectory for the robot. The robot resumes its motion once a better trajectory is found.

Such planning/control/sensing cycles continue to interact and move the robot toward a goal configuration in the best possible way in real time: *improving* the trajectories it follows if there is no change in the environment, or both *adapting and improving* the trajectories if there is a sensed change. Fig. 1 illustrates a possible relationship among planning, control, and sensing cycles (note that the planning cycles actually vary in length).

The RAMP algorithm is outlined as Algorithm 1.

Unlike an evolutionary algorithm, we use random selection and random modification operators that cannot be called "mutation" operators because they introduce drastic rather than small changes to create a diverse population of trajectories ready to adapt to changing environments. Our RAMP algorithm further maintains diversity and prevents homogeneity in a population of trajectories by creating and preserving distinct subpopulations of trajectories as explained in detail in Section VIII. Moreover, the RAMP algorithm does not need tuning probabilities as well as most other parameters that many evolutionary algorithms do. As the result, it is easy to implement and is robust to different task environments. In fact, our algorithm only needs to decide the parameter *population size*, but the value can be invariant or rather insensitive to many different environments, as will be described later in Section VIII.

The fitness evaluation procedure of RAMP is also original, incorporating multiple criteria that are often not considered in many other motion planning algorithms, and not only feasible but also infeasible trajectories are evaluated.

Algorithm 1: RAMP

```
t ⇐ 0;
initialize population P(t) of unique trajectories;
evaluate P(t) and obtain the current best trajec. τ;
τ_move ⇐ τ;
create subpopulations in P(t);
while goal is not reached do
    simultaneously do (1) and (2):
    (1)
    t ⇐ t + 1;
    call modification();
    if end of current control cycle then
        update starting configuration & velocity of P(t);
        create subpopulations in P(t);
        evaluate P(t) and obtain the best τ;
        τ_move ⇐ τ;
    if new sensing cycle then
        evaluate P(t) and obtain the best τ;

    (2)
    if D is not reached then
        move the robot along τ_move;
    else
        stop the motion; /**a forced stop**/
```

Procedure `modification`

```
randomly select operator o_i;
randomly select 1 or 2 trajectories from P(t);
apply o_i to the selected trajectories to produce 1 or 2
unique new trajectories;
evaluate new trajectories;
if new trajectory better than worst trajectory then
    if new trajectory is feasible then
        randomly replace a member of P(t) which is not
        the best in a (sub)population;
    else
        randomly replace an infeasible member of P(t)
        which is not the best in a (sub)population;
return P(t) and the best member of P(t) as τ;
```

Our RAMP approach also supports the partial specification of a goal: only the end-effector position and orientation with respect to the world coordinate system are needed. Different trajectories may hold different goal base configurations and arm configurations (that achieve the same end-effector goal) in the case of mobile manipulators so that redundancy is exploited to achieve flexibility amid environments with dynamic changes.

The details of the RAMP algorithm are presented in the sections next.

III. Trajectory Representation

We represent a trajectory of a mobile manipulator uniquely as a pair of loosely coupled manipulator and base subtrajectories with the following characteristics.

1) For the manipulator subsystem, a path of knot configurations is specified in the joint space, based on which a cubic-splined trajectory is used.
2) For the base subsystem, a path of knot configurations is specified in the Cartesian space of the world coordinate

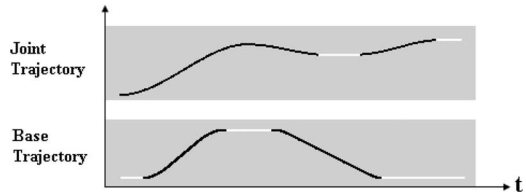

Fig. 2. Loose coupling of locomotion and manipulation.

system, based on which a linear-with-parabolic-blends trajectory is used.

3) Either trajectory as a function of time may consist of variable segments of constant values, i.e., intervals of no movement, and these time intervals may or may not overlap between the two trajectories. This is the key to achieve loosely coupled behavior: either subsystem can move while the other does not,[1] or they can move at the same time.
4) Two subsystem trajectories are aligned in time to form a trajectory that uniquely determines the motion of the whole system.

Fig. 2 illustrates this notion with one joint trajectory and one dimension of the base trajectory.

The main advantage of loose coupling for place-to-place tasks is its flexibility in dealing with uncertain dynamic environments: when a dynamic obstacle shows up, sometimes an agile arm movement makes a better avoidance motion than a heavy base movement; sometimes it is more efficient to vary the motion of the base rather than having a large motion of the arm (especially if a large payload is held by the end-effector); sometimes both need to move to make an avoidance; and sometimes it is more energy and even time efficient to just stop the entire system for some period to let the obstacle pass. Our RAMP planner of loosely coupled motion allows all the previous varieties to happen based on the circumstance.

A trajectory leads the mobile manipulator from its *current* configuration (with certain velocity and acceleration) to one of the goal configurations. Each subtrajectory may consist of an arbitrary number of segments, separated by knot configurations, also called *knot points*. The data structure for each segment contains the bounding knot points of the segment, the information of feasibility and fitness of the segment (see Section V), and the desired velocities and accelerations at the bounding knot points if the segment is feasible.

Each nonconstant trajectory segment is generated using the minimum time for the slowest joint (including the base) of the robot to move from the starting knot point to the ending knot point of the segment, taking into account constraints on speed and acceleration. For the arm, the segment for each joint is a cubic polynomial of time, and for the base, the segment is a linear function with parabolic blends. Specifically, the minimum execution time T_{ij} of each joint i for each path segment j is first calculated based on the cubic trajectory, under the maximum acceleration and maximum speed constraints of joint i. For the same path segment j, the minimum time for the base $i = 0$ is also calculated based on a linear trajectory with parabolic blends under its maximum acceleration and velocity constraints. Next, the maximum of T_{ij} among all joints (including the base), $T_{\max,j}$, is used as the time needed to complete the path segment j. Based on $T_{\max,j}$, the corresponding trajectory segment (for the arm and for the base) can be generated.

[1]Note that when we say "the base moves while the arm does not move," we mean that the arm has no relative motion to the base.

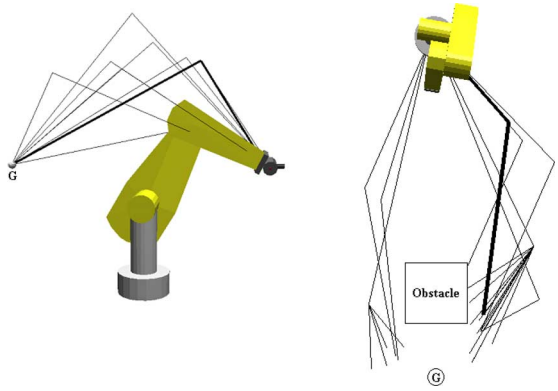

Fig. 3. Illustration of arm and base trajectory populations with gripper goal at "G."

Note that more sophisticated methods taking into account dynamics and torque constraints [36], [37] can be used to determine minimum-time trajectories. However, here we need to compromise computation for true minimum-time trajectory for real-time performance in planning.

Fig. 3 illustrates a robot and its arm and base trajectory populations, respectively. Each trajectory is indicated by line segments connecting wrist position or base position at each knot point. Note that the lines themselves are simply to show the order in which the knot points are visited in each path and certainly not the actual paths. The heavy line in each figure indicates the path with the highest fitness for its trajectory.

IV. Initialization

An initial population is a combination of randomly generated and deliberately seeded trajectories. The RAMP algorithm generates random trajectories made up of randomly chosen knot configurations as uniform samples within the joint limits of the manipulator and the workspace boundaries of the base. The initial base and arm configuration, respectively, determine the starting point of each. An ending base configuration is randomly generated within a reasonable neighborhood of the goal gripper location so that the gripper goal can be reached. Inverse kinematics is used to find a corresponding ending arm configuration in joint space. A random number of intermediate knot points are inserted into the base's trajectory and the arm's trajectory. Each knot point is a randomly sampled configuration. Once the knot points are created, the trajectory can be computed (see previous section). The other information about each trajectory segment in the segment data structure is determined through fitness evaluation of the trajectory.

The RAMP algorithm can also deliberately seed the initial population with trajectories built based on different "departing directions" to provide diversity (more on this in Section VIII).

Algorithm 3: Offline Planner

$i \Leftarrow 0$;
initialize population $P(i)$ of unique trajectories;
evaluate $P(i)$ in the known environment with known obstacle motions;
repeat
  $i \Leftarrow i + 1$;
  call modification();
  evaluate $P(i)$;
until *the best feasible trajectory has not become better in 1000 generations*;
output best feasible trajectory

Moreover, the initial population can also include trajectories based on preplanned paths feasible with respect to known static obstacles. Such preplanned paths can be obtained from an existing method (e.g., PRM or RRT) or from running our offline planner (see Algorithm 3). In this way, RAMP can take advantage of the existing offline planners' capabilities in dealing with challenging static environments (such as handling "narrow static passages" as some PRM variants are focused on).

V. Fitness Evaluation

The use of explicit fitness evaluation functions enables flexible applications of different optimization criteria and combination and aggregation of multiple criteria. In our planner, fitness evaluation has two components: *feasibility checking* and *optimization criteria*. We use two different evaluation functions for feasible and infeasible trajectories. In each case, the evaluation function is a cost function to measure the fitness of a trajectory. The higher the value of the evaluation function, the worse or less fit a trajectory is.

A. Feasibility Checking

We currently use two hard constraints to define feasibility of a trajectory: collision-free and singularity-free.

Once a trajectory is generated (Section III), our planner checks a discretized trajectory for feasibility. Thus, it needs to make sure that there is no missed collision or singular configurations between two adjacent, discrete configurations. Our strategy to avoid missing collisions is to enlarge the obstacles slightly before collision checks. In the same spirit, we consider a configuration whose manipulability cost [38] is higher than a threshold, i.e., very close to a singular configuration, to be a singular configuration.

A trajectory is feasible if it is collision-free and singularity-free during the entire time period. A trajectory segment is feasible if every interpolated configuration is feasible at its time step; likewise, if all trajectory segments are feasible, a trajectory is feasible. Otherwise, it is called infeasible.

B. Evaluation Function for Feasible Trajectories

The evaluation function for a feasible trajectory combines three optimization criteria, which are in fact soft constraints:

1) time (minimize);
2) energy (minimize);
3) manipulability (maximize).

The execution time of a trajectory is simply the sum $\sum_j T_{\max,j}$ of the time durations of all its segments.

Since the motions of the mobile base and the manipulator arm are not decoupled so that they can happen together, minimizing time as a measure alone cannot differentiate an efficient trajectory with minimum motion from another one with unnecessary arm or base movement while they have the same execution time. Therefore, we use minimum energy as another measure of optimality. This way, we can distinguish between two trajectories whose time requirements are equal but energy requirements are not; the one that requires less energy is preferred.

The energy calculation for a trajectory is a simple but effective approximation: our interest is not in determining the precise energy consumption in executing a trajectory, but rather in comparing and ranking energy consumptions of different trajectories. Therefore, an estimate is sufficient for our need. We approximate the shapes of the base and the arm links of a mobile manipulator by cylinders to simplify the computation of the inertia tensors. After a feasible trajectory is generated, we compute the energy consumption of each link (treating the base as link 0) in terms of the total kinetic energy changes of the link during the entire trajectory and sum up energy consumption of all links as the energy consumption of the trajectory.[2]

Finally, to evaluate the manipulability associated with a trajectory, we take the manipulability measure at each configuration on each trajectory segment. The inverse of this value will grow in proportion to the proximity to a singularity, and can therefore give a measure of cost. We take the average of such inverse values along the whole trajectory as the cost related to manipulability.

The overall fitness value for a feasible trajectory is a combination of the energy cost E, the time cost T, and the cost M related to manipulability of the trajectory

$$\text{Cost} = C_1 \frac{E}{a_1} + C_2 \frac{T}{a_2} + C_3 \frac{M}{a_3}$$

where $C_i, i = 1, 2, 3$, is a weight that indicates the importance of the respective cost, and a_i is a normalization factor determined as the estimated largest respective cost.

C. Evaluation Function for Infeasible Trajectories

If a trajectory is infeasible, we define a fitness value as the sum of a dominating penalty term P and the trajectory's evaluation function value $Cost$ as if it were feasible.[3] The large penalty term P serves two purposes. One purpose is to make sure that infeasible trajectories are less fit than feasible trajectories. The other is to serve as a measure of relative safety so that infeasible trajectories with smaller penalty terms are considered safer and therefore fitter than ones with larger penalty terms. For the latter purpose, we define the penalty term of an infeasible trajectory as $P = Q/T_{\text{coll}}$, where Q is a large constant[4] and T_{coll} is the

[2]Note that since vertical components in linear velocities are considered in the computation, changes in potential energy are already included. As all trajectories share the same starting state, they share the same starting energy.

[3]For a trajectory with singularities, we compute its "manipulability as if it were feasible" by excluding the singular configurations.

[4]Q is set to 10^4 in our experiments.

time before either the first predicted collision or the first singular configuration, whichever comes first, in the trajectory. That is, we consider an infeasible trajectory safer if it has a longer time before the first predicted collision/singularity. The value $Cost$ is used as an indicator of the potential fitness of an infeasible trajectory if it becomes feasible due to modification operations.

By allowing infeasible trajectories in a population, our algorithm aggressively maximizes the chances to optimize a robot's real-time actions efficiently. Often infeasible trajectories may lead to good feasible trajectories later.

D. Remarks

It should be noted that in addition to the optimization criteria considered, other criteria could be used and aggregated into the evaluation function for either feasible trajectories or infeasible trajectories, requiring changes only in the evaluation procedure, and not to the overall algorithm. We could choose to optimize feasible trajectories based on any number of criteria, including, for example, safety and stability measures [39]. For nonholonomic mobile manipulators, the nonholonomic constraints could be added as additional hard constraints for evaluating the feasibility of a trajectory and incorporated in the evaluation function for infeasible trajectories.

Note also that regardless of whether a trajectory is feasible or infeasible, the corresponding evaluation function is computed as the sum of the costs for individual trajectory segments. This property greatly facilitates efficient evaluation of trajectories in each generation of the RAMP algorithm since only the altered and affected trajectory segments need to be reevaluated, especially in real time. The evaluation of infeasible trajectories is further speeded up by that once the first collision is detected between a single link of a robot and an obstacle, the entire trajectory is labeled infeasible, and no further collision checking is required by the evaluation function (for infeasible trajectories).

VI. Modification Operations

Recall that in each generation of our RAMP algorithm, certain modification operations are performed on certain trajectories to generate hopefully fitter trajectories. We use the following six modification operations.

1) *Insert*—A new, random knot point is inserted between two randomly chosen adjacent knot points of a path.
2) *Delete*—A randomly selected knot point is deleted.
3) *Change*—A randomly selected knot point is replaced with a new, randomly generated knot point.
4) *Swap*—Two randomly selected adjacent knot points from a single path are swapped.
5) *Crossover*—The knot point lists of two paths are divided randomly into two parts, respectively, and recombined: the first part of the first path with the second part of the second path, and the first part of the second path with the second part of the first path.
6) *Stop*—The base movement or arm movement stops at a randomly chosen knot point for a random duration.

The first five operations are used to change the shape of a path, and subsequently, the corresponding trajectory. The *Stop* operation is used to change a trajectory only. We simply randomly select one of those operations (also called operators) to apply to the selected trajectory(s). All operators are used to change the trajectories of the base and the manipulator either separately or together in a stochastic fashion.

The *Stop* operator enables loose coupling of subsets of variables that have redundancy in a redundant robot, which, in the case of a mobile manipulator, means loose coupling of trajectories of the base and the manipulator. Both subsystems can stop their movements independently or together. The probabilistic nature of our approach simply offers a stop as a possibility; in the cases where stopping is advantageous, the planner will utilize it.

Note that except for *Crossover*, the other operations above are unary transformations that change a single trajectory. The crossover generates two offsprings from two parent trajectories.

The evaluation of a new trajectory can be very fast since each operation only alters certain trajectory segments, and only the altered segments need to be reevaluated. As shown in the RAMP algorithm (Algorithm 1), the fitter offspring is put back into the population to replace a trajectory that is not the sole member of a subpopulation to preserve diversity (more on this in Section VIII). The fittest trajectory in the entire population is always kept in the new population for the next generation $P(t+1)$. Note that $P(t)$ and $P(t+1)$ are of the same size and differ in one trajectory.

VII. Real-Time Adaptiveness

As shown in Algorithm 1, at the end of each control cycle, all trajectories in the population are updated so that their initial configuration and velocity becomes the robot's current configuration and velocity. Since a feasible trajectory may emerge as the result of continued planning, the manipulator may readily change course to execute this new best trajectory instead when the new control cycle begins. Note that when the robot changes course from one trajectory to another, the new trajectory is indeed better even after taking into account the cost of change (i.e., the possible acceleration or deceleration needed for the change) as ensured by the fitness evaluation function (Section V). Thus, the change is smooth and stable, and the actual trajectory executed by the robot is the best or most rational result.

When a change in the environment is sensed (from a sensing cycle), the constantly running planner will adapt the trajectory population to the change in real time in that trajectories are rechecked for feasibility and fitness values against the changed or changing part of the environment. The processing effort to reevaluate is kept to a minimum: the planner only checks for collisions against obstacles that have moved or are moving during *that* sensing cycle.

When there are moving obstacles, our planner predicts the future trajectory of each moving obstacle. From an obstacle's sensed configurations at the past two sensing cycles and the current sensing cycle, i.e., at time t_{i-2}, t_{i-1}, and t_i (the current time), we can compute (approximately) the linear and angular velocities of the object $\mathbf{v}(t_{i-1})$, $\mathbf{v}(t_i)$, and $\omega(t_i)$. The obstacle's motion is predicted as one of the four simple types: 1) translation only with nonzero $\mathbf{v}(t_i)$ if $\omega(t_i)$ is close to zero; 2) self-rotation only with nonzero $\omega(t_i)$ if $\mathbf{v}(t_i)$ is close to zero; 3) translation

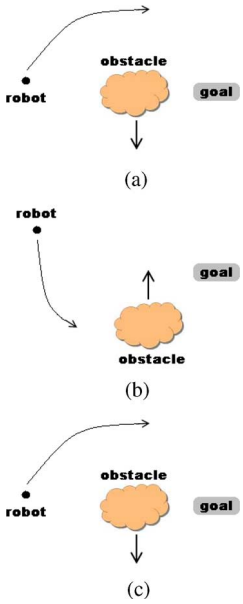

Fig. 4. Dynamic trap caused by high-frequency cyclic motion of obstacle. (a) Robot plans motion to avoid the moving obstacle. (b) Obstacle reverses direction; robot does likewise. (c) Obstacle reverses direction again; the process repeats.

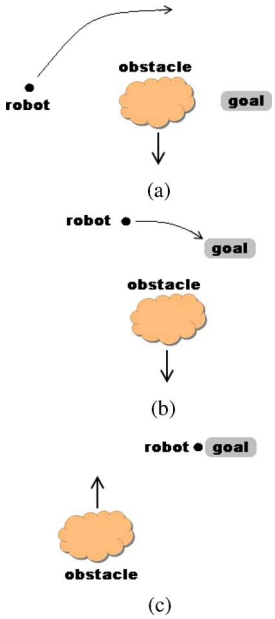

Fig. 5. Low-frequency cyclic obstacle motion does not present a trap. (a) Robot plans motion to avoid the moving obstacle. (b) Obstacle has not yet reversed direction. (c) Obstacle reverses direction, but robot has already reached goal.

and self-rotation with nonzero $\mathbf{v}(t_i)$ and $\omega(t_i)$ if $\mathbf{v}(t_{i-1})$ and $\mathbf{v}(t_i)$ have similar directions; and 4) a global rotation (about an axis not through the object) with nonzero $\mathbf{v}(t_i)$ and $\omega(t_i)$ if $\mathbf{v}(t_{i-1})$ and $\mathbf{v}(t_i)$ have different directions. The axis for global rotation is along $\omega(t_i)$ and away from the reference position of the obstacle by a distance $\mathbf{r}(t_i)$, which can be computed from $\mathbf{v}(t_i)$ and $\omega(t_i)$. The planner next checks the robot's trajectory against this predicted trajectory of each obstacle to see if there will be a collision. Our prediction only has to be good enough for a short period before the next sensing cycle (which may be longer or shorter than a control cycle) since it will be corrected constantly with newly added sensory information. Thus, the simple method suffices.

Note that to predict any cyclic or periodic behavior of an object, such as moving back and forth in opposite directions, more previous states need to be observed by our motion predictor, with more computation cost. Fortunately, if the cyclic behavior has a very high frequency, we can detect that from just a few previous states without adding much computation cost, and subsequently, the planner will take into account the cyclic behavior of an obstacle in planning feasible trajectories for a robot to avoid the dynamic trap. If the frequency of the object's cyclic motion is higher than or comparable to the sensing frequency, then as few as the past four sensing cycles can review the cyclic tendency. Conversely, if the cyclic behavior of an obstacle has a rather low frequency, we can afford not to detect it because it does not really trap a robot. Figs. 4 and 5 illustrate these concepts.

VIII. Diversity Creation and Preservation

The need for a diverse population of trajectories can be evident from the following example. Fig. 6 shows a task that requires a robot to pass through one of several doorways to the other side of the room. However, each door may open or close unexpectedly at any time; therefore, the robot should be always prepared to move out of a trajectory leading to a closed door and switch to a trajectory leading to an open door. As trajectories through different doorways belong to different homotopic groups and are thus quite diverse, it is necessary that the population of trajectories that RAMP creates and maintains be diverse with a good coverage of different homotopic groups. As shown in Fig. 6, a randomly generated initial population of base trajectories can be quite diverse. The key then is for the RAMP algorithm to preserve or further promote such diversity.

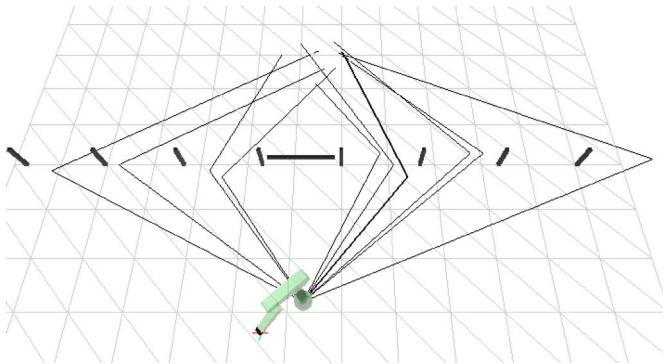

Fig. 6. Task that requires a population of trajectories as doors can close and open unexpectedly. The initial trajectory set has a good diversity to cover the environment. The figure indicates base trajectories only.

Clearly, since the robot operates in a changing environment with unknown dynamics, its CT-space cannot be known beforehand. Let alone that even for a static, known environment, how to construct the C-space of a high-DOF robot efficiently remains an open problem. Therefore, it is impossible to create all

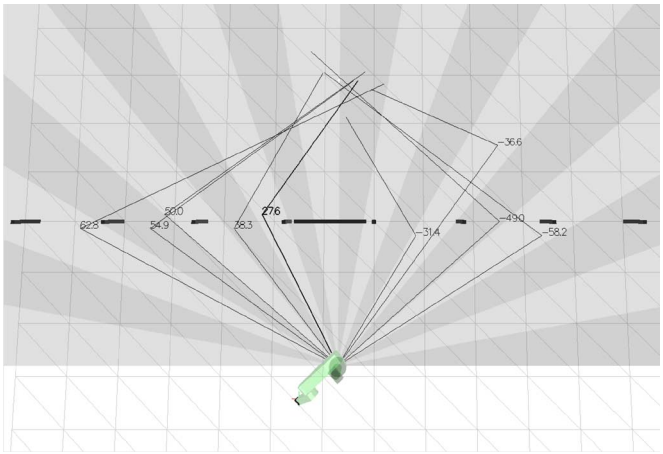

Fig. 7. Assignment of subpopulations based on departure directions at each control cycle: each subpopulation is indicated by a shade of gray.

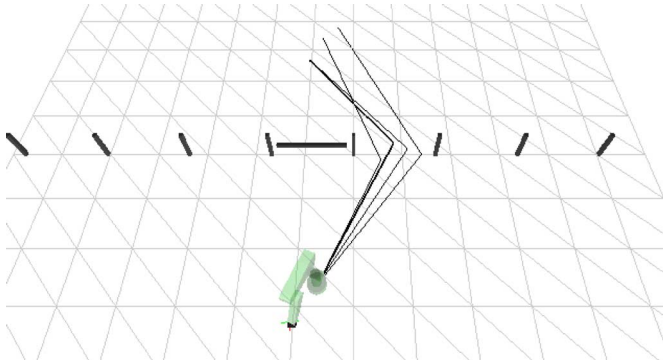

Fig. 8. Without diversity preservation, the fittest homotopic group dominates. The figure indicates base trajectories only.

homotopic groups of trajectories precisely at any moment. More viable measures are needed.

The RAMP algorithm preserves a diverse population of trajectories through the following measures: 1) not allowing identical trajectories in a population; 2) randomly selecting a trajectory for modification, rather than base the selection on fitness; 3) using modification operators that introduce drastic changes in trajectories; 4) replacing a randomly chosen trajectory rather than the least fit; and 5) creating and preserving subpopulations of trajectories. The last measure is very important and is further explained next.

A. Subpopulations and Diversity Preservation

Our RAMP algorithm divides a population of trajectories into subpopulations when a new control cycle begins based on their *departure directions* at that time. The departure direction of a trajectory is defined as the n-dimensional difference vector from the initial configuration to the first knot configuration (for a robot with n DOF). We calculate the angle ϕ between the departure directions $\mathbf{a}$ and $\mathbf{b}$ of two trajectories with the dot product:

$$\mathbf{a} \cdot \mathbf{b} = ab \cos \phi.$$

If ϕ is larger than a threshold, then the two trajectories can be viewed as belonging to two different subpopulations. From the example of Fig. 7, we can see a strong correlation between each base trajectory's homotopic group and the subpopulation it belongs to according to departure directions.

Grouping n trajectories based on similar departing directions can be expensive [as it requires a time complexity of $O(n^2)$]. Hence, we make the following compromise [with time complexity $O(n)$]: pick a reference departure direction $\mathbf{a}$, divide the range of ϕ into a number of intervals, and compare each departing direction to $\mathbf{a}$ to divide the population of trajectories into subpopulations corresponding to the intervals. These subpopulations still capture a rough level of diversity.

Note that since the subpopulations are updated or reassigned at each new control cycle from the updated departure directions, diversity over time is captured and preserved. If two trajectories are in two subpopulations at time t based on the departure directions of the *then* control cycle but become very similar at time $t + T$, then they will not likely be assigned to two different subpopulations again in the future control cycle corresponding to or after $t + T$. In essence, diversity is for providing more options each time a robot needs to decide which trajectory to follow, and that happens with every new control cycle. Using departure directions to capture diversity satisfies this requirement.

Recall that in each planning cycle (see Procedure modification in Section II), the RAMP algorithm will produce one or two new trajectories, and it will use the (better) new trajectory to replace an existing trajectory. The RAMP algorithm first randomly picks an existing trajectory and then checks if it is the only member of its subpopulation. If so, it will not be replaced, and the RAMP will randomly pick another existing trajectory, and so on. In this way, diversity is preserved.

Without such measure to preserve different subpopulations, the fittest subpopulation may dominate after a number of generations, as shown in the example of Fig. 8. If that door is suddenly closed, all trajectories will become infeasible. With diversity preservation, this is not likely to happen (see the result in Section IX-B and also in the video accompanying this paper available at http://ieeexplore.ieee.org).

Fig. 9 shows a different example to demonstrate the benefit of diversity preservation for arm trajectories. Here, the robot's forearm may take a number of different routes to go below the bars shown. Again, if these passages may be blocked unexpectedly, a diverse population of trajectories will help the robot to find quickly a feasible passage.

B. Population and Subpopulation Size

The lower bound on population size is directly related to the number of subpopulations.

In order to choose the appropriate number of subpopulations M, we must find an angle increment $\Delta\theta$ that divides the range $[-180^\circ, 180^\circ]$. For example, using 30° will divide this range into 12 groups. To find the smallest increment for a given environment, we consider the smallest dimension of any obstacle in the environment, which we denote as L, as well as the distance threshold D (the smallest allowable distance between the robot

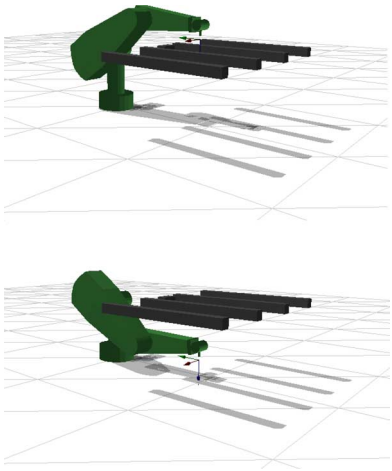

Fig. 9. Task to demonstrate the need for diverse arm trajectories, with the start configuration (top) and the goal configuration (bottom). The passages may be blocked and reopen unexpectedly.

and an obstacle). We compute an angle γ as a function of L and D as the upper bound on $\Delta\theta$

$$\gamma = \operatorname{atan}\left(\frac{L}{D}\right).$$

Of course, we may choose a $\Delta\theta$ smaller than γ, but γ gives us the minimum necessary resolving power for the environment.

We now have

$$M = \frac{360^\circ}{\Delta\theta}$$
$$N = KM$$

where N is the overall population size and K is the average size of each subpopulation ($K \geq 1$). We choose an appropriate K depending on the available computer processing power. In general, the upper bound on N can be determined based on the minimum number of planning cycles per control cycle to enable sufficiently fast improvements of trajectories.

The lower and upper bounds on N reflects the balance of exploration and exploitation capabilities of our RAMP planner. Our planner is quite robust in that it provides decent performance over different N within the bounds.

IX. Implementation, Results, and Discussion

In this section, we present our implementation results and discuss the performance of the RAMP algorithm.

A. Implementation

In order to test the RAMP algorithm, we built a mobile manipulator simulator for a PUMA 560 mounted to a mobile base. Both the robot and the objects in the environment are modeled as polygonal meshes for generality. We use the software package [40] to perform real-time collision detection.

To simulate environment dynamics, objects (or obstacles) are allowed to move in different ways during the trajectory execution; however, the planning algorithm has no *a priori* knowledge of these movements. At the beginning of each sensing cycle, the locations (i.e., positions and orientations) of the obstacles are provided as supposed to be from sensing. We implemented the RAMP algorithm in C# and C++, and the execution is on a four-core Xeon PC with each core operating at 3.0 GHz.

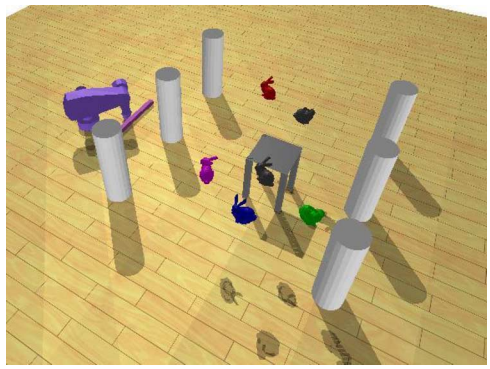

Fig. 10. Task environment 1—carrying a rod (with six dynamic obstacles and seven static obstacles).

In our experiments, we set the following parameter values. The weight of the manipulator arm and the base are set to be 35 and 20 kg, respectively. The maximum joint velocity and acceleration for the PUMA are set to be 120°/s and 60°/s^2, respectively. The maximum base velocity and acceleration are set to be 2 m/s and 1 m/s^2, respectively. The frequency of the control cycle for the mobile manipulator is set to be 60 Hz. The control cycle is therefore quite slow, as compared to the planning cycle, which has a frequency many times of that of the control cycle, depending on the task environment.

B. Performance Evaluation

We measure the performance of our RAMP planner in terms of effectiveness and efficiency.

Figs. 10 and 11 show two task environments.

1) In task environment 1, the robot's task is to move a long rod from the floor to the table. The table is positioned far enough away that a base movement is required. There are a number of static columns in the environment and six moving bunnies as dynamic obstacles: two revolve about the table with different angular velocities, parallel to the floor, and four move in different directions diagonally (i.e., neither vertically nor horizontally but across different altitudes).
2) In task environment 2, the task is to enter the second room and grasp the object on the counter. There is a static table in the first room where the robot is initially located. There are 12 dynamic obstacles of various shapes with changing trajectories; some change their direction and velocity, and some change from linear to angular motion at various times. Two of the dynamic obstacles move in the first room, and the rest move in the second room.

In order to measure the optimality of an executed trajectory generated by our real-time RAMP planner in a dynamic environment, where obstacle motions are not known in advance, we compare such a trajectory with a fully offline generated trajectory for the same task in the same dynamic environment but

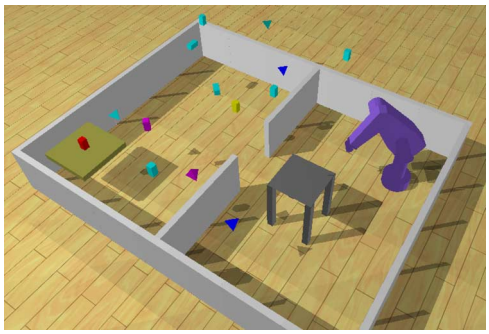

Fig. 11. Task environment 2—reaching an object on the counter (with 12 dynamic obstacles and static walls and table).

with *known obstacle motions*. This is because a fully optimal trajectory can only be produced when the environment is fully known.

We obtain the offline planner by slightly modifying our RAMP algorithm as shown in Algorithm 3. The offline planner runs as many planning generations as needed to generate a near-optimal trajectory connecting the starting configuration and a goal configuration. The planner will terminate when the best trajectory is not getting better in 1000 generations. Note that we used the index i to count the generations in the offline planner, which is *not* the absolute time t in the real-time planner RAMP.

Our offline planner searches entirely feasible trajectories connecting a starting configuration and a goal configuration in the *known* CT-space through essentially random sampling just as other randomized planners (such as the PRM or the RRT). It does offer advantages over the PRM and the RRT: it has the flexibility of optimizing under a number of optimization criteria in the entire CT-space rather than on a limited graph or roadmap. It generates a near-optimal solution just as an evolutionary algorithm will do given sufficient running time. Note that since the obstacles and their motions are entirely known, a good, feasible trajectory generated by taking account of all the obstacles and motions will always be good and feasible. Thus, subpopulations, which were introduced to cope with unpredictability in RAMP, are not necessary for this offline planner.

Table I compares the results of our RAMP planner and those of the offline planner in the two task environments shown in Figs. 10 and 11. All the results are the average over 25 executions of each task, with $M = 18$ and $K = 1.1$ (which is found to produce the best results on our hardware), so that the population size $N = KM = 20$ (see Section VIII-B). Column 1 lists the environments as shown in Figs. 10 and 11. Column 2 presents the overall cost (i.e., fitness value) of the executed trajectory by our real-time RAMP planner. Column 3 shows the overall cost of the near-optimal trajectory as the result of offline planning. Column 4 shows the percent increase of the real-time trajectory cost over the near-optimal trajectory cost, as a measures of the performance of our RAMP planner.

The test results show that a trajectory generated by the RAMP planner in real time carries a cost that is on average about 20%–30% higher than that of an offline planned, near-optimal trajectory. The energy cost E and time cost T are each about 20%–30% on average.[5] Of course, what we gain from this modest increase of cost is the ability to deal with unknown environment changes in real time, which is not possible with offline planning.

TABLE I
REAL-TIME VERSUS NEAR-OPTIMAL TRAJECTORIES AVERAGED OVER 25 EXECUTIONS ($N = KM = 20$)

Task/Env	*Real-time cost*	*Near-optimal cost*	*% increase*
1	27.2	22.2	22.6 %
2	29.6	24.5	21.3 %

TABLE II
RAMP ALGORITHM STATISTICS AVERAGED OVER 25 EXECUTIONS ($N = KM = 20$)

Task/ Env	*Avg. exe. time (s)*	*Avg. # plan. cyc.*	*Avg. plan. cyc. time (ms)*	*Plan. cyc. per control cyc.*
1	16.3	3736	4.4	3.82
2	20.3	4928	4.1	4.04

TABLE III
COMPARISON OF TRAJECTORIES WITH AND WITHOUT STOP OPERATOR

(a) Energy Cost (J) averaged over 25 executions ($N = KM = 20$)

Task/Env	*w/Stop*	*w/o Stop*	*% diff*
1	7.7	11.1	-30.8%
2	5.7	8.6	-33.0%

(b) Time Cost (s) averaged over 25 executions ($N = KM = 20$)

Task/Env	*w/Stop*	*w/o Stop*	*% diff*
1	16.3	16.2	0.1%
2	20.3	21.5	-5.7%

Table II presents the statistics of the RAMP planner applied to the two example tasks. It shows the planning efficiency of the RAMP planner. Since each task environment has a different length of travel distance for the mobile manipulator, the average execution time also differs.

In order to evaluate the effectiveness of loose coupling, we compare the results of our RAMP planner with the *Stop* operator fully functional against the results of the planner with the *Stop* operator not used. Table III shows the energy cost E and the time cost T with and without the *Stop* operator, and the percent increase of each.

Table III clearly shows the advantages of having the *Stop* operator: the energy costs are less compared to without the *Stop* operator and so is the time cost. The reduction in the energy cost is especially significant. This is because *Stop* permits the mobile manipulator to stop either the arm or the base (or both) to avoid moving obstacles whenever preferred in a nondeterministic fashion (see Fig. 2) rather than having the mobile manipulator "dancing around" in order to make the same kind of avoidance. The *Stop* operator saves energy not at the expense of wasting time, but saves time as well, as demonstrated by the test results.

Finally, we also used the two environments shown in Figs. 6 and 9, as task environment 3 and task environment 4, respectively, in testing the effectiveness of creating and preserving

[5]For the two example task environments, the time cost for real-time planned trajectories, which is also the time for execution, is in the range of 15–20 s, as indicated in Table II.

TABLE IV
PERFORMANCE WITH AND WITHOUT SUBPOPULATIONS AVERAGED OVER 25 EXECUTIONS ($N = KM = 20$)

Task/ Env	*Elapsed w/Subpop.*	*Elapsed w/o Subpop.*	*Forced Stops w/Subpop.*	*Forced Stops w/o Subpop.*
1	16.3	16.5	5.1	7.0
2	20.3	20.7	4.8	8.2
3	30.4	56.1	0.4	3.8
4	7.5	9.9	0.8	1.9

trajectory diversity through subpopulations. We compare the results obtained with subpopulations (i.e., $M = 18$, $K = 1.1$, and $N = KM = 20$) and without subpopulations (i.e., $M = 1$, $K = 20$, and $N = KM = 20$) in Table IV. Clearly, we want to minimize the time or the number of forced stops. Table IV shows that our technique of diversity through subpopulations can do that and reduce the total time of motion execution, i.e., the *elapsed time*.

The video accompanying this paper (available at http://ieeexplore.ieee.org) shows the real-time execution of several tasks, and one of them is the real-time execution of the task in the task environment Fig. 6 to demonstrate the effectiveness of diversity preservation through subpopulations.

C. Multiple Mobile Manipulators

Our RAMP approach also applies directly to scenarios with multiple mobile manipulators, where each robot does not know the motion of another robot and views other robots as obstacles to be avoided. In such a scenario, each robot has its own instance of the RAMP planner, and it views another mobile manipulator as consisting of seven or eight moving bodies (or obstacles),[6] due to the number of links (including load) of each mobile manipulator.

Figs. 12 and 13 show two different task environments with multiple mobile manipulators. The task environment 5 in Fig. 12 has four mobile manipulators cooperating in the task of picking up the 12 tennis balls. Each robot chooses a ball randomly, picks it up, and places it in the bucket in the corner. All robots repeatedly pick and place balls until no ball is left on the ground. During the process, for any one robot, the other robots present dynamic obstacles of unknown motions.

In task environment 6 of Fig. 13, three mobile manipulators share the same workspace but perform separate tasks: one picks up the tennis balls from the floor and places them in the bucket; one moves the small boxes from one table to the other; and one moves the long rods from one location to another. Each robot repeatedly moves one object at a time until all objects it is supposed to move are moved. In the process, the robots repeatedly cross paths, and therefore, create a considerably complex and uncertain dynamic environment for all the robots.

In both task environments 5 and 6, each mobile manipulator repeatedly executes a task (by moving multiple objects, one at a time) so that its instance of the RAMP algorithm is applied repeatedly with varied starting and goal configurations. The video

[6]The number of bodies depend on if the mobile manipulator holds an object or not.

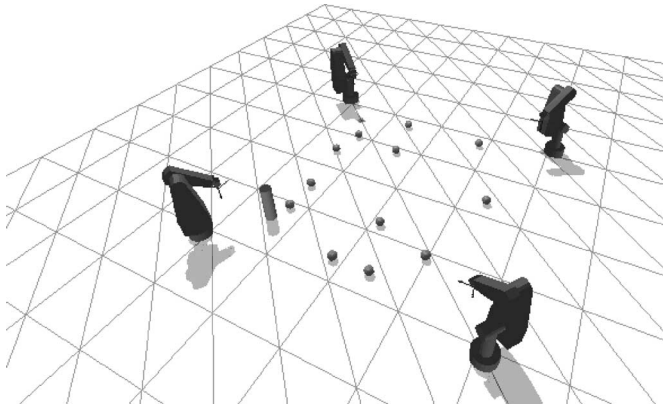

Fig. 12. Task environment 5—four robots picking up tennis balls.

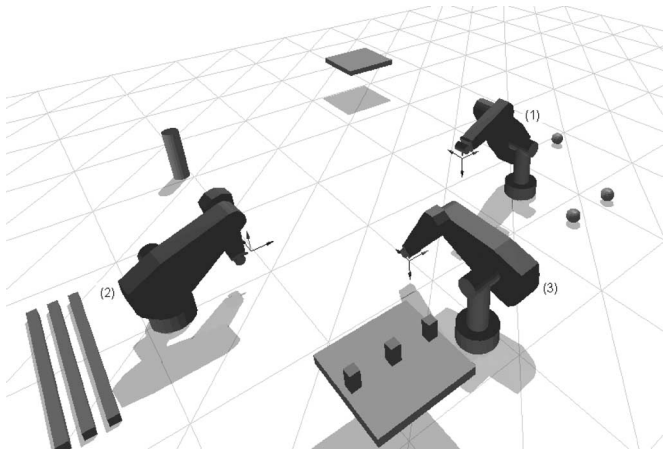

Fig. 13. Task environment 6—three robots performing various tasks.

accompanying this paper (available at http://ieeexplore.ieee.org) shows the real-time execution of these tasks.

In these multirobot task environments, we see a dynamic *interaction* among the simultaneous planning and execution processes of the different mobile manipulators: each mobile manipulator represents obstacles to other mobile manipulators and affects the trajectory planning and execution of others. These mobile manipulators affect one another's motion back and forth. In task environment 5, each robot decides spontaneously which ball to pick each time so that even its task goals are affected by the actions and movements of the other robots. The kind of spontaneous and dynamic interactions among the robots in both environments 5 and 6 are nondeterministic and cannot be known beforehand to enable offline motion planning. Whereas the RAMP planner suits these situations well with RAMP for each robot and is able to find and execute a feasible and near-optimal trajectory of the robot.

Table V presents the statistics of the RAMP planner for each mobile manipulator in the multirobot task environments 5 and 6. Each mobile manipulator has an instance of the planner, which runs on a separate CPU core, and therefore, the RAMP algorithm achieves similar real-time performance for each mobile manipulator as in the single-robot cases. For each RAMP instance, again, $M = 18$, $K = 1.1$, and $N = KM = 20$. The

TABLE V
RAMP ALGORITHM WITH TWO ROBOTS AVERAGED OVER 20 EXECUTIONS ($N = 20$)

Task/Env (Robot)	*Avg. exe. time (s)*	*Avg. # plan. cyc.*	*Avg. plan. cyc. time (ms)*	*Plan. cyc. / cont. cyc.*	*#Forced stops*
5(1)	86.6	14,684	5.9	2.8	0.28
5(2)	93.7	15,109	6.2	2.7	4.26
5(3)	95.5	15,878	6.0	2.8	5.84
5(4)	94.4	13,589	6.9	2.4	1.38
6(1)	86.1	14,126	6.1	2.7	7.29
6(2)	146.3	20,384	7.2	2.3	4.97
6(3)	138.9	19,061	7.3	2.3	3.25

average execution time in the table is the average total execution time for each robot to complete the task. In the case of task environment 5, this is the working time for each robot until all balls are put into the bucket; since the robots share the same task, they have comparable total execution time, implying that their work loads are more or less the same, even though the robots have different numbers of forced stops. In the task environment 6, however, each robot has a different job, and therefore, the total time to complete a job is different from one robot to another. As shown in Table V, the robot that moves the three long rods takes the longest time to complete its job. In both environments, as these robots work simultaneously, the longest time of an individual robot is in fact the overall time to complete all tasks.

We have also run the RAMP planner without subpopulation (i.e., $M = 1$ and $N = 20$) for robots in the environments 5 and 6, and the results show that the average execution time increases from 1% to 7%, and the number of forced stops increases from 0.3 times to 7.7 times for most of the robots, comparing to the results shown in Table V (with subpopulations). Hence, promoting diversity helps to produce better results.

We have also tested the RAMP planner in numerous other task environments in addition to the examples shown in this paper. In all cases, the RAMP algorithm for a mobile manipulator was able to avoid dynamic obstacles with unknown motions and allow the mobile manipulator to accomplish its task well.

X. CONCLUSION

This paper introduces a novel RAMP approach to plan high-DOF robot motion amid dynamic obstacles of unknown motions. The approach has the following characteristics.

1) It achieves real-time adaptiveness by planning path and trajectory together and also by simultaneous planning and execution of motion. This is accomplished by the unique design of the planner and also by exploiting the speed difference between physical motion and computer processing.
2) It effectively deals with drastic changes in the environment through global planning of diverse trajectories and through further preservation of diversity if needed.
3) It has the flexibility to incorporate different optimization criteria depending on the need without changing the overall planning algorithm.
4) It produces loosely coupled trajectories for redundant subsystems (such as the manipulation and locomotion subsystems of a mobile manipulator) to take advantage of redundancy in optimizing overall motion while avoiding unknown obstacle motions.
5) With its high efficiency and flexibility, it can also readily and effectively plan motions for each high-DOF robot (such as a mobile manipulator) in an environment shared by multiple such robots. Thus, the RAMP approach makes truly distributed planning possible for multiple high-DOF robots working in the same environment.

The RAMP approach is tested with simulations of mobile manipulators in different task environments, with very promising results. Future work includes further testing and improving the algorithm for more complex robots and robot tasks and incorporating realistic sensing scenarios and constraints. Testing on a real robot is also necessary.

REFERENCES

[1] J.-C. Latombe, *Robot Motion Planning*. Norwell, MA: Kluwer, 1991.
[2] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.
[3] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, 1996.
[4] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, B. R. Donald, K. M. Lynch, and D. Rus, Eds. Wellesley, MA: A K Peters, 2001, pp. 293–308.
[5] L. Jaillet and T. Simeon, "Path deformation roadmaps," presented at the 7th Int. Workshop Algorithmic Found. Robot., New York City, Jul. 2006.
[6] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2002, vol. 3, pp. 2383–2388.
[7] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed. New York: Springer-Verlag, 1996.
[8] T. Back, U. Hammel, and H.-P. Schwefel, "Evolutionary computation: Comments on the history and current state," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 3–17, Apr. 1997.
[9] P. P. Bonissone, R. Subbu, N. Eklund, and T. R. Kiehl, "Evolutionary algorithms + domain knowledge = real-world evolutionary computation," *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 256–280, Apr. 2006.
[10] P. Bessiere, J.-M. Ahuactzin, E.-G. Talbi, and E. Mazer, "The 'Ariadne's clew' algorithm: Global planning with local methods," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, Jul. 1993, vol. 2, pp. 1373–1380.
[11] X.-C. Li, D.-B. Zhao, J.-Q. Yi, and X.-H. Lu, "A coordinated and hierarchical path planning approach for mobile manipulators," in *Proc. Int. Conf. Mach. Learning Cybern.*, Aug. 2005, vol. 5, pp. 3013–3018.
[12] J. Xiao, Z. Michalewicz, L. Zhang, and K. Trojanowski, "Adaptive evolutionary planner/navigator for mobile robots," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 18–28, Apr. 1997.
[13] C. Zheng, L. Li, F. Xu, F. Sun, and M. Ding, "Evolutionary route planner for unmanned air vehicles," *IEEE Trans. Robot.*, vol. 21, no. 4, pp. 609–620, Aug. 2005.
[14] C. Hocaoglu and A. Sanderson, "Evolutionary path planning using multiresolution path representation," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 1998, vol. 1, pp. 318–323.
[15] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *Int. J. Robot. Res.*, vol. 21, no. 12, pp. 1031–1052, 2002.
[16] O. Brock and L. E. Kavraki, "Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces," in *Proc. IEEE Int. Conf. Robot. Autom.*, Apr., 2001, pp. 1469–1474.
[17] P. Fiorini and Z. Shiller, "Time optimal trajectory planning in dynamic environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1996, pp. 1553–1558.
[18] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, May 2001.

[19] J. van den Berg and M. Overmars, "Roadmap-based motion planning in dynamic environments," *IEEE Trans. Robot.*, vol. 21, no. 5, pp. 885–897, Oct. 2005.
[20] C. Ferrari, E. Pagello, M. Voltolina, J. Ota, and T. Arai, "Varying paths and motion profiles in multiple robot motion planning," in *Proc. IEEE Int. Symp. Comput. Intell. Robot. Autom.*, Monterey, CA, Jul.1997, pp. 186–193.
[21] D. Vasquez, F. Large, T. Fraichard, and C. Laugier, "High-speed autonomous with motion prediction for unknown moving obstacles," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2004, pp. 82–87.
[22] J. van den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, May, 2006, pp. 1243–1248.
[23] W. Carriker, P. Khosla, and B. Krogh, "Path planning for mobile manipulators for multiple task execution," *IEEE Trans. Robot. Autom.*, vol. 7, no. 3, pp. 403–408, Jun. 1991.
[24] H. Seraji, "A unified approach to motion control of mobile manipulators," *Int. J. Robot. Res.*, vol. 17, no. 2, pp. 107–118, 1998.
[25] H. Tanner and K. Kyriakopoulos, "Nonholonomic motion planning for mobile manipulators," in *Proc. IEEE Int. Conf. Robot. Autom.*, Apr. 2000, vol. 2, pp. 1233–1238.
[26] F. Pin, J. Culioli, and D. Reister, "Using minimax approaches to plan optimal task commutation configurations for combined mobile platform-manipulator systems," *IEEE Trans. Robot. Autom.*, vol. 10, no. 1, pp. 44–54, Feb. 1994.
[27] D. H. Shin, B. S. Hamner, S. Singh, and M. Hwangbo, "Motion planning for a mobile manipulator with imprecise locomotion," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, Oct., 2003, vol. 1, pp. 847–853.
[28] Y. Yamamoto and X. Yun, "Coordinating locomotion and manipulation of a mobile manipulator," *IEEE Trans. Autom. Control*, vol. 39, no. 6, pp. 1326–1332, Jun. 1994.
[29] J. Tan and N. Xi, "Unified model approach for planning and control of mobile manipulators," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2001, vol. 3, pp. 3145–3152.
[30] L. Jaillet and T. Simeon, "A PRM-based motion planner for dynamically changing environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2004, vol. 2, pp. 1606–1611.
[31] O. Brock, O. Khatib, and S. Viji, "Task-consistent obstacle avoidance and motion behavior for mobile manipulation," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2002, vol. 1, pp. 388–393.
[32] P. Ögren, N. Egerstedt, and X. Hu, "Reactive mobile manipulation using dynamic trajectory tracking," in *Proc. IEEE Int. Conf. Robot. Autom.*, Apr. 2000, vol. 4, pp. 3473–3478.
[33] J. Mbede, S. Ma, Y. Toure, V. Graefe, and L. Zhang, "Robust neuro-fuzzy navigation of mobile manipulator among dynamic obstacles," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2004, vol. 5, pp. 5051–5057.
[34] T.-Y. Li and J.-C. Latombe, "On-line manipulation planning for two robot arms in a dynamic environment," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 1995, vol. 1, pp. 1048–1055.
[35] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Proc. Conf. Neural Inf. Process. Syst*. Cambridge, MA: MIT Press, 2003.
[36] J. E. Bobrow, S. Dubowski, and J. S. Gibson, "On the optimal control of robotic manipulators with actuator constraints," in *Proc. Amer. Control Conf.*, Jun. 1983, pp. 782–787.
[37] K. G. Shin and N. D. Mckay, "Minimum-time control of robotic manipulators with geometric path constraints," *IEEE Trans. Autom. Control*, vol. 30, no. 6, pp. 531–541, Jun. 1985.
[38] T. Yoshikawa, "Manipulability of robotic mechanisms," *J. Robot. Res.*, vol. 4, no. 2, pp. 3–9, Apr. 1985.
[39] Q. Huang, S. Sugano, and I. Kato, "Stability control for a mobile manipulator using a potential method," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 1994, vol. 2, pp. 839–846.
[40] P. Terdiman. (2003). Opcode: Optimized collision detection [Online]. Available: www.codercorner.com/Opcode.htm

John Vannoy received the Ph.D. degree in information technology from the University of North Carolina, Charlotte, in 2007.

Since 1992, he has been a Professional Software Developer. His current research interests include real-time motion planning of robots with high degrees of freedom and distributed planning of multiple mobile manipulators in tight coordination or loose collaboration.

Jing Xiao (S'88–M'90–SM'04) received the Ph.D. degree in computer, information, and control engineering from the University of Michigan, Ann Arbor, in 1990.

She is currently a Professor of computer science at the College of Computing and Informatics, University of North Carolina at Charlotte, Charlotte, where she is also the Associate Dean for Research. From October to December 1997, she was a Visiting Researcher at the Scientific Research Laboratories, Ford Motor Company. From January to June 1998, she was a Visiting Associate Professor at the Robotics Laboratory, Computer Science Department, Stanford University. From August 1998 to December 2000, she was the Program Director of the Robotics and Human Augmentation Program, National Science Foundation. Her current research interests include robotics, haptics, and intelligent systems in general. She has authored or co-authored over 100 research papers.

Dr. Xiao is on the Membership Board of the IEEE Robotics and Automation Society and is also the Associate Vice President for Membership Activities. She has been a member of program committees of major IEEE robotics conferences for many years and has also served in various roles including the Program Co-Chair of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems and the General Chair of the 2005 IEEE International Symposium on Assembly and Task Planning.