Real-Time Adaptive and Trajectory-Optimized Manipulator Motion Planning

John Vannoy
Computer Science Department
University of North Carolina – Charlotte
Charlotte, NC 28223, USA
jmvannoy@uncc.edu

Jing Xiao
Computer Science Department
University of North Carolina – Charlotte
Charlotte, NC 28223, USA
xiao@uncc.edu

Abstract— **While there has been a large body of literature addressing offline path planning for manipulators, there is relatively less study on real-time motion planning that occurs as a manipulator moves in an environment with unknown obstacles or unknown changes. This paper introduces a unified and general motion planning approach based on evolutionary computation that is suitable for both offline and real-time adaptive motion planning for manipulators under various optimization criteria and manipulator constraints in environments with obstacles or changes not known *a priori*. The implementation and testing results demonstrate the effectiveness and efficiency of the approach.**

Keywords— manipulator motion planning, real time, adaptive, time optimal trajectory, environments with unknown changes

I. Introduction

Motion planning is a fundamental problem in robotics [7] concerned with devising a desirable motion for a robot to reach a goal, and motion planning for articulated robotic manipulators is usually more challenging than for mobile robots because of the high-degrees of freedom. A well-studied sub-problem is offline path planning, where the task is to find a suitable and often collision-free path between initial and goal configurations of a robot manipulator in a known and static physical environment.

Randomized algorithms, such as the PRM method [6] and the evolutionary approach [5], are found to be very effective in finding a collision-free path for a manipulator with high degrees of freedom because such algorithms avoid the hard (and largely open) problem of building high-dimensional configuration space explicitly by sampling the configuration space. The PRM method is particularly popular and has inspired considerable work on various improvements. One approach [10] improves the efficiency of PRM by pre-building a road map for a manipulator in an obstacle-free Cartesian workspace so that the presence of obstacles can be accounted for by modifying only the affected portion of the road map; as the result, path planning can be done in less than 1 second so that on-line path re-planning is possible if the environment changes.

More recently a different approach [2] was introduced for mobile manipulator path planning in real-time. The idea was to decompose the problem into three-dimensional workspace planning of collision-free volumes in terms of a "tunnel" connecting the initial and goal locations of the robot and to guide the manipulator moving through the tunnel by potential fields in the joint space for local obstacle avoidance. This resulted in efficient real-time path planning.

Unlike path planning, motion planning has to produce executable trajectories for a robot 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 [3], which can deform a trajectory locally to avoid moving obstacles inside a "tunnel" generated from path planning such as mentioned above. The other approach is to conduct path and trajectory planning simultaneously. However, most existing 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 [4, 9]. Only recently a method was introduced for dealing with unknown moving obstacles in mobile robot (vehicle) navigation [7].

This paper addresses the problem of simultaneous path and trajectory planning of a manipulator in an environment that can change dynamically in ways not known beforehand. For a manipulator to work in such an environment safely and efficiently, it needs to be able to plan feasible motion in real-time (to adjust to sensed changes). We introduce a unique manipulator motion planning approach based on evolutionary computation, which is able to produce not only collision-free but optimized trajectories in real-time because of the following characteristics:

- Different optimization criteria can be accommodated flexibly and easily in a seamless fashion. Optimization is done directly in the original, continuous space rather than being confined to a certain limited graph or roadmap. Trajectories and paths are optimized at the same time rather than making trajectory planning conditional to the results of path planning. Thus trajectories found are more optimized and adaptive to globally changing environments and situations.

- Sampling of collision-free configurations and search of feasible and optimized trajectories are done opportunistically through heuristic operators and steered by the optimization criteria rather than being done blindly. Thus search is very efficient.
- Whole paths/trajectories are represented at once and constantly evolved/improved during planning or simultaneous planning and execution, unlike algorithms that build a path/trajectory sequentially so that a whole path/trajectory can become available only at the end of the planning process. Our planner can provide a valid trajectory quickly and continues to produce better trajectories at any later time – which we may call *anytime* planner to suit the need of real-time planning.
- Our planner is intrinsically parallel with multiple valid and diverse trajectories present all the time to allow instant and, if necessary, drastic adjustment to adapt to newly sensed changes in the environment. This is different from planners capable of only local trajectory adjustment based on a set of homotopic paths [3]. It is also different from sequential planners such as anytime A* search [11], which speeds up A* search by loosening bound on solution optimality based on available search time. The latter also requires building a grid-like state space for search, which is a limitation that our planner does not have. Our planner works in the original continuous environment directly.
- Trajectory evolution (i.e., 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.

Note that, with the above characteristics, our real-time motion planner guides a manipulator to follow a feasible trajectory leading to the goal all the time (according to the knowledge up to the time). In other words, real-time *global* planning is provided. An approach similar in spirit was introduced in [15] but only for low-dimensional mobile robots and only for path planning rather than trajectory planning. The problem addressed in this paper, on the other hand, is more difficult and less studied.

The rest of the paper is organized as follows. Section II provides an overview of our planning algorithm and its components as well as optimization criteria for trajectory evaluation and the strategies for evaluation. Section III introduces the simulation environment for testing the planner applied to a PUMA 560 robot. Section IV presents some experimental results and discussions. Section V concludes the paper.

II. Planning Approach

One basic premise of our approach is that the planning process and the manipulator control process are interweaving to enable simultaneous robot motion planning and execution. This is achieved through our planning algorithm based on evolutionary computation and customized with effective use of heuristic knowledge.

A. Basic Planning Algorithm

Our planning algorithm has the following overall structure of evolutionary computation [12]: A *population of chromosomes* is initialized such that each chromosome represents a potential solution of the problem, which is a trajectory in our case. The *fitness* of each chromosome is evaluated through an *evaluation function* coding the optimization criteria. This population is then evolved to be a fitter population through iterations of improvements, called *generations*. In each such generation, certain chromosomes are selected and altered by certain *genetic operators* to form offspring. If an offspring is better than the worst chromosome in the population, it is used to replace the worst chromosome and therefore improves overall fitness of the population. The fittest chromosome in a population represents the best trajectory up to that moment. After a number of generations, the fittest chromosome in the population gives the near-optimal trajectory.

B. Trajectory Representation and Initialization

To facilitate real-time simultaneous planning and execution, a chromosome in our planner is defined as a trajectory from the manipulator's *current* configuration (with certain velocity and acceleration) to its goal configuration. It is represented in an ordered list of successive *trajectory segments*, where the intersection configuration of two adjacent segments defines an intermediate *knot point*. A chromosome may consist of an arbitrary number of knot points. The data structure for each segment contains the bounding knot points of the segment, the feasibilities of these knot points (i.e., whether they are collision-free), the fitness information of the segment itself, and the desired velocities and accelerations at the bounding knot points if the segment is collision-free, all of which are obtained through fitness evaluation of the chromosome based on some optimization criteria (see next section).

Each chromosome in the initial population is generated as follows. First, the initial manipulator configuration is used as the first knot point, and the goal manipulator configuration is used as the last knot point. Next, a random number of intermediate knot points are decided, and each intermediate knot point is a randomly sampled configuration. Now these knot points define a sequence of *path segments*. The rest of the information about the corresponding trajectory segment in the segment data structure is determined through fitness evaluation of the chromosome.

C. Optimization Criteria and Fitness Evaluation

One of the main strengths of the evolutionary technique is the use of explicit fitness evaluation functions that enables flexible applications of different optimization criteria and combination and aggregation of multiple optimization criteria.

For our manipulator motion planning problem, the hard optimization constraints are *collision and singularity avoidance*. A chromosome is *feasible* if the trajectory is collision-free and singularity-free. Note that a path segment is obtained from linear interpolation (in joint space) of configurations between the two bounding knot points. The corresponding trajectory segment is based on the trajectory of linear interpolation with parabolic blends in joint space and denoted by a sequence of configurations as a function of time: $\Theta(t)$, $t_1 \leq t \leq t_2$. The trajectory segment is collision-free and singularity-free if every interpolated configuration is singularity-free and is collision-free at its time step. If all trajectory segments are collision-free and singularity-free, the chromosome is called feasible. Otherwise, it is called *infeasible*.

We use two different evaluation functions for feasible and infeasible chromosomes. 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. However, a feasible trajectory is always considered to have better fitness than an infeasible one. Note that by defining fitness for not only feasible trajectories but also infeasible ones and by including both types of chromosomes in the evolution process for trajectory improvement rather than discarding infeasible ones, our algorithm does not overlook any useful information represented in infeasible chromosomes and thus maximizes the efficiency and effectiveness of generating near-optimal feasible trajectories.

The evaluation function for a feasible chromosome combines two optimization criteria: *time-optimal trajectory* and *manipulability*. To measure the former, we use the minimum time needed for the manipulator to move through all path segments, taking into account constraints on joint speed and acceleration of the manipulator. First, the minimum execution time T_{ij} of each joint i for each path segment j is calculated based on the trajectory of linear interpolation with parabolic blends in joint space under the maximum acceleration and maximum speed constraints of joint i. Next, the maximum of T_{ij} among all joints, $T_{max,j}$, is considered the minimum time to complete the path segment j and is used as the fitness value of the trajectory segment j. The sum $\sum_j T_{max,j}$ of all trajectory segments is considered the minimum time needed to complete the whole path in linear trajectory with parabolic blends. Note that to find the true minimum time, more sophisticated methods taking into account dynamics and torque constraints [1, 13] can be used to determine trajectories.

To evaluate the manipulability associated with a feasible trajectory, we take the manipulability measure at each configuration [16] on each path segment, which, for a square manipulator Jacobian, is simplified as the determinant of the Jacobian. The inverse of this value will grow in proportion to the proximity to a singularity, and can therefore give a measure of cost. The average of such values along the whole path is aggregated with the minimum time cost of the trajectory to form a single fitness value for the trajectory.

If a trajectory is infeasible, the corresponding evaluation function is defined as the number of collision configurations (at their respective time steps of the time-optimal trajectory as computed above) and singularity configurations found along the path. This number is the sum of the number of collision configurations and singularity configurations on each trajectory segment.

It is worth emphasizing that whether a trajectory is collision-free is evaluated against not only sensed static obstacles but dynamic obstacles with unknown trajectories. We track the velocity of each sensed moving obstacle continuously in each sensing cycle and use the information to predict the trajectory of the obstacle in order to check if the obstacle will collide with the robot at the same time on a particular trajectory of the manipulator. To be safe our prediction is conservative in that we do not assume that the obstacle will suddenly reduce its velocity or stop. This enables our planner to make judgment just as humans do: when we humans see an obstacle moving towards us, we try to avoid it under the assumption that it won't stop or reduce its speed or change directions *until we sense the changes*. Of course, this does not guarantee that our robot will surely avoid colliding with any moving obstacle, but without knowing the actual trajectory of such an obstacle beforehand and by relying on only real-time sensing, that is the best one can do. If the trajectory or velocity bound of a moving obstacle is known, then a guaranteed collision-free trajectory for the manipulator can be determined if one exists.

It should be noted that in addition to the above criteria, other criteria could be used and aggregated into the evaluation function for either feasible chromosomes or infeasible chromosomes, requiring changes only in the evaluation procedure, and not to the overall algorithm. We could choose to optimize feasible chromosomes based on any number of criteria, including, for example, the sum total degrees of joint rotation or the amount of energy consumed. For non-holonomic mobile manipulators, the non-holonomic 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 planning algorithm since only the altered or affected trajectory segments need to be re-evaluated, especially in real-time (see the following two subsections).

D. *Genetic Operations*

Recall that in each generation s of the planning algorithm, certain genetic operations are performed on certain chromosomes to generate hopefully fitter offspring. There are many ways to design and select genetic operations and the chromosomes to be operated on in the literature of evolutionary computation [12, 15]. In our current implementation, we simply randomly choose one of the following genetic operations, each of which is designed heuristically to change the shape of a path:

Insert – a new, random knot point is inserted between two randomly chosen adjacent knot points of a path.

Delete – a randomly selected knot point is deleted from the path.

Mutate – a randomly selected knot point is replaced with a new, randomly generated knot point.

Swap – two randomly selected adjacent knot points from a single path are swapped.

Crossover – the knot point lists of two parent 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.

Note that the first four operations above are unary transformations that produce an offspring by changing a single parent chromosome, which can all be called mutation operations. The crossover is a reproduction operation that generates two offspring from two parent chromosomes.

Depending on if the selected operation is of mutation or crossover, one or two chromosomes from the current population $P(s)$ are selected at random. One or two new chromosomes are generated by applying the selected genetic operation to the selected chromosome(s) and are then evaluated. The evaluation of such a new chromosome can be very fast since the genetic operation only alters certain path segments, and only the altered segments needs to be re-evaluated. The fitter offspring is put back into the population to "squeeze out" the worst chromosome so that the new population for the next generation $P(s+1)$ is fitter. Note that $P(s)$ and $P(s+1)$ are of the same size and differ in one chromosome. This selection procedure is quite effective for our purpose of improving population fitness while maintaining diversity at the same time, but it is by no means the only way: other alternatives may also work.

E. Real-time Improvement and Adaptiveness

The control process of a manipulator normally consists of a sequence of *control cycles* for the controller to issue motion commands to operate the manipulator, with a known frequency. Conversely, our planning algorithm based on evolutionary computation plans motions iteratively through generations. We shall call each such generation the *planning cycle*. As we shall see, the frequency of the control cycle is much lower than the frequency of the planning cycle, which affords us the opportunity to perform motion planning and execution simultaneously.

The process of simultaneous manipulator motion generation and execution begins once the planner generates at least one feasible trajectory chromosome relative to the known information of the environment from the initial configuration to the goal configuration; this usually takes 20—50 generations. Now the manipulator controller can start the first control cycle by commanding the manipulator to follow the time-optimal trajectory of the best feasible chromosome. During the control cycle, as the manipulator moves, the planner continues to run with more planning cycles and continues to improve the fitness of the chromosome population. The number of planning cycles executed within each control cycle is limited by the amount of time that elapses in a control cycle. As we will see later, one measure of the planning algorithm's efficiency is the number of planning cycles that can be executed within a control cycle.

At the end of the first control cycle, all trajectories in the population are updated so that their initial configuration becomes the manipulator's current configuration (see Section II.B). At this moment a better feasible trajectory may emerge as the result of continued planning. If so, the manipulator will readily change course to execute this (current) best trajectory instead, and a new control cycle begins. When a change in the environment is sensed (from a sensing cycle), the constantly running planner will adapt the chromosome population to the change in real time in that trajectories are re-checked for feasibility and fitness values against the part of the environment that has changed.

Such control/sensing/planning loop continues to move the manipulator towards the goal configuration while 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.

In general, by either only re-evaluating selected trajectory segments if there is no change in the environment or only checking against selected obstacles if there is a change, the result is a very efficient evaluation process.

Fig. 1 illustrates the relationship between planning, control and sensing cycles. Sensory information is assumed to be up-to-date at every control cycle, but this is not mandatory. The sensing cycle may be more or less frequent than the control cycle, but the precision of detecting environmental changes is obviously limited by the resolution of the sensing cycle, up to a maximum resolution of once per control cycle.

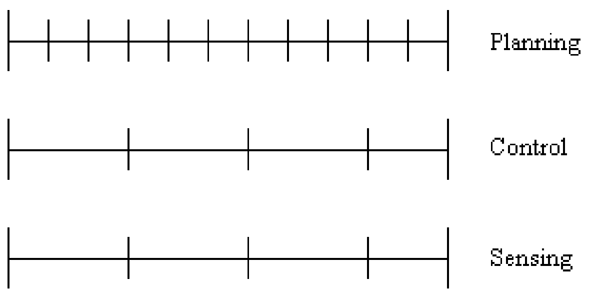

Figure 1. Relationship among planning, control and sensing cycles

It is worth noting that during the simultaneous planning and motion execution, when the manipulator changes course from one trajectory to another, the new trajectory is indeed more time-optimal even after taking into account the cost of change (i.e., the possible acceleration or deceleration time needed for the change) as ensured by the fitness evaluation function (Section II. C) so that the change is smooth and stable, and the actual trajectory executed by the robot is the best possible result.

III. PUMA 560 AND ENVIRONMENT SIMULATOR

In order to test the introduced motion planner, we build a manipulator simulator for PUMA 560 equipped with forward and inverse kinematics, manipulator Jacobian, trajectory generation and control/execution, as well as an interactive three-dimensional graphic display. Inverse kinematics is needed to allow the user to interactively move the manipulator to feasible initial and goal configurations, shown on the screen in Cartesian space.

The interactive, three-dimensional graphic display employs the mouse and keyboard for user input and features the following user interface functionality: (1) trackball-style rotation of the environment about two axes; (2) straight-line movement of the wrist position along all three axes; (3) control of the wrist rotation angles, about all three axes; (4) real-time display of the path population and trajectory execution.

Figure 2 shows three views of the simulator. The line segments shown are to illustrate the chromosome population (total 20), indicating the wrist position at each knot point in each path starting from the position where the gripper is at to the goal position. Note that the lines themselves are meant 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 view is to indicate the path with the highest fitness for its trajectory.

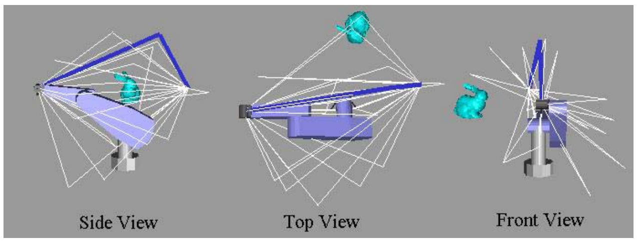

Figure 2. Illustration of object and path population (line segments)

Both the PUMA and the objects in the environment are modeled as polygonal meshes[1] to be general. Real-time collision detection is achieved by the software package [14]. To simulate environment dynamics, objects are allowed to move during the trajectory execution; however, the planning algorithm has no a priori knowledge of these movements. As explained earlier, the planning algorithm adapts to the environment dynamics in real time.

IV. Experimental Results and Discussion

Given the control cycle frequency and the acceleration and velocity constraints of the PUMA, we are able to apply our planner and simulate the simultaneous motion planning and execution of the PUMA realistically. In our experiments, the frequency of the control cycle is 50Hz, which is similar to that of an actual manipulator, and the maximum joint velocity and acceleration for the PUMA are set to be 120 deg/sec and 60 deg/sec^2, respectively. The control cycle of the manipulator is therefore quite slow, as compared to the clock cycle of the computer. The result of this is a surplus of computer clock cycles that are useful for motion planning, as the computer in effect "waits" for the manipulator to move. This is exactly as it would be using a real manipulator, in place of our simulator, since the speeds of the actuators are very slow as compared to the speed of a modern computer processor. One measure of the time efficiency of our algorithm can therefore be in terms of how many planning cycles can be executed during each control cycle when the motion is executed. Another measure is the total number of planning cycles during the motion that leads the manipulator from the initial configuration to the goal configuration.

[1] Thanks to Diego Ruspini for the PUMA mesh code.

Given the initial and goal configurations in an environment, the overall performance of the algorithm depends on the environment complexity:

- **static complexity** as determined by the number, size and arrangement of objects;
- **dynamic complexity** as determined by the number of moving objects and their (often unknown) trajectories.

Higher complexity results in a longer planning cycle, and thus fewer planning cycles executed between control cycles.

Figure 3 illustrates the working of the motion planner in a sample test environment that contains one static obstacle (floor) and two moving obstacles. The goal is to move the

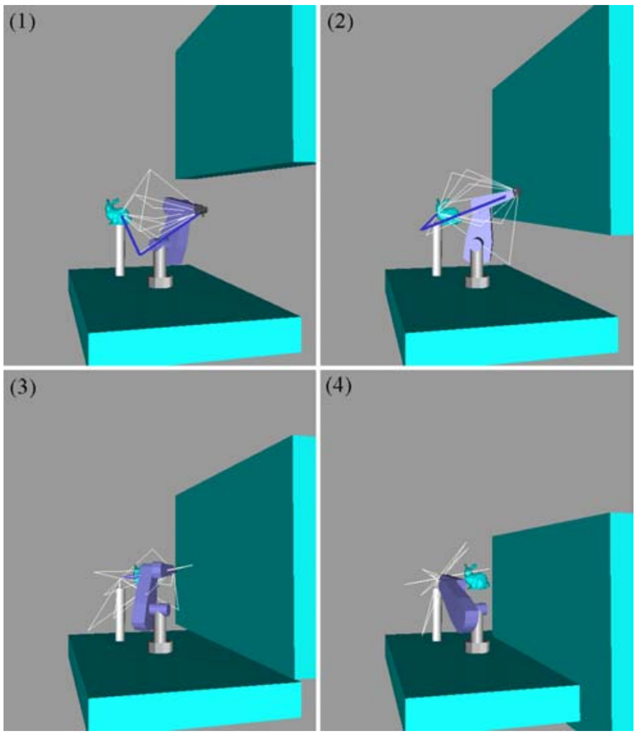

Figure 3. A sample environment

end-effector to the top of the white pedestal without collision. First, the rectangular solid moves down during the robot motion to close the open space between it and the floor. The robot trajectory initially passes through the open space, but as the rectangular solid moves down, closing up the opening, the planner adapts the robot trajectory to avoid collisions while still achieving the goal. Second, the bunny moves through the goal location of the end-effector directly from left to right during the robot motion, but since the planner tracks the obstacle's trajectory and plans the robot's trajectory accordingly, it enables the robot to reach the goal without colliding with the moving bunny. Note that the path population looks different in different snapshots, which shows the constant adaptation over time.

Table 1 summarizes the performance of the sample task on a 2.6GHz personal computer. The total elapsed time indicates the time used by the manipulator to complete the motion under its joint velocity and acceleration constraints. If the environment has no obstacle, the execution time of the motion from the initial to the goal configuration in linear trajectory with parabolic blends takes 3.8 seconds.

TABLE I. PERFORMANCE DATA OF THE SAMPLE ENVIRONMENT

Total Elapsed (s)	*Time per Planning cycle (ms)*	*Plan Cycles Per Control Cycle*	*Total Planning Cycles*
4.96	4.3	4.65	1156

We have tested our algorithm in different environments of different static and dynamic complexities. Figure 4 shows a setting with 5 bunnies. With different spatial arrangements of these bunnies and by allowing different bunnies to be either static or dynamic, we created different environments for the same task – the same start and goal configurations of the manipulator. The environments can be roughly characterized by the number of static and dynamic obstacles. Among the 5 obstacles, we change the dynamic nature of the environments from allowing only one object to move (i.e., free-fly) to allowing all objects to move in different directions. Table 2 shows the performance data measured over these different environments of varying complexity with the same computer. Note that *ENV Type* classifies the 5 different environments by the number of static (*S*) and dynamic (*D*) obstacles. The last row shows the average performance of all these environments. As the results show, the planning cycle of our planner for the entire population of trajectories, i.e., the planner update rate, has the average frequency of about 200Hz.

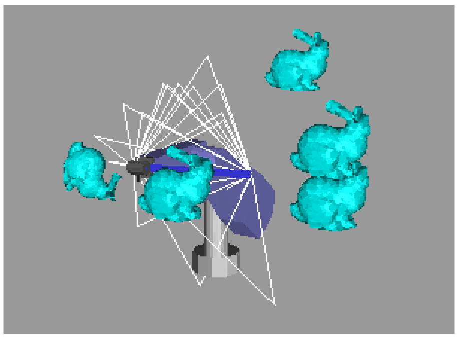
Figure 4. A setting with five bunnies

V. CONCLUSIONS

This paper addresses the problem of real-time adaptive and trajectory-optimized planning of manipulator motion in an environment with changes that are not known beforehand. A general approach based on evolutionary computation is introduced, which achieves simultaneous planning and execution of not only collision-free but also optimized motion, taking into account manipulator constraints. The method is tested on a PUMA 560 robot in simulated task environments of different static and dynamic complexity with promising results. Future work includes further testing and improving the algorithm for more complex robots and tasks and incorporating realistic sensing scenarios and constraints. Testing on a real robot is also necessary.

TABLE II. AVERAGE PERFORMANCE DATA OVER 50 EXECUTIONS FOR EACH OF THE 5 ENVIRONMENTS

ENV Type S, D	*Total Elapsed Time (s)*	*Time per Planning cycle (ms)*	*Plan Cycles Per Control Cycle*
4,1	4.62	4.27	4.68
3,2	4.80	4.61	4.34
2,3	5.65	5.14	3.89
1,4	6.24	5.49	3.64
0,5	6.49	5.97	3.35
Average	5.56	5.10	3.98

REFERENCES

[1] J. E. Bobrow, S. Dubowski, and J. S. Gibson, "On the Optimal Control of Robotic Manipulators with Actuator Constraints," Proc. Amer. Control Conf., June 1983.

[2] O. Brock and L. E. Kavraki, "Decomposition-based Motion Planning: A Framework for Real-time Motion Planning in High-dimensional Configuration Spaces," Proc. IEEE Int. Conf. Robotics & Automation, April 2001.

[3] O. Brock and O. Khatib, "Elastic Strips: A Framework for Motion Generation in Human Environments," *International Journal of Robotics Research* (*IJRR*), 21(12):1031-1052, 2002.

[4] P. Fiorini and Z. Shiller, "Time Optimal Trajectory Planning in Dynamic Environments," Proc. IEEE Int. Conf. Robotics & Automation, 2:1553-1558, 1996.

[5] C. Hocaoglu and A.C. Sanderson, "Evolutionary Path Planning Using Multiresolution Path Representation," Proc. IEEE Int. Conf. Robotics & Automation, May 1998.

[6] L. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars, "Probabilistic Roadmaps for Path Planning in High-dimensional Configuration Space," *IEEE Trans. Robotics & Automation*, 12(4):566--580, 1996.

[7] F. Large, D-A. Vasquez-Govea, Th. Fraichard, and C. Laugier, "Hihg-speed Navigation among Unkonwn Moving Obstacles," Proc. IEEE Intell. Vehicles Symposium, 2004.

[8] J.C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Norwell, MA, 1991.

[9] S. M. LaValle, J. J. Kuffner, "Randomized Kinodynamic Planning," *IJRR*, 20(5):378-400, May 2001.

[10] P. Leven, and S. Hutchinson, "Toward Real-time Path Planning in Changing Environments," Proc. Workshop Algorithmic Foundations Robotics, pp. 363-376, 2000.

[11] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with Provable Bounds on Sub-Optimality," Proc. Conf. Neural Info. Processing Sys., MIT Press, 2003.

[12] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd Edition, Springer-Verlag, New York, 1996.

[13] K. G. Shin and N. D. Mckay, "Minimum-Time Control of Robotic Manipulators with Geometric Path Constraints," *IEEE Trans. Automatic Control*, 30(6):531-541, 1985.

[14] P. Terdiman, http://www.codercorner.com/Opcode.htm.

[15] J. Xiao, Z. Michalewicz, L. Zhang, and K. Trojanowski, "Adaptive Evolutionary Planner/Navigator for Mobile Robots," *IEEE Trans. Evolutionary Computation*, 1(1):18-28, April 1997.

[16] T. Yoshikawa, "Manipulability of Robotic Mechanisms," *IJRR*, 4(2):3-9, April 1985.