Hierarchical Genetic Path Planner for Highly Redundant Manipulators

George Mesesan, Esra Icer, and Matthias Althoff

Abstract— **This paper is concerned with path planning of highly redundant manipulators, which are particularly well suited for flexible manufacturing. We envision that highly redundant manipulators will be composed of a set of standard modules. In order to find the optimal configuration of modules, one has to perform a discrete search on possible assemblies of modules combined with a continuous search of each assembly to verify the feasibility for completing the given task. This requires efficient planning techniques in the workspace since configuration space conversion is too demanding when considering many possible assemblies. For this reason, we propose a hierarchical path planner consisting of a global path planner (GPP) and a local motion planner (LMP). The GPP generates multiple collision-free paths in the workspace by using the free space structuring algorithm MAKLINK. The GPP passes the paths to the LMP, which determines their feasibility. If one path is found to be unfeasible, another one is chosen automatically. Along the path the LMP selects a number of points as intermediate goals and for each goal it solves the inverse kinematics problem. The LMP is based on a genetic algorithm that reuses the population from one intermediate goal to the next. This non-random initial population technique significantly reduces the joint angle variations, thereby increasing the probability that a collision-free path exists in the configuration space.**

I. Introduction

A current trend in robotics is to provide new solutions for highly flexible manufacturing processes. Due to the great demand of flexibility, these processes are typically performed by humans. A possible approach towards more efficient manufacturing is to use modular robots, which can be easily assembled from a small number of given modules. This often results in highly redundant robotic designs. A challenge towards modular robots is to find an optimal assembly of modules to perform a given task among all possible combination of modules. Consequently, fast planning algorithms are required since a very large number of possible assemblies have to be checked.

In this paper, we focus on fast planning methods when the assembly is given, such that as many assemblies as possible can be automatically investigated. Path planning of manipulators is a well-known problem [1]–[9]. Most approaches can be grouped into two main categories: configuration space (c-space) approaches and workspace approaches.

For a robotic manipulator with n joints, the c-space approach constructs an n-dimensional space, in which the robot is represented by a point [4]. The n coordinate values represent the joint angles of rotational joints or displacements of prismatic joints. The projection of an obstacle into the c-space is called a c-obstacle and is a set of configurations in which the robot collides with the obstacle. Path planning in c-space is relatively easy once the full extent of the c-obstacles is known, however computing the c-obstacles has an exponential complexity in n. The requirement for generating c-obstacles renders configuration-space-based approaches unfeasible for finding optimal assemblies of modular robots. Popular solutions that avoid computing the c-obstacles are potential fields [2], roadmap algorithms [5], [6], rapidly-exploring random trees (RRT) [7], and cell-based methods [8].

Workspace-based approaches try to find a solution directly in the workspace of the robot (usually the 3D environment) [3]. A common workspace-based approach is to compute a collision-free path for the end-effector. For certain points along this path, the inverse kinematics problem is then solved to find a robot configuration that places the end-effector in the desired position. Solving the inverse kinematics means searching in the n-dimensional configuration space and for this kind of problems genetic algorithms (GA) have been shown to provide good results [9], [10]. Typically, in those approaches the inverse kinematic problems are solved independently of each other, yielding solutions that might be far away in the configuration space. Even though the end-effector positions can be chosen arbitrarily close, the found c-space path might lead to a collision of the manipulator with nearby obstacles. In order to circumvent this problem, [1] proposes a genetic algorithm with non-random initial population, which greatly reduces the distances between adjacent points in configuration space. However, the planner in [1] selects only one path for the end-effector to follow and if this turns out to be unfeasible, the planner fails.

Our planner improves upon [1] by selecting multiple collision-free paths and finding the shortest feasible path. In Sec. II we describe our planning algorithm, followed by implementation details in Sec. III and the discussion of the results in Sec. IV.

II. Overall Approach

We present a workspace-based path planning approach consisting of a global path planner (GPP) and a local motion planner (LMP), inspired by [1].

A. Global Path Planner (GPP)

We assume that obstacles are represented by polygons, whose vertices are called *obstacle vertices*. The global path planner algorithm *MAKLINK* [11] uses the obstacle vertices to divide the free space into convex areas as shown in Fig. 1. The lines connecting obstacle vertices are called *free links*. The algorithm works as follows:

All authors are with the Department of Informatics, Technische Universität München, 85748 Garching, Germany {george.mesesan,esra.icer,althoff@tum.de}

- We select as candidate free links all segments connecting vertices of an obstacle to vertices of other obstacles or to the workspace boundaries. Only those segments are selected that do not intersect with the obstacle edges.
- We sort the candidates according to length, from the shortest one to the longest one.
- We select the candidates one by one and we check if the connected vertices are convex. A convex vertex is defined as a vertex where all angles formed by adjacent segments are less than $180°$. If the candidate segment makes a convex vertex or at least reduces the largest angle, it is selected as a free link.
- We repeat the step above until all obstacle vertices are convex.
- We remove redundant free links. A free link is redundant if its removal does not change the convexity of the vertices it connects.
- For all remaining free links, we select the midpoints (see Fig. 1a).
- We create a visibility graph by connecting all midpoints using segments that do not intersect the obstacle edges (see Fig. 1b).
- For any given start and end point, we construct multiple collision-free paths from the graph (see Fig. 1c).

It is important to note that MAKLINK was proposed for mobile robots, but it is being used here and in [1] for manipulator path planning. As manipulators have a fixed base, they cannot move freely in the workspace, so paths generated by MAKLINK are not always feasible, as can be seen in the examples below:

- In Fig. 2 and Fig. 3, the shortest path is seen to be dependent on the height of the workspace. The free link center point connecting the obstacle to the top of the workspace is lower in the workspace from Fig. 3. This makes the path going above the obstacle to be shorter than the path going below it. However, the manipulator cannot reach the goal on the other side of the obstacle by reaching over, but it can easily do so by reaching below the obstacle.
- In Fig. 4, the manipulator has already moved from its starting position. The base point is fixed and because of the leftmost obstacle the manipulator cannot reach the goal position on the right side by following the prescribed path.

The path planner in [1] selects the shortest path in the graph, which may turn out to be unfeasible. In this case, the LMP will fail to find a solution. Our planner will select multiple paths and is therefore more robust: if one path happens to be unfeasible, another one can be tried out instead.

An important concept in path planning is *homotopy*. Two paths are *homotopic* if one can be continuously deformed into the other. For example, in Fig. 2 the path going above the obstacle and the one going below are non-homotopic, as we cannot deform one into the other while keeping the end points fixed without going through the obstacle. A set of homotopic paths is called a *homotopic group*. Our planner will construct only non-homotopic paths, always selecting the shortest path from each homotopic group.

The paths are constructed using a depth-first traversal algorithm with a cutoff condition that ensures that the length of the path is not greater than 2.5 times the length of the robot. This maximum length was chosen empirically. Our algorithm keeps a set of shortest non-homotopic paths. For each new path, the homotopy with every path in the set is computed and if the new path is non-homotopic with all of them, it is added to the set. If a homotopic path is found in the set, the shortest one of the two is kept. This approach reduces the computation effort because for the purpose of finding the optimal assembly only the feasibility of the shortest path from each homotopic group is relevant. Even if another path in the homotopic group is feasible, we prefer a robot assembly that can follow the shortest path over one that requires a longer homotopic path.

In our approach, the GPP provides the LMP with each non-homotopic path in turn, starting with the shortest.

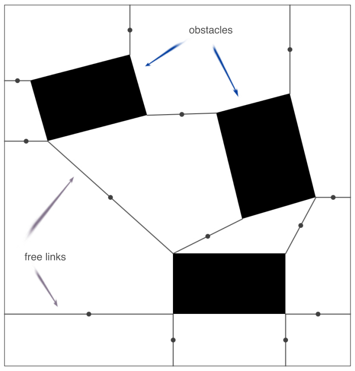
B. Local Motion Planner (LMP)

The generated global path consists of a sequence of vertices from the MAKLINK graph. Along the path additional equally-spaced points are interpolated linearly and a finer path is generated. We call these points intermediate goals, which the end-effector will follow towards the final goal. Finding the robot configuration that places the end-effector at the intermediate goal is done using a genetic algorithm. The main advantage of genetic algorithms consists in the ability to find minima (or maxima) of non-differentiable functions.

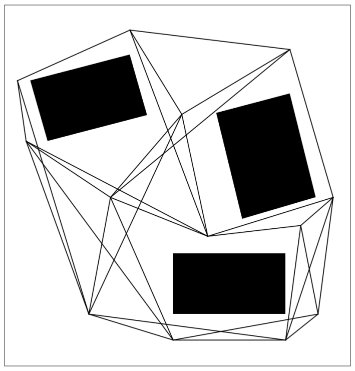
Genetic algorithms are based on the concept of *chromosomes*, which is an encoding of a solution to the problem at hand. A chromosome is a sequence of *genes*, each gene taking values from a predefined alphabet. Typical alphabets are binary (genes are 0 and 1), alphabetical ('a' to 'z') or floating point values. Each chromosome has a *fitness value*, which is typically the value of the objective function whose maximum we are looking for. An initial population of chromosomes is selected and is called the first generation. The population is evolved generation by generation using a set of three operations (Fig. 5):

- *selection*: the chromosomes with the highest fitness are selected for reproduction. Several algorithms exist: roulette, tournament selection, etc.
- *crossover*: offspring chromosomes are created from selected parents through the exchange of genes. Several algorithms exist: one-point crossover, two-point crossover, cut and splice, three parent crossover, etc.
- *mutation*: gene values of the offspring chromosomes are adjusted according to a mutation rate. The higher the rate, the higher the probability of a change.

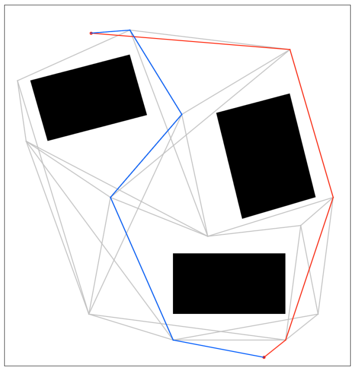
The fitness values for each chromosome in the new generation are computed and the process is repeated until a chromosome with a satisfactory fitness value is found or a maximum number of generations is reached.



(a) Free link division of the free space into convex areas



(b) Visibility graph for midpoints



(c) Collision-free paths in the visibility graph

Fig. 1: Global Path Planner

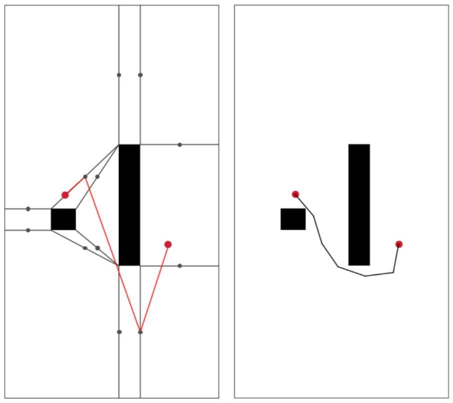


Fig. 2: Reachable goal on a path below the obstacle

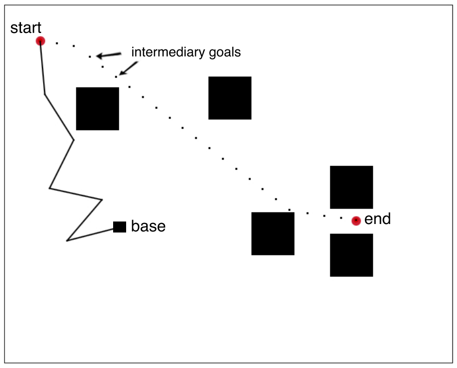


Fig. 4: Shortest path in the visibility graph

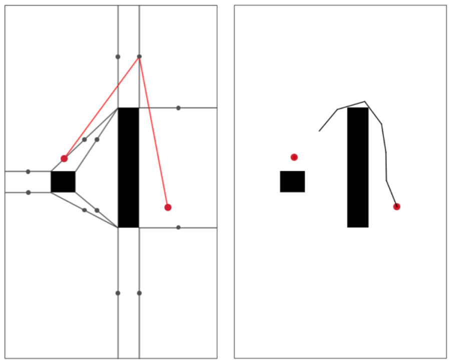


Fig. 3: Unreachable goal on a path above the obstacle

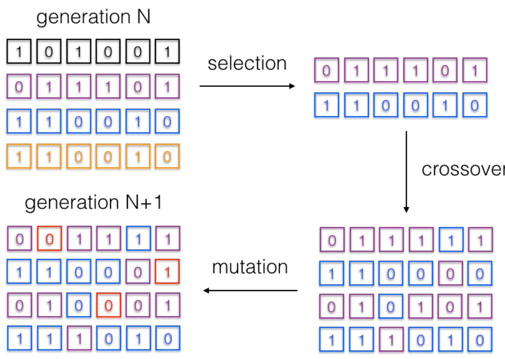


Fig. 5: Genetic algorithm operations

The Local Motion Planner with non-random initial population uses the algorithm in [1], which is also shown in Fig. 6. A chromosome in this algorithm is a point in the n-dimensional configuration space and is the sequence of the angle joints $[\theta_1, \theta_2, \ldots, \theta_n]$. The gene values are floating point values from the interval $[0, 2\pi)$. The fitness function is defined as follows:

$$F = f_{collision} \cdot D_s \ ,$$

where $f_{collision}$ is the collision coefficient, which is equal to 1 if the robot in the given configuration is collision-free or a high value constant (V_{max}) otherwise. D_s is the Euclidean distance between the end-effector position and the intermediate goal.

Once an intermediate goal is reached, the final population is reused as the initial population for the next intermediate goal. In a conventional genetic algorithm, the initial population is always randomly generated. However, a non-random initial population technique increases the probability that the solution for the next intermediate goal is not too far away

```
1:  $i \leftarrow 1$ {Initialize the first intermediate goal}
2:  $t \leftarrow 1$ {Initialize the genetic generations}
3:  randomly generate an initial population $P_i(t)$
4:  compute fitness $P_i(t)$
5:  **repeat**
6:      **repeat**
7:          select $P_i(t+1)$ from $P_i(t)$
8:          crossover $P_i(t+1)$
9:          mutate $P_i(t+1)$
10:         compute fitness $P_i(t+1)$
11:         $t \leftarrow t+1$
12:     **until** reached intermediate goal $q_i$
13:     $P_{i+1}(t) \leftarrow P_i(t)$
14:     $i \leftarrow i+1$
15: **until** reached final goal $q_n$
```

Fig. 6: Genetic algorithm with non-random initial population

from the previous goal's solution, thus making it more likely that the obtained c-space path is viable.

In some cases (see Fig. 7), the random search performed by the genetic algorithm might lead to incorrect solutions. Even though the LMP has found a new configuration that places the end-effector at the next intermediate goal, the movement is not possible as the robot has passed through the obstacle. Our planner detects this by checking that there are no obstacle vertices in the space swept by the manipulator. To do this, it constructs a polygonal shape (not necessarily convex) using the manipulator in the two adjacent configurations. If this polygonal shape contains a obstacle vertex, then the path is rejected as being unfeasible and the GPP will select an alternative path (see Fig. 8).

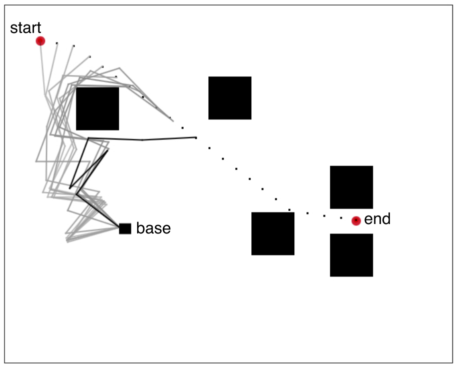


Fig. 7: Incorrect avoidance of an obstacle

III. Implementation

To demonstrate our path planner, we have implemented it using the Java programming language. The source code can be downloaded from an online Git repository[1]. In order to validate our implementation and compare the results, we recreated the examples of a 6-joint manipulator in a workspace with several rectangular obstacles as used in [1].

[1]https://bitbucket.org/GeorgeMesesan/hierarchical-ga-path-planner

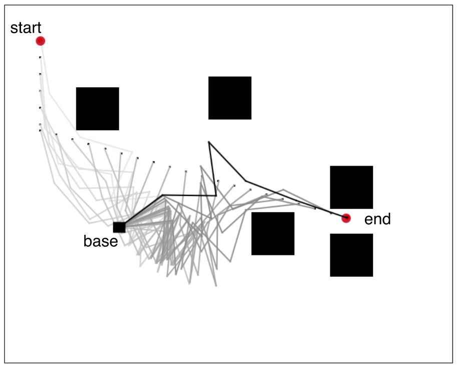


Fig. 8: Alternative path around the obstacle

Fig. 3 (Fig. 10 in [1]) tests the ability of the algorithm to find a path around a large obstacle and Fig. 4 (Fig. 15 in [1]) requires the manipulator to reach into a narrow passage.

A. Path Planner Robustness

To demonstrate the robustness of our planner, we compare it with another hierarchical planner, whose GPP considers only the shortest path. The LMP algorithm and its parameters are identical for the two planners. We perform two experiments, one for each workspace mentioned above. In each experiment, we run the planner for 1000 randomly selected pairs of start and end points and we count the number of solved path-planning problems. We run our experiments on a Mac mini with a 2.6 GHz Intel Core i5 processor.

We consider a planner to be successful if it finds a collision-free path that places the end-effector within a circle of predefined small radius around the final goal. The chosen radius value is 0.1 while, for comparison, the width of the workspace in Fig. 4 is 67 and the height is 53. To detect unfeasible paths early, we enforce the condition that for each intermediate goal the maximum distance between the end-effector and the goal does not exceed twice the distance between two intermediate goals. In Fig. 4 this condition will no longer hold when the end-effector is about halfway between the start and the end point, and the manipulator is fully extended and almost wrapped around the leftmost obstacle. In this case, no further progress is possible along the prescribed path.

For each run we choose a random base for the manipulator, while ensuring that it is outside any of the workspace obstacles. Further, while selecting the start and the end points, we ensure that both are outside the workspace obstacles and are reachable from the base point. We check reachability by running the GA once, thereby solving the inverse kinematics for the start and the end points.

For the workspace in Fig. 4, the planner that only considers the shortest path finds a solution in 771 cases, while our planner finds a solution in 987 cases with an average execution time for each case of 8.07 seconds. For the workspace in Fig. 3, the shortest-path planner finds a solution in 906 cases, while our planner finds a solution in 977 cases with an average execution time of 0.089 seconds.

B. Genetic Algorithm Parameters

For the selection of the genetic algorithm parameters, we considered the following criteria:

- the percentage of intermediate goals that were reached,
- the number of generations required to reach the intermediate goal,
- the mean of joint variations from one intermediate goal to another one, computed for each joint.

A high percentage of reached intermediate goals (high accuracy) and a low number of generations (high efficiency) are naturally desired. A low value for the joint variations means that the Euclidean distance in the configuration space between two chromosomes is small and the chance of finding a collision-free path in the configuration space is higher. In Fig. 9 we see two sequences of points in configuration space that correspond to the same end-effector position in the workspace.

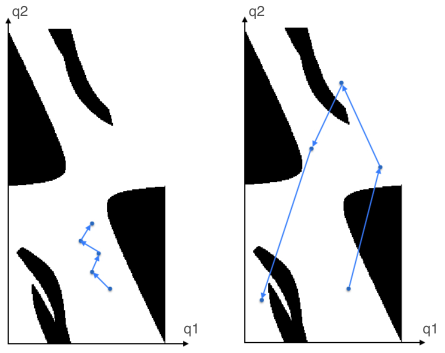


Fig. 9: Two paths in c-space, same end-effector path in workspace

A low value of joint variations is thus desirable and is achieved by reusing the population from the previous intermediate goal as an initial population, this being one the two main ideas in [1] (the other being the hierarchical approach). Some parameters of the genetic algorithm are explicitly mentioned in [1]:

- the population size is 100;
- the maximum number of generations is 600;
- the selection step selects the fittest 50% chromosomes;
- the crossover step exchanges one gene between two parents to generate two offspring;
- the mutation step changes one gene in a chromosome.

For the missing parameters we make the following choices:

- The collision punishment cost (V_{max}) is chosen as 1000 to heavily penalize those chromosomes, where the manipulator collides with the obstacles or with the workspace boundaries. Experiments have shown that smaller values (e.g. 100, 10) often produce solutions that are not collision-free.
- An intermediary goal is considered to have been reached if the distance between the end-effector and the goal is less than 0.1.
- The distance between two intermediary goals is 2.5. This value is computed using the formula $(w+h)/l$, where w and h are the width and the height of the workspace, respectively, while l is the total length of the manipulator.
- In the crossover step, the parents are chosen randomly between the fittest 50% chromosomes obtained through the selection step.

The most important parameter, i.e. the one with the biggest influence on the quality of the path planning has been found to be the mutation rate. The work in [1] mentions that the probability of mutation for the base joint is lower than that of the distal joint, however the exact mutation rates are not given. To determine these we analyze all combinations for the mutation rate of the base and distal joint from 0 to 0.9 in 0.1 intervals. As a mutation rate of 0 is not very useful, we use 0.01 instead. Also a mutation rate of 1, meaning that all genes change their values all the time, is not very useful and is not considered in the following tests.

For the intermediate joints an arithmetic progression is computed using the end joints. For example, if the mutation rate for the base joint is 0.2 and that of the distal joint is 0.6 then the mutation rates for all joints is the array $[0.2, 0.28, 0.36, 0.44, 0.52, 0.6]$. The mutation rates can also be decreasing, for example the base joint mutation rate is 0.8 and the distal joint mutation rate is 0.3. Then the full array is $[0.8, 0.7, 0.6, 0.5, 0.4, 0.3]$.

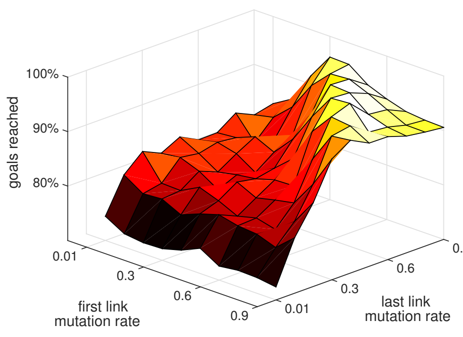
For each pair, the algorithm was run 100 times and averages for the mentioned quality criteria were gathered. The results for the workspace in Fig. 4 are in Fig. 10. The best values for the mutation rate in terms of efficiency, accuracy and low value of joint variations can be found along a line going from $(0.4, 0.9)$ to $(0.9, 0.4)$. These values can be shown to be appropriate also for the workspace in Fig. 3.

Finally, a comparison (Fig. 11) with a conventional GA implementation, where a random initial population is generated for each intermediate goal, shows that an important goal of the GA with non-random initial population is achieved: the average joint variations are reduced 4 to 8 times, thus increasing the probability of finding a feasible c-space path.

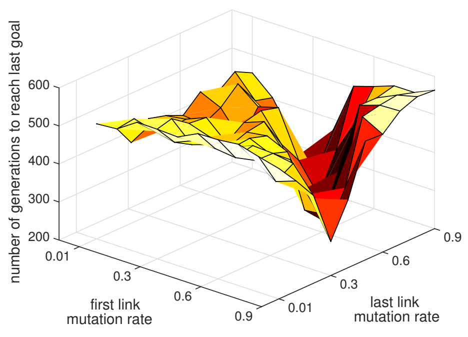
IV. Conclusion

Our path planner is a significant improvement to hierarchical path planners based on genetic algorithms. Our planner is more robust than the planner presented in [1], as it considers alternative paths towards the goal, not just the shortest path. By considering alternative paths, our planner improves the percentage of solved path planning problems (placing the end-effector withing a given distance of the target goal) from 77% to 99% in the case of the workspace in Fig. 4, and from 91% to 98% in case of the workspace in Fig. 2.

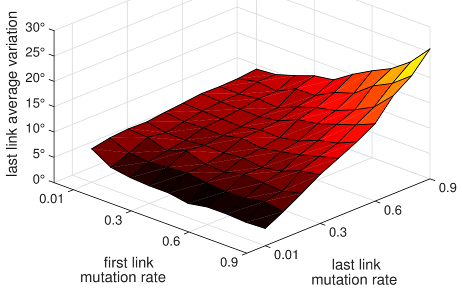
Our mutation rate analysis allows us to find appropriate genetic algorithm parameters, that place the robot end-effector within a defined distance of each intermediate goal in 96% of the cases. The presented genetic algorithm is much better than a conventional GA in creating a viable c-space path, as the average variations in the robot joints from one



(a) Percentage of intermediate goals reached



(b) Number of generations to reach last goal



(c) Last link angle variation

Fig. 10: Mutation rate analysis for the workspace in Fig. 4

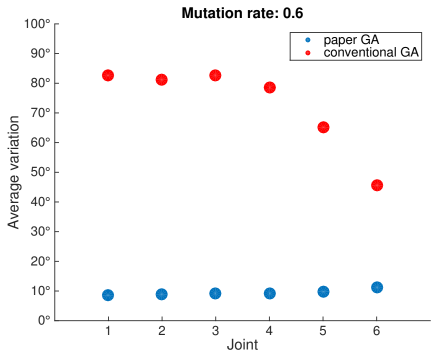


Fig. 11: Average joint variations for the workspace from Fig. 4

step to the next are 4 to 8 times smaller. Further, we found out that the genetic algorithm is only slightly more efficient when using non-random initial populations compared to random ones, which is in contrast to the claims in [1].

The genetic algorithm used by the local motion planner could find solutions that are not viable when applied in the c-space (See Fig. 7). We verify the viability of the solution directly in the workspace by checking that the space swept by the manipulator does not contain any obstacle vertices. If it does, then the solution is not viable and another path must be selected.

Another way would be to verify the viability of the solution by checking a linear path in c-space and if a collision is detected, to select another chromosome (c-space point) with high fitness and a clear path to the goal. This verification could be integrated into the fitness function, however we need to take into account the computation penalty associated with it. This viability verification in c-space is also necessary when the robot is in close proximity to obstacles and would therefore not be able to perform certain movements (for example from a elbow-up into an elbow-down position) without collisions.

The achieved performance of our path planner encourage us to consider it for the automatic search of robot assemblies using a given set of different modules. Its robustness in finding feasible paths for a single assembly and its short computation time open the possibility of checking many possible assemblies in a very short time.

Acknowledgment

The research leading to these results has received funding from the People Programme (Marie Curie Actions) of the European Unions Seventh Framework Programme FP7/2007- 2013/ under REA grant agreement number 608022.

References

[1] C.-C. Lin, *Hierarchical path planning for manipulators based on genetic algorithm with non-random initial population*, International Journal of Innovative Computing, Information and Control 8.3 (2012): 1773-1786.

[2] O. Khatib, *Real-time obstacle avoidance for manipulators and mobile robots*, The international journal of robotics research 5.1 (1986): 90-98.

[3] T. Lozano-Perez and M. A. Wesley, *An algorithm for planning collision-free paths among polyhedral obstacles*, Communications of the ACM 22.10 (1979): 560-570.

[4] T. Lozano-Perez, *Spatial planning: A configuration space approach*, Computers, IEEE Transactions on 100.2 (1983): 108-120.

[5] L. E. Kavraki et al, *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*, Robotics and Automation, IEEE Transactions on 12.4 (1996): 566-580.

[6] N. M. Amato and Y. Wu, *A randomized roadmap method for path and manipulation planning*, Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on. Vol. 1. IEEE, 1996.

[7] S. M. LaValle, *Rapidly-Exploring Random Trees A New Tool for Path Planning*, 1998.

[8] A. Chakravarthy and D. Ghose, *Obstacle avoidance in a dynamic environment: A collision cone approach*, Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on 28.5 (1998): 562-574.

[9] E. S. Pires and J. Machado, *A trajectory planner for manipulators using genetic algorithms*, Assembly and Task Planning, 1999.(ISATP'99) Proceedings of the 1999 IEEE International Symposium on. IEEE, 1999.

[10] J. K. Parker, A. R. Khoogar, and D. E. Goldberg, *Inverse kinematics of redundant robots using genetic algorithms*, Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on. IEEE, 1989.

[11] M. K. Habib, and H. Asama, *Efficient method to generate collision free paths for an autonomous mobile robot based on new free space structuring approach*, Proc. of IEEE/RSJ International Workshop on Intelligent Robots and Systems, vol.2, pp.563-567, 1991