

An Adaptive Framework for 'Single Shot' Motion Planning*

by

Christopher V. Jones

Submitted to the

Office of Honors Programs and Academic Scholarships

Texas A&M University

in partial fulfillment of the requirements for

1998-99 UNIVERSITY UNDERGRADUATE RESEARCH FELLOWS PROGRAM

April 15, 1999

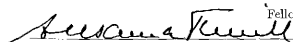
Approved as to style and content by:


Nancy Amato

Department of Computer Science

Susanna Finnell, Executive Director

Honors Programs and Academic Scholarships

 Fellows Group: Engineering I

*This research supported in part by NSF CAREER Award CCR-9624315 (with REU Supplement), NSF Grants IIS-9619850 (with REU Supplement), EIA-9805823, and EIA-9810937, and by the Texas Higher Education Coordinating Board under grant ARP-036327-017.

Abstract

This paper proposes an adaptive framework for single shot motion planning (i.e., planning without preprocessing). This framework can be used in any situation, and in particular, is suitable for crowded environments in which the robot's free C-space has narrow corridors. The main idea of the proposed framework is that one should adaptively select a planner whose strengths match the current situation, and then switch to a different planner when circumstances change. This approach requires that we develop a set of planners, and characterize the strengths and weaknesses of each planner in such a way that we can easily select the best planner for the current situation. Our experimental results show that adaptive selection of different planning methods enables the algorithms to be used in a cooperative manner to successfully solve queries that none of them would be able to solve on their own.

1 Introduction

Automatic motion planning has applications in many areas such as robotics, virtual reality systems, and computer-aided design. Although many different motion planning methods have been proposed, most are not used in practice since they are computationally infeasible except for some restricted cases, e.g., when the robot has very few degrees of freedom (dof) [11, 14]. Indeed, there is strong evidence that any complete planner (one that is guaranteed to find a solution or determine that none exists) requires time exponential in the number of dof of the robot [18]. For this reason, attention has focussed on randomized or probabilistic motion planning methods.

When many motion planning queries will be performed in the same environment, then it may be useful to preprocess the environment with the goal of decreasing the difficulty of the subsequent queries. Examples are the *roadmap* motion planning methods, in which a graph encoding representative feasible paths is built (usually in the robot's configuration space²). Queries are then processed by connecting the initial and goal configurations to the roadmap, and then finding a path in the roadmap between these two connection points. Recently, randomized or *probabilistic roadmap methods (PRMs)* have gained much attention for problems involving high-dimensional C-spaces [1, 2, 3, 4, 10, 12, 13, 15, 16]. Typically, the vertices of these roadmaps are obtained by sampling the robot's configuration space. Many difficult problems that could not be solved

before have been solved by these methods.

1.1 ‘Single Shot’ Motion Planning

If the start and goal configurations are known *a priori*, and only one (or a very few) queries will be performed in a single environment, then it is generally not worthwhile to perform an expensive preprocessing stage, particularly if there are time constraints as in animation or virtual reality applications. In this case, a more directed search of the free configuration space is needed (e.g., as opposed to roadmap methods which are designed to try to cover the entire freespace). Motion planning methods that operate in this fashion are often called *single shot* methods.

One of the first randomized planning methods was the Randomized Path Planner (RPP) of Barraquand and Latombe [5], which is a single shot planner. This method belongs to the general class of potential field methods, and uses random walks to attempt to try to escape local minima. In general, these methods can be quite effective when the C-space is relatively uncluttered, but there exist simple situations in which they can fail [6, 12].

Some success has been achieved in adapting the general PRM strategy to solve single queries by trying to restrict attention to ‘useful’ portions of the C-space [10, 16]. A related idea is to use a sample of free points to specify promising subgoals for planning [8, 9].

Single shot methods are also useful in dynamic environments where obstacles in the workspace can move between queries. For example, roadmaps are rendered obsolete when an obstacle moves and must be regenerated or updated before a new query can safely be made. Finally, single shot methods may be used as ‘local planners’ to connect roadmap nodes in PRMs.

1.2 Our Strategy for Single Shot Queries

Although much progress has been made, there are still important classes of problems for which good single shot solutions are needed. In particular, problems in crowded, or cluttered, environments in which the robot

³The configuration space (C-space) of the robot is the parametric space representing all possible positions and orientations of the robot in the workspace.

must maneuver through tight spots. The difficulty for such problems arises because successful planning requires one to find free configurations in so-called *narrow corridors* in the robot's configuration space, which is not a strength of the previous single shot methods (particularly when the dimension of C-space is high).

In this paper we propose a general framework for single shot planning which has been designed to tackle such problems. Briefly, our approach (described in detail in Section 2) is based on the following rather obvious idea:

Different planning methods have different strengths and weaknesses. Thus, we should adaptively select a planner whose strengths match the current situation, and then we should switch to a different planner when circumstances change.

Of course, as usual, the devil is in the details. This approach requires that we develop a set of planners, and characterize the strengths and weaknesses of each planner in such a way that we can easily select the best planner for the current situation.

For example, as we are particularly interested in problems involving narrow corridors in C-space, we need to develop methods for determining whether a configuration (e.g., the start or goal) is inside a narrow corridor, at the entrance of a narrow corridor, or in a relatively uncluttered area of freespace.

Similarly, we must identify/develop a set of planning methods and rank them according to their ability to navigate inside narrow corridors or in the free C-space, discover configurations in (nearby) narrow corridors, escape from narrow corridors, and determine subgoals for subsequent subqueries. Note that we do not require any planner to be very powerful, but only to perform well in some very specific situation. In some sense, this can be viewed as extending the philosophy of the PRM methods to single shot planning.

We have designed a framework for implementing this general idea, and results from our preliminary implementation have been very encouraging. In particular, our experimental results show that the adaptive selection of different planning methods enables the algorithms to be used in a cooperative manner to successfully solve queries that none of the planners would be able to solve on their own.

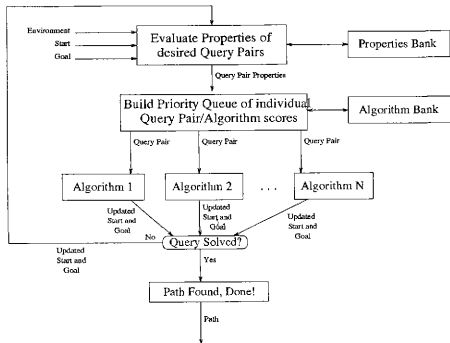


Figure 1: Single Shot Motion Planning Architecture.

2 A Framework for Single Shot Motion Planning

We propose a single shot motion planning architecture that will utilize a collection, or bank, of individual single shot motion planning algorithms (the individual planners are presented in Section 3). This architecture allows for the best suited algorithms to be executed for the current situation and enables the algorithms to be used in a cooperative manner. Anywhere from one to all of the algorithms in the bank may be executed during a single query. Figure 1 illustrates the structure of the architecture.

2.1 Evaluation Criteria

To select the best planner for the current situation, evaluation criteria are needed to characterize: (i) the local situation of the start and goal configurations in the current query pair, (ii) the possible path requirements of the current query pair, and (iii) the strengths (and weaknesses) of each planner. The evaluation of configurations and query pairs is the key aspect of our single shot architecture. It allows us to characterize the local environment of the current start and goal configurations and the approximate path requirements

of the query pair to select the most appropriate algorithm to execute at any given time during the query process. Each algorithm in the bank has preset characteristics corresponding to those of the configurations which enable us to perform the matching of algorithm to configuration.

In order for our single shot architecture to be effective, the most fitting algorithm must be chosen at each sub-query. We accomplish this by using a hierarchical based decision tree scoring approach [17]. The decision tree approach begins by determining general, low-level characteristics of individual configurations and query pairs, such as the distance to the nearest obstacle, relative locations of configurations in the query pair, or similarity of local environments of configurations in the query pair. Next, higher level characteristics, such as an approximate path, are evaluated. An approximate path is constructed by connecting the start and goal configurations of a query pair with the very fast straightline local planner using an extremely coarse step size. Based on the number of free configurations and the number of configurations in collision on this "approximate path," we can compute such things as how many obstacles are between the query pair or the length of approximate path segments in collision. This information may help us to infer the type of planning required to solve the query (i.e., general obstacle avoidance, narrow corridor navigation). This implementation allows for an expandable tree in that the more nodes added to the tree, the more detailed and accurate our evaluation of the configuration and query pair will be; thus, the more likely an appropriate algorithm will be matched with the query pair. With our current very simple decision tree structure, we have had encouraging success in the architecture's ability to select an appropriate algorithm to apply to the current query pair.

Each planning algorithm in the bank has characteristic values corresponding to how well it performs in the situations distinguished above. When an algorithm is added to the bank, values are assigned to these characteristics to represent the optimal or "ideal" working situation for the algorithm. If two algorithms in the bank have the same characteristic set, an additional, distinguishing, characteristic should be added. That is, the evaluation criteria should be sufficient so that the algorithms in the bank can be uniquely ranked for any situation. For example, we might rank each planner according to its ability to:

P1: navigate inside narrow corridors or in the free C-space and/or escape from narrow corridors,

P2: discover configurations in narrow corridors, or

P3: navigate efficiently in relatively free areas of C-space and/or determine subgoals for subsequent subqueries.

The ranking of the various planning methods is based on programmer expertise and empirical evidence.

2.2 Single Shot Planning

The goal of the architecture is to grow a tree of paths from the initial start configuration towards the initial goal configuration, and vice versa. To begin, two query pairs are constructed. The first query pair with its start configuration being the initial start configuration and its goal configuration being the initial goal configuration. The second query pair with its start configuration being the initial goal configuration and its goal configuration being the initial start configuration. Next, each of these query pairs is evaluated as described above. From the properties of each individual query pair, a set of best matching algorithms is selected based on the corresponding algorithm characteristics. All the algorithms selected by these two query pairs are then put into a priority queue, with the best matching query pair/algorithm combinations at the top. At this point, the top N algorithms of the priority queue are executed on their respective query pairs. If an algorithm fails to solve its query, it returns any partial path it found and the architecture subsequently builds query pairs containing the last configuration on the returned partial path paired up with a set of "goal" configurations, with the "goal" configurations being a set of configurations in the opposite tree. The process now repeats, returning to the top of the architecture. This continues until a predefined number of iterations is reached, or a path is found connecting the initial start and goal trees. This method of algorithm handoff allows cooperation among algorithms in solving the original query. This process of reevaluation allows for another algorithm set to be selected in response to any change in the local environment after the previous round of algorithms has been given a chance at the query.

3 Single Shot Algorithm Bank

Ideally, the algorithm bank should contain at least one method that is good at planning in any possible situation. Note that we do not expect any of the algorithms to be good in all situations. Instead, our goal is to include a diverse set of algorithms in the bank so that at least one of them will perform well in the current situation. Thus, it is not the individual planners, but their execution in concert that gives our approach its diversity and power.

In this section we briefly describe the individual single shot algorithms available in our prototype implementation.

3.1 Simple ‘Local Planning’ Methods

The first algorithms in our algorithm bank are methods that are commonly used in PRMS as local planners. That is, they are relatively simple, deterministic methods that are fast but not very powerful. Thus, these planners are best used when C-space is uncluttered, although the A^* -like methods can be used in slightly more crowded situations.

The particular methods currently in our bank are:

- straight-line in C-space
- rotate-at-s planners (for rigid bodies)
- simple A^* -like planners

All methods are described in detail in [2]. Briefly, the rotate-at-s planner first translates to an intermediate configuration c' , then rotates to the goal's orientation, and then translates to the goal; the parameter s represents the fractional part of the translational distance between the start and the goal that the robot first translates. The A^* -like planners generate a limited number of candidate neighbor nodes and then select among them in a deterministic fashion.

3.2 Iterative Translation Method (ITM) and Iterative Rotation Method (IRM)

The objective of the ITM method is to get as far as possible from the start configuration using only translation. An outline of the method is presented in Figure 2.

```
ITM(start,goal){
  Add start to Path;
  previous = start;
  prevDir = FindRandomDirection();
  For i = 1 to iterations {
    Try to use prevDir to find a new cfg newCfg
    from previous that is farthes from start;
    If that is not possible try to find a new
    random direction such that the new cfg newCfg
    from previous is free and farthest from start;
    Add newCfg to Path;
    previous = newCfg;
    If it is not possible to find a random
    direction such that the new cfg from previous
    is free,
    then return Path;
  }
  return Path;
}
```

Figure 2: Iterative Translation Method (ITM).

The IRM method is completely analogous to ITM, with the only difference being IRM tries to get far from the start configuration using only rotations.

3.3 Random Walk Method (RWM)

This method performs a random walk from a given start configuration.

An outline of the method is presented in Figure 3.

```
RWM(start,goal){
  Add start to Path;
  previous = start;
  For i = 1 to iterations {
    Find a random free cfg newCfg
    close to previous;
    Add newCfg to Path;
    previous = newCfg;
    If it is not possible to find a random
    free cfg close to previous
    then return Path;
  }
  return Path;
}
```

Figure 3: Random Walk Method (RWM).

3.4 First Intersection Method (FIM)

The idea behind FIM is to try to go directly from the start configuration to the goal configuration. If that is not possible because of collision with an obstacle, then this method tries to go around the obstacle.

An outline of the method is presented in Figure 4, and an example of FIM can be seen in Figure 5.

```
FIM(start,goal){
  If it is possible to go directly from start
  to goal using a straightline local planner,
  then return the path from start to goal;
  else
    Try to go directly from start to goal
    using a straightline local planner, and
    Find the first obstacle in collision
    obstCollision;
    Continue until there is no collision with
    that obstacle;
    Save the last free cfg beforeCfg before
    collide with obstCollision;
    Save the first free cfg afterCfg after
    collide with obstCollision;
    firstPath = path from start to beforeCfg;
    Generate free cfgs around obstCollision;
    Connect those cfgs to construct a roadmap;
    If it is possible to connect beforeCfg to
    afterCfg using that roadmap
      secondPath = path from beforeCfg to goal
      Path = firstPath + secondPath;
      return Path;
    else
      return firstPath;
}
```

Figure 4: First Intersection Method (FIM).

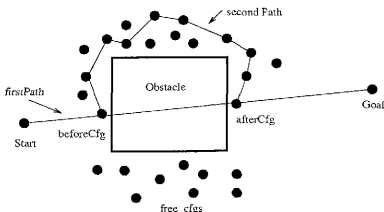


Figure 5: Example of First Intersection Method (FIM).

3.5 Iterative Spread Method (ISM)

A more specialized planner included in the algorithm bank is the iterative spread method (ISM). The ISM planner was designed to operate in cluttered C-Space, and in particular in narrow C-space corridors where large movements are not possible. Intuitively, it is good at ‘feeling’ its way along the corridor, and spreading out of corridor openings. These characteristics make the ISM a good candidate to be used when another algorithm in the bank finds itself at the entrance, or in the interior, of a narrow passage in C-Space.

The general idea of ISM is to ‘grow’ or ‘spread’ a component of connected configurations away from the start configuration, and to do so towards the goal configuration when possible. An outline of the method is presented in Figure 6.

```
ISM(start,goal) {
  1.fringe = {start}. FRINGE = {start,goal}
  for i=1 to nIters {
    fringeSpread = null
    for each fcfg in fringe
      generate n_s configs close to fcfg
      and add them to fringeSpread
    fringe' = n_fs configs in fringeSpread
      that are farthest from the start
    fringe = n_cg configs in fringe'
      that are closest to the goal
    FRINGE = FRINGE + fringe
  }
  2.try to connect start & goal using configs
    in FRINGE as intermediate nodes
}
```

Figure 6: Iterative Spread Method (ISM).

At the beginning of each iteration, a set of **fringe** configurations of the current component is known. For each **fcfg** in **fringe**, we generate uniformly at random **n_s** nearby configurations (so we can likely connect them using a simple local planner such as those mentioned in Section 3.1). The **fringe** set for the next iteration is a subset of the current iterations **fringeSpread** set; it is first biased towards configurations far from the start, and then towards configurations close to the goal. (Distances are computed based on some C-space metric [2].)

After the iterative process, a path from the start to the goal is sought which uses intermediate configurations from all previous **fringe** sets. This is implemented by passing all **fringe** configurations and the start and goal to the connection phase of our OBPRM probabilistic roadmap planning algorithm [3].

3.6 Recursive Midpoint Method (RMM)

Another single shot motion planning method included in the algorithm bank is the recursive midpoint method (RMM). The ideal operation of RMM is in un-cluttered C-Space requiring navigation around obstacles and general free space navigation. RMM will also help identify problem areas in C-Space such as narrow corridors. This method will possibly find the entrance and exit to the corridor, but will not be able to navigate through it. These characteristics make RMM ideal for our proposed single shot architecture. It can get a path started and find where the regions requiring an algorithm with a specific strength are needed, and hand control over to them at this point.

RMM effectively breaks down the initial motion planning query into recursively smaller queries until each sub-query results in a direct collision free connection. The final path is then only the traversal of each sub-path in the appropriate sequence. An outline of RMM is given in Figure 7 and an example of RMM can be seen in Figure 8.

```
RMM(start,goal) {
  if (CanConnect(start,goal))
    then add edge (start,goal) to path
    return
  else
    mpt = midpoint(start,goal)
    if (InCollision(mpt,obstacles))
      then move mpt 'away' until no collision
    if (CanConnect(start,mpt))
      then add edge (start,mpt) to path
      else RMM(start,mpt)
    if (CanConnect(mpt,goal))
      then add edge (mpt,goal) to path
      else RMM(mpt,goal)
}
```

Figure 7: Recursive Midpoint Method (RMM).

The basic idea is that the midpoint *mpt* of the current *start* and *goal* configurations will serve as a subgoal in the query. If the *mpt* configuration is in collision with the obstacles in the environment, then RMM will try to transform the *mpt* to a free configuration and then use that as the subgoal. This transformation can be done in a number of ways. One simple strategy is to test configurations on a line in C-space that passes by the *mpt* and is orthogonal to the line between the *start* and *goal*.

To increase the likelihood that connections can be made, RMM actually works with 'clouds' of configurations for the *start*, *goal*, and *mpt*. These clouds are sets of configurations close to the configuration of

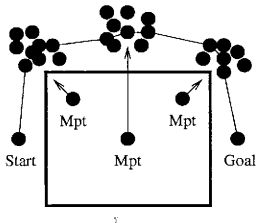


Figure 8: Example of Recursive Midpoint Method (RMM).

interest and are generated in the same manner as the `fringeSpread` configurations in `ISM`. Note that all configurations in a cloud can be connected together, either directly or indirectly. Thus, the `CanConnect` tests in `RMM` simply test if any configuration in one cloud can be connected with any configuration in the other cloud.

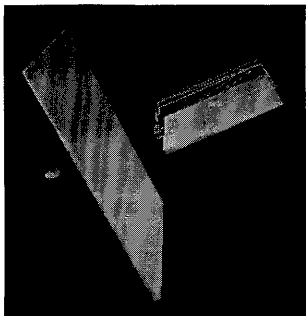
4 Experimental Results

In this section we examine the performance of our single shot planning framework. We first consider queries in some relatively simple environments whose successful solution requires the path to pass through different types of regions. We show that our framework solves these queries by adaptively selecting an algorithm to execute which is suitable for the current situation. We also consider some queries arising in some real problems involving complex CAD models of mechanical designs.

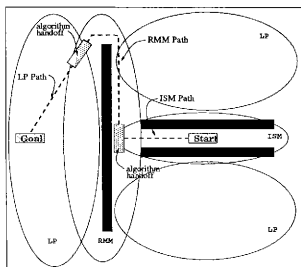
4.1 Artificial Environments

Environment 1. Our first environment consisted of three blocks, two parallel to each other, and the third perpendicular to the other two. The robot is a smaller block that barely fits in the corridor between the parallel blocks. (See Figure 9(a).)

Although this environment is very simple, it contains regions in which different planning methods are



(a)



(b)

Figure 9: Test Environment 1: (a) three-dimensional snapshot, (b) schematic of algorithm regions and path.

ENVIRONMENT 1 RESULTS			
<i>Method</i>	<i>Nodes Generated</i>	<i>Execution Time (sec)</i>	<i>Solved Query</i>
OBPRM	745	28072	YES
SingleShot	309	597	YES

Table 1: Environment 1 Results

needed. In particular, it has a narrow corridor which requires delicate maneuvering (e.g., ISM, ITM, IRM, RWM), regions in which more global navigation is needed (e.g., RMM, FIM), and relatively free regions (e.g., the local planners). These regions are depicted with ovals in Figure 9(b).

The query we consider has the **start** configuration in the corridor between the parallel blocks and the **goal** configuration on the far side of the perpendicular block. This query involves several substeps to successfully find a path, see Figure 9(b) for a schematic representation of a path found by our architecture.

This example shows that our architecture has the ability to adaptively select an algorithm suitable for the current situation. Moreover, it does so in a relatively efficient manner. As can be seen from the results shown

ENVIRONMENT 2 RESULTS			
Method	Nodes Generated	Execution Time (sec)	Solved Query
OBPRM	670	39569	NO
SingleShot	557	326	YES

Table 2: Environment 2 Results

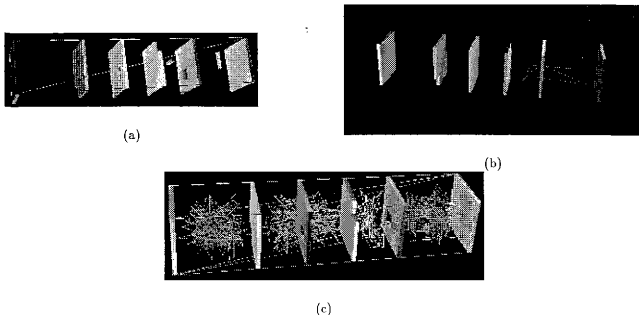


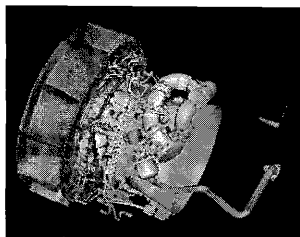
Figure 11: Walls Environment: (a) Environment (b) Path generated by SingleShot (c) Roadmap generated by PRM

197 free configurations, and then ISM ran for 240 seconds and generated 360 free configurations. As can be seen, the OBPRM planner generated 670 nodes and ran for 39569 seconds, and still hadn't solved this query.

Walls Environment. The walls environment, as seen in Figure 11(a), consists of a series of 4 parallel walls, each with only a small hole cut out, creating a series of narrow corridors. For this environment, our query only requires the robot (the long, thin stick) to traverse through one of these narrow openings. The robot positions in Figure 11(a) represent the start and goal configurations. In this case, the singleshot method is more efficient because it only considers the part of the environment necessary for the query; thus, ignoring obstacles irrelevant to the query. Roadmap methods, such as the PRM or OBPRM methods,



(a)



(b)

Figure 12: CAD Model: (a) Flange (b) Part Removal

RESULTS FROM WALLS ENVIRONMENT			
Method	Nodes Generated	Execution Time (sec)	Solved Query
PRM	592	2311.59	NO
OBPRM	199	5005	YES
SingleShot	352	508	YES

Table 3: Results from Walls Environment

construct a roadmap spanning the entire environment, regardless of the query. Notice the drastic difference in the single path generated by Singleshot, as shown in Figure 11(b), and the roadmap generated by the PRM, as shown in Figure 11(c). It can be seen from the roadmap graphs and the walls environment data in Table 3 that not only can our singleshot method solve the query, but it can solve the narrow corridor query significantly faster than the PRM and OBPRM methods. Note, the number of nodes generated by singleshot represents the sum total number of nodes generated by each individual method; however, only the nodes needed for the path are stored. Although the number of nodes generated by singleshot was 352, the number of nodes retained by the singleshot architecture was only 8.

4.2 Complex CAD models

An important design criteria for complex mechanical systems is that certain parts are easy to service and/or replace. Motion planning algorithms for testing such conditions automatically during the design process would be a great aid to engineers, and have been the subject of previous work [7]. Such problems are ideal for single shot methods since one typically only needs to solve the problem once in a given environment.

The Flange. The flange environment consists of two objects; the flange and the elbow. The flange is a fairly flat object with a hole in the center, it is formed of 974 vertices and 990 triangles. The elbow is an "L" shaped object with one extreme wider than the rest of the object, it is formed of 3525 vertices and 5306 triangles.

The start configuration corresponds to the elbow completely inside the flange (see Figure 12(a)), and the goal configuration corresponds to the elbow outside the flange, so basically the problem is to remove the elbow from the flange. Note that this problem is one with a very long and narrow corridor in C-space, requiring a lengthy series of very small translations and rotations of the elbow to successfully remove it from the flange.

Our singleshoot architecture solved a version of the original problem with the elbow scaled down to .98 of its original size. To solve the problem (i.e., to find the path from the start to the goal configuration) the program ran 8 iterations, 7 of them using the ISM method, and the last iteration the program used the rotate-at-s local planner. It took 25165 sec for our singleshoot architecture to solve the problem. Due to the high complexity of this model and the very fine translations and rotations required to remove the elbow from the flange, probabilistic roadmap methods are not suitable for this query.

Engine Part Removal Environment. A problem solved by our method involved a complex design consisting of 15 parts, 14 of which were considered obstacles and the other the 'robot' (see Figure 12(b)). The obstacles consist of many pipes encasing a few larger parts; in total, the models of these parts consist of 126,970 vertices and 269,712 triangles. The robot is another part consisting of some 2887 vertices and 5802 triangles. Originally, the 'robot' part is between the large object and covered by several pipes. The motion planning problem was to find a removal path for the 'robot' part. Again, we note that a general

probabilistic roadmap method is not suitable for such a problem due to the large size of the models involved and the complexity of the C-space, much of which is irrelevant to this particular query.

5 Conclusions and Future Work

This paper proposes a framework for single shot motion planning that adaptively selects a planner whose strengths match the current situation, and then switches to a different planner when circumstances change. We have shown that this strategy can be used to solve challenging problems.

However, much work remains to fully develop the framework. For example, more diverse algorithms need to be added to the algorithm bank which can handle specialized situations (e.g., backing out of dead end, turning a sharp corner, etc.). Also, more work is needed in adding additional nodes to our decision tree to allow more precise matching of query pairs to algorithms.

Finally, even in our limited prototype system we have found it a very challenging task to select appropriate parameters for the framework (e.g., number of algorithms to run, how long to run them) and for the individual algorithms in the bank (e.g., iterations to run, how many configurations to generate). Thus, if we hope to exploit the full power of the framework, it is imperative that we develop automatic parameter tuning methods. When applied to algorithms in the bank, such methods would lead to self-adaptive algorithms.

Acknowledgement

I would like to thank Nancy Amato, Daniel Vallejo, and the rest of the robotics group at Texas A&M. The collision detection library used in our code was Pat Xavier's C-Space Toolkit [19]. GE provided us with CAD models and *Product Vision*, their CAD animation package.

References

- [1] J. M. Ahuactzin and K. Gupta. A motion planning based approach for inverse kinematics of redundant robots: The kinematic roadmap. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 3609–3614, 1997.

- [2] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 630–637, 1998.
- [3] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 155–168, 1998.
- [4] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 113–120, Minneapolis, MN, April 1996.
- [5] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. Robot. Res.*, 10(6):628–649, 1991.
- [6] D. J. Challou, M. Gini, and V. Kumar. Parallel search algorithms for robot motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 2, pages 46–51, 1993.
- [7] H. Chang and T. Y. Li. Assembly maintainability study with motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1012–1019, 1995.
- [8] B. R. Donald. A search algorithm for motion planning with six degrees of freedom. *Artif. Intell.*, 31(3):295–353, 1987.
- [9] B. Glavina. Solving findpath by combination of directed and randomized search. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1718–1723, 1990.
- [10] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2719–2726, 1997.
- [11] Y. K. Hwang and N. Ahuja. Gross motion planning – a survey. *ACM Computing Surveys*, 24(3):219–291, 1992.
- [12] L. Kavraki and J. C. Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2138–2145, 1994.
- [13] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.
- [14] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.
- [15] M. Overmars. A random approach to path planning. Technical Report RUU-CS-92-32, Computer Science, Utrecht University, The Netherlands, 1992.

- [16] M. Overmars and P. Svestka. A probabilistic learning approach to motion planning. In *Proc. Workshop on Algorithmic Foundations of Robotics*, pages 19–37, 1994.
- [17] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1992.
- [18] J. Reif. Complexity of the piano mover's problem and generalizations. In *Proc. IEEE Symp. Foundations of Computer Science (FOCS)*, pages 421–427, 1979.
- [19] P. G. Xavier and R. A. LaFarge. A configuration space toolkit for automated spatial reasoning: Technical results and ldrd project final report. Technical Report SAND97-0366, Sandia National Laboratories, 1997.