2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Single-query, Bi-directional, Lazy Roadmap Planner Applied to Car-like Robots

Stephen Balakirsky, *Senior Member, IEEE*, and Denis Dimitrov

***Abstract*—The Single-query, Bi-directional, Lazy roadmap (SBL) algorithm successfully builds upon the traditional Probabilistic Roadmaps (PRM) approach by introducing a number of related optimizations. While these optimizations are applicable in the domain of car-like robots, the non-holonomic constraints of these systems and non-independence of several of their degrees of freedom pose challenges that will be examined in this paper. We present several enhancements that improve the quality of the generated path in comparison with the simple adaptation of the SBL algorithm. Results demonstrate that this work provides a planner that quickly and reliably discovers efficient paths for car-like robots.**

I. Introduction

More than a decade ago, Probabilistic Roadmap Planners (*PRM*) were introduced as a computationally feasible alternative to complete path planning algorithms [5, 8], especially in environments with many degrees of freedom (*dof*). The idea behind this approach is that through random sampling of the configuration space, it is possible to build a roadmap with sufficient coverage for answering subsequent path planning queries.

Since then, optimizations and modifications have been suggested. In particular, the single-query, bi-directional, lazy roadmap (SBL) algorithm proposed by Latombe was shown to reduce the running time by a factor of 4 to 40 in comparison with traditional PRM implementations [2].

The majority of this research introduces innovations in the context of holonomic robots such as robot arms or free-flying rigid bodies. While the PRM approach has been successfully applied to solving the path-planning problem for car-like robots [6], to the best of our knowledge this work is the first attempt at adapting SBL and demonstrating that the relative merits of the algorithm are applicable in this domain. This paper demonstrates that the algorithm reliably solves the path planning problem in several different scenes, while producing paths of superior quality when compared to a simple application of the SBL algorithm. This approach can therefore be used as a planner with a horizon of several seconds in a control hierarchy of a car-like robot.

A. The Classical PRM Method

The path planning problem deals with computing a feasible, collision free motion for an object (robot) from a given start position q_{init} to a given goal position q_{goal} in a workspace with obstacles. Many path planning algorithms map the workspace into *configuration space* C (the space of all possible placements of the object), with each configuration space dimension corresponding to one of the degrees of freedom in the workspace [1].

The computational cost of creating an explicit representation of the collision-free subset of the configuration space (*free space*, or C_{Free}) is prohibitive in practice for all but the simplest cases, i.e. low-dimensional C [3]. Instead, the *Probabilistic Roadmap Planner* (PRM) approach probes the configuration space at random. The collision free points (*milestones*) are added as nodes to a *probabilistic roadmap* graph. Pairs of promising, nearby milestones are connected using a simple and fast local planner, typically by straight segments in C, and the collision-free paths (*local paths*) form the edges of the roadmap graph. As the number of milestones increases, the roadmap will eventually provide a sufficient representation of the connectivity of the free space. This completes the roadmap *construction phase*.

During the *query phase*, the roadmap is used to answer multiple motion planning queries. For each query, the initial and goal configurations q_{init} and q_{goal} are connected to the roadmap graph. If the connection succeeds, the problem is reduced to finding a path in the graph which connects the two configurations.

B. The SBL Algorithm

PRM is efficient, easy to implement, and applicable in many domains [1]. It has found application in areas such as animation, CAD, virtual environments, computer games, and computational chemistry [7]. It became a popular research topic, with many suggested modifications (especially in the three major areas of sampling, node adding, and collision checking strategies) aimed at improving the success rate and efficiency of the approach. A number of these techniques are reviewed and compared in [1].

The SBL algorithm, developed by Latombe et. al [2], achieved a significant performance improvement through a combination of techniques in all of the three mentioned areas. In particular, it utilizes the following optimizations:

-- **Single-query, bi-directional**: SBL explores as little space as necessary to answer a specific single query, rather than generating a roadmap distributed over the entire free space for answering multiple different queries. The roadmap

Manuscript received September 15, 2009.

Stephen Balakirsky is with the National Institute of Standards and Technology, Gaithersburg, MD 20899 USA (phone: (301) 975-4791; fax: (301) 990-9688; e-mail: stephen@nist.gov).

Denis Dimitrov is with Georgetown University, Washington, DC 20057 USA. (e-mail: dd322@georgetown.edu).

is represented by two trees rooted at the query configurations and expanded concurrently.

-- **Lazy collision checking**: binary collision tests along the roadmap edges are performed to validate only the candidate path between the query configurations, which is found when the two trees are connected. No time is wasted on unnecessary checking. As soon as a collision is detected, the failed edge is removed from the roadmap.

-- **Adaptive sampling**: SBL takes shorter steps in cluttered areas than in open spaces. More specifically, when generating a new milestone for expanding one of the trees, the candidates are generated from successively smaller neighborhoods of a pre-selected milestone q. The process continues until a candidate is tested collision-free, at which point it is added as a child node of q. The neighborhood of q is defined as a set of all milestones, for which the distance between them and q over some pre-defined metrics d is less than a specified threshold (radius).

II. Theoretical Implications

A. Car-like Robots

SBL, assumes that the object, such as a robot arm, can move independently along each of its dofs. As a result, any two milestones can be connected by simple linear interpolation in each dof, i.e. a straight segment in configuration space. Since the range of values of each dimension is normalized in SBL (between [0, 1]), the length of this straight segment can be used as the distance between its endpoint milestones. In the n-D unit cube of the configuration space, the n-sphere with center q and radius r geometrically represents the neighborhood of q.

A car-like robot in a 2 dimensional world typically has 2 position and 1 orientation degrees of freedom – (x, y, Θ). With only 2 control degrees of freedom, acceleration and steering angle, the robot is non-holonomically constrained. It can move only forwards or backwards. In addition, the path curvature is constrained by the robot's minimum turning radius r_{min}.

B. Implications and Design Decisions

1) Connections between milestones: any two milestones can be connected by a series of several arcs. However, especially with nearby milestones (due to the turning radius constraint), the resulting path is likely to be impractical in terms of frequent changes of steering angle and driving direction. Moreover, if one chooses to define additional dimensions, such as speed range, the connection will be even more problematic.

Based on these considerations, it was decided to limit the connections between milestones to a single constant curvature arc. In turn, such a decision restricts the milestone's neighborhood to a relatively small subset of the geometric region to the front and rear of the robot, spatially bounded by the two outgoing arcs of r_{min} radius and whose heading angle Θ is within a certain tolerance of the angle of the tangent of the arc connecting the two milestones. A combination of an arc and a straight segment was also considered, but eventually deemed unnecessary.

2) Choice of dofs: even without the simplification of using a single arc between two milestones, a car-like robot cannot follow an arbitrary path. For example, as it travels along an arc, its positional dofs (x, y) change non-linearly and non-independently ($x^2 + y^2 = r^2$). Similarly, the orientation cannot be changed independently of the robot's position. Unlike holonomic robots, car-like robots cannot compensate for changes in one dof through changes in other dofs, and each additional dof further partitions the configuration space and increases the roadmap size. In this context, the configuration space is characterized by weaker connectivity, even between spatially nearby milestones, which neutralizes the benefit of dealing with fewer dofs compared to typical PRM applications.

To avoid the computational overhead resulting from the higher density of milestones and mutual penetration of milestone trees that are required for a successful connection, it is necessary to keep the dofs to the minimum. These concerns lead us to abandon the idea of early creation of speed profiles by adding a "*speed*" dof to each milestone and allowing connections only between milestones with close speed values. Instead, optimizing speed is more efficiently accomplished during post processing.

3) Distance metrics and neighborhoods: since relatively shorter segments have a higher probability of being collision-free, a metric to utilize during tree growing, tree connection, and collision checking is desirable. However, the aforementioned limitations disallow the straightforward definition of distance metrics between a pair of milestones as a simple aggregation of changes across individual dimensions.

Arc length is utilized as a reasonable approximation to the distance metric that would normally be computed by equating to distances in the (x, y) coordinate plane. Distance (absolute difference) between heading angles of two milestones is strongly correlated to the arc length: for each particular arc, the difference in tangential angles of the two milestones increases as they are moved further away from each other. i.e. the spatial dimensions (x, y) are designated as primary, whereas any other dimensions are still accounted for through some sort of heuristics, but excluded from the formal distance metrics.

Finally, since the dofs are not independent and are not used in defining an explicit distance metrics, they need not be normalized. The algorithm can work directly with the workspace, rather than with the configuration space C.

4) Applicability of SBL: despite these considerations, SBL is an appropriate choice for car-like robot path planning. As a single-query model, it is suitable for many applications such as navigating in a "sliding window" over a large map, or planning temporary, shorter paths while performing world exploration. It is also a logical choice to extend milestones iteratively, rather than to attempt connecting milestones that were generated without considering the feasibility of such

connections.

III. Implementation

A. Basic Implementation

The implementation presented in this paper is subsequently referred to as PRM-Ackerman Steering or PRM-AS. The basic PRM-AS implementation follows SBL very closely and uses all of its optimizations (description based on and slightly modified from [2]):

Algorithm PRM-AS (q_{init}, q_{goal})

1. Install q_{Init} and q_{Goal} as roots of T_{Init} and T_{Goal}, accordingly.
2. Repeat s times, where s is the max number of milestones
   a. EXPAND-TREE
   b. Path p = CONNECT-TREES
   c. If (p != nil and IS-COLLISION-FREE(p)) then return p
3. Return FAILURE.

1) EXPAND-TREE randomly selects between T_{Init} and T_{Goal}, from which it randomly picks a milestone q with probability inverse to the density of milestones around it. The procedure randomly generates a milestone m from q's neighborhood, until it finds a collision-free placement to add as a child of q and return. The adaptive sampling strategy is implemented by halving the neighborhood size after each unsuccessful attempt.

PRM-AS differs from SBL in the specifics of generating m from q's neighborhood. Given minimum turning radius r_{min} and a maximum neighborhood size (corresponding to maximum arc length l), PRM-AS first determines the maximum angular deviation β_{max} from the straight path, up to pi/2, that can be achieved. After randomly selecting an angle β_{ran} between $-\beta_{max}$ and $+\beta_{max}$ and a distance d_{ran} between d_{min} and l, PRM-AS generates a constant curvature arc that positions the robot at an angular offset of β_{ran} and a distance of d_{ran} relative to its previous path. The value of d_{min} is calculated to ensure that the generated arc does not violate the constraint on minimum turning radius r_{min}.

2) CONNECT-TREES finds the milestone m' closest to the most recently generated milestone m, in the tree not containing m. If the distance between them is less than a threshold, then m and m' are connected by a bridge w and the path from q_{Init} to q_{Goal} via w is returned.

PRM-AS first prunes the m' candidates to milestones within a certain physical distance of m. It generates an arc from m (using m.x, m.y, m.Θ) to each of the candidates m' (using m'.x, m'.y, but not m'.Θ). There always exists exactly one such arc (or straight segment)[1]. Most of these arcs are rejected for violating the constraints of minimum turning radius r_{min}, the maximum arc length l, or orientation m'.Θ. m'.Θ must be within a given tolerance of the arc's tangent at point (m'.x, m'.y). If any arcs are retained, the shortest of them is used as the bridge w in the returned path; else, the procedure returns nil.

3) IS-COLLISION-FREE checks for collisions in the candidate path found by CONNECT-TREES. A binary testing method is applied to each segment: the middle position (as determined by its arc) is tested first, then the middle positions of each half, and so on, until either a collision is found or tested positions become close enough together for the segment to be considered collision-free[2].

Each segment keeps track of this gap, i.e. the current arc length between its tested positions. This way, it is possible to break the process into steps, at each step testing the segment at twice the resolution of the previous step. Rather than completely testing one path segment before testing another, the segments are tested one step at a time, each time choosing the segment with the largest untested gap. Such process is more likely to reveal a collision earlier. It continues until either all segments on the path are tested collision-free or a collision is found.

In the latter case, the colliding segment u is removed from the roadmap. The roadmap, temporarily joined by the bridge w during the CONNECT-TREES stage, is split again into two trees. This results in all the milestones on the path between u and w, as well as of their children transferring between the original trees.

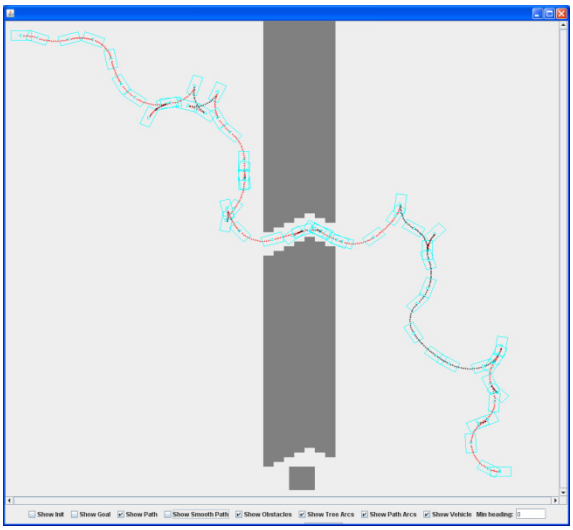

Figure 1: Erratic motion produced by the basic PRM-AS algorithm.

IV. Extensions

Considering that PRM-AS employs the same optimizations as SBL and operates on relatively few dimensions, its performance was not a major concern.

[1] There is also the complementing arc from the same circle, but the algorithm always chooses the shorter of the two.

[2] The motivation is that the middle position is more likely to produce collisions. Experimental results in [1] confirm that in each test scene this method was more efficient than incremental checking.

Instead, the focus of our efforts was on improving the quality of the generated path. Experiments in open test scenes show that the implemented modifications also improve performance in terms of number of generated milestones by approximately 20%. Apparently, the improved version of SBL connects the init and goal trees faster by reducing the number of milestones that blatantly deviate from the optimal path.

As expected from a randomized algorithm, the basic version produces paths such as those shown in Figure 1, characterized by (1) chaotic changes between forward and backward movement; (2) excessive backward movement; and (3) unnecessary deviations from the general driving direction. The extensions, described below, deal with these problems by introducing a probabilistic bias in the otherwise random process of milestone generation, increasing the likelihood of creating a path with the desired characteristics.

A. Bias for consistent driving direction

To prevent frequent switching between forward and backward motion, when extending a milestone q, PRM-AS with higher probability[3] generates either a forward or a backward arc according to the direction of q, and assigns the corresponding direction to the child milestone m.

Driving direction takes one of the two possible values, "forward" and "backward". It is added as a pseudo-dof of each milestone, in that it is not an explicit part of the distance metrics and does not preclude connecting milestones with opposite directions – the connectivity is determined solely on the basis of their position and orientation - but still, it in effect influences the shape of a milestone's neighborhood in the tree expansion phase. It is a compromise from adding a full "speed" dimension, as described earlier.

Further refinements can be made by incorporating "smarter" change-of-direction decisions, based on analyzing the surroundings. A simple example would be to switch directions when facing an obstacle, evident after a number of unsuccessful attempts to generate a collision-free milestone in progressively smaller neighborhoods in the current direction.

B. Bias for forward movement

Because the basic algorithm does not discriminate between forward and backward movement, it generates undesirable backward sections with the same probability as the preferred forward sections. The modified version favors forward movement when expanding the init tree and backward movement when expanding the goal tree. The reason for the latter is that the backward movement in the goal tree represents forward movement in the final path.

These two corrections work in conjunction with each other. For example, bias factors of 10 and 3 respectively mean that at each milestone, the robot maintains the current direction with 9:1 probability. If this direction is undesired, there is a 3:1 chance of switching it; else, forward bias is irrelevant.

C. Bias for straight paths

Depending on configuration, in particular with smaller ratios of minimum turning radius r_{min} to maximum arc length l, the basic version produces widely wavering paths, including occasional loops. As described earlier, during milestone generation it determines the maximum angular deviation β_{max} from the straight path, up to pi/2, that can be achieved given r_{min} and l.

The bias is introduced by reducing β_{max} proportionally to the ratio of the maximum arc length for the current attempt to the maximum allowable arc length. In other words, PRM-AS is restricted to straighter arcs in longer segments, which are possible only in open areas, but is allowed to deviate progressively further from the straight course in tighter spaces, where it is necessary to avoid obstacles.

The combination of these three configurable parameters provides a significant degree of control over the generated path. It is worth noting that caution should be exercised in determining their appropriate values. Because each of them distorts the probabilistic distribution of milestones, setting them too high may lead to a delay or even failure in finding a solution. For example, strong preference for forward motion complicates maneuvering in tight spaces, whereas excessive restriction on turns effectively channels the generated milestones along several major "highways" without providing coverage of the entire configuration space.

Unlike the probabilistic bias approach, the following two extensions involve path post-processing in some form, rather than affecting milestone generation:

D. Path smoothing

This technique, finding direct segments between selected pairs of milestones on the discovered path, is used in most PRM planners. Due to the specifics of connecting milestones by constant curvature arcs, bypassing milestones does not necessarily result in more efficient paths, as is the case with straight segments in C. Therefore, PRM-AS evaluates the local improvement on a case by case basis, while attempting to minimize the total length of the path.

In the test scenes, representative of the actual environments where PRM-AS is expected to be used, the number of milestones on the discovered paths is relatively small – often less than 200. Therefore, despite the quadratic running time, the implemented path smoother is able to test each milestone for a potential connection with each of the subsequent milestones on the path. Most of these pairs are quickly rejected for lack of possible connecting arc; the remaining pairs are used for finding the shortest path, applying the familiar binary approach to test the candidate arcs for collision.

Path smoothing efficiently remedies winding paths, to some extent improves erratic forward/backward driving, and

[3] This probability bias is a configurable parameter that works in conjunction with the forward movement bias. An example is provided in subsection B.

has almost no effect on avoiding backward motion. It is best used in combination with techniques that tackle each of these issues.

E. Discovering and comparing multiple paths

Experiments were performed with a modification that required the algorithm to generate a pre-determined number of solutions, usually between 100 and 1000, and return the best one, as opposed to reporting the first discovered path. The criterion for comparing path quality was the path's total length. We speculated that after finding the first solution, most of the remaining paths would be produced along this first path, resulting in a negligible performance hit.

While this appears to be mostly true, the modification did not provide a significant improvement in path quality. The probable explanation is that all of the paths under consideration suffer with a varying degree from the same issues, and that the shorter, less deviant paths are likely to be discovered earlier anyway.

V. Experimental Results

PRM-AS is written in Java 1.6 and uses an open source Java implementation of VClip collision checker, which is described as "a robust algorithm for computing distances and checking for collisions between convex polyhedra and collections of convex polyhedra" [4]. PRM-AS comes with a custom arc generator which provides the flexibility of incorporating any necessary modifications.

The implementation was put to a test in several scenes typical for benchmarking probabilistic roadmap planners: passage (hole), clutter, and corridor (maze). The environments are configured to match the intended application of the algorithm as a short-term planner with a time horizon of several seconds. Parameters such as vehicle dimensions and turning radius correspond to the parameters of a small car, assuming meters as environment units (see Table 1). Scene size is chosen in such a way that travelling between the start and goal configurations requires at least several seconds, i.e. as much time as it is necessary for the algorithm to plan the next path.

Table 1: Configuration used to test both the basic and optimized version of PRM-AS.

World dimensions	100 x 100
Vehicle dimensions	4 x 2
Vehicle r_{min}	5
A_{Init} and A_{Goal} dimensions	20 x 20
Max arc length l	7
Factor for consistent driving direction (*)	10
Factor for forward movement bias (*)	3
Factor for straight bias (*)	0.4

(*) when enabled

Figures 2, 3, and 4 present typical (smoothed) paths, found by PRM-AS in each scene. The black and red lines represent the forward and backward path arcs, respectively, while the rectangle represents milestones, i.e. locations of vehicle heading changes. The vehicle direction is depicted by the arrow contained in the vehicle rectangle, which is drawn to scale relative to the scene dimensions.

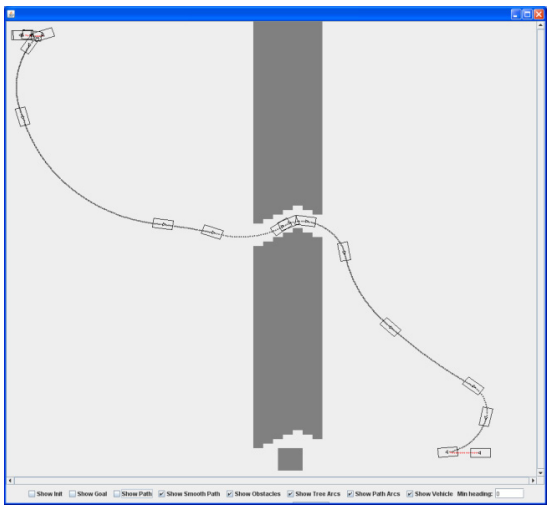

Figure 2: PRM-AS path solution in the "hole" environment (Scene I).

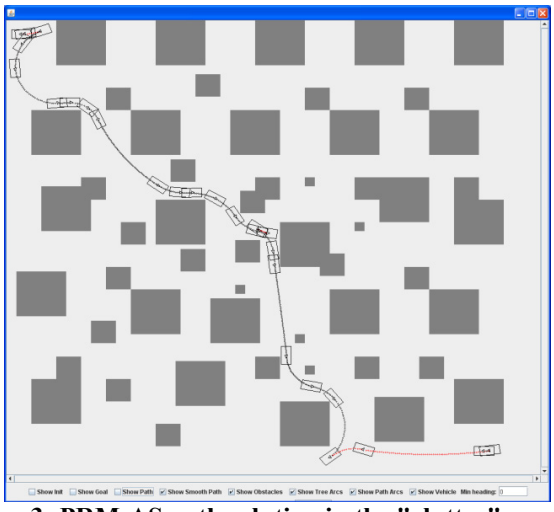

Figure 3: PRM-AS path solution in the "clutter" environment (Scene II).

In all cases it can be seen that unlike the basic planner, the optimized version is able to find realistic, efficient paths. The initial (upper left) and goal (lower right) configuration are intentionally oriented against the general direction of movement, to demonstrate that the robot drives backward only when it is necessary to maneuver from/into these positions or around obstacles. Several such maneuvers are required, especially in the corridor scene, and the planner is able to handle them.

All experiments were performed on a VMWare[5] virtual

[4] See http://people.cs.ubc.ca/~lloyd/java/vclip.html

[5] Certain commercial software and tools are identified in this paper in order to explain our research. Such identification does not imply recommendation or endorsement by the authors, nor does it imply that the software tools identified are necessarily the best available for the purpose.

machine/WinXP with 1 dedicated core and 640 MB of dedicated memory hosted on laptop with Core 2 Duo 2.4 GHz processor/WinXP. Their results are summarized in Tables 2 and 3. Performance may vary significantly depending on the type of scene and the configuration parameters, especially in combination. Either version is able to reliably solve the problem in an order of several seconds[6]. In open scenes, the modified version consistently outperforms the basic algorithm in the number of generated milestones[7]; however, in tight spaces the focus on path quality puts the modified version to a disadvantage.

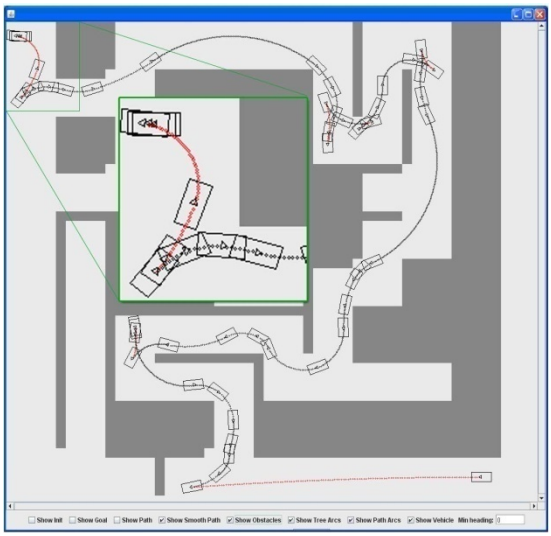

Figure 4: PRM-AS path solution in the "corridor (maze)" environment (Scene III). The magnified area shows the maneuver undertaken to exit the tight space while positioning the vehicle for forward movement.

Table 2: Roadmap statistics (number of milestones) and running times for the basic algorithm. The results are compiled from 100 consecutive runs of each setup.

Scene	Basic version (excl. path smoothing)					
	Num of Milestones			Time, seconds		
	Min	Max	Avg	Min	Max	Avg
Hole	3755	26983	11869	0.19	0.88	0.36
Clutter	2896	10479	6161	0. 24	1.05	0.55
Corridor	11975	27844	18575	1.8	3.98	2.71

It should be noted that for consistency, the same optimization parameters were used across all test scenes. Experiments confirm that tuning the configuration based on the specifics of each scene, significantly improves performance. For example, after turning off straight path bias in the corridor scene, the modified version matches the basic algorithm performance-wise, while still generating better paths. In fact, implementing an adaptive planner, self-configurable depending on observed scene properties, is a promising direction for future work.

Table 3: Roadmap statistics (number of milestones) and running times for the modified algorithm. The results are compiled from 100 consecutive runs of each setup.

Scene	Optimized version (incl. path smoothing)					
	Num of milestones			Time, seconds		
	Min	Max	Avg	Min	Max	Avg
Hole	2716	29608	8338	0.22	1.25	0.47
Clutter	1486	8295	5034	0.31	1.53	0.8
Corridor	12758	52687	27786	3.31	11.8	6.77

VI. Conclusion and Future Work

In this paper, we present an adaptation of the SBL algorithm that solves the planning problem in the context of non-holonomic constraints of car-like robots. We discuss the implications of these constraints and work around them, leading to an implementation that benefits from all the original optimizations of SBL. We then propose several extensions that significantly improve the quality of the generated path, on a par with paths that could be selected by a human operator.

Being a first attempt, this work leaves room for many additional improvements. One prospective direction is experimenting with more complex arcs and consequently, smooth transitions between them. Another area is bridging the gap between the single-query and multiple-query PRM approaches, where the planner could selectively retain parts of the roadmap constructed while answering single queries in the process of world exploration. Finally, an important step in our future efforts is a thorough discussion of how the proposed algorithm satisfies non-holonomic constraints for car-like robots.

[6] In order to demonstrate the planner's capability to handle more complex problems, the scale of the presented configuration is on the longer end of the intended planning horizon. During additional experiments with closer initial placement in the same scenes, running time was also shorter than the expected driving time, confirming that the algorithm is suitable for planning in real time.

[7] The basic version has better running time because path smoothing is not applied.

References

[1] R. Geraerts and M. Overmars, "*A Comparative Study of Probabilistic Roadmap Planners*", in Proc. Workshop on the Algorithmic Foundations of Robotics (WAFR'02), 2002, pp. 43-57.

[2] G. Sanchez and J.-C. Latombe, "*A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking*", in Int. Symp. Robotics Research, 2001, pp. 403-417.

[3] M. Strandberg, "*Robot Path Planning: An Object-Oriented Approach*", 2004, ISBN 91-7283-868-X.

[4] G. Song, N. Amato, "*Randomized Motion Planning for Car-like Robots using C-PRM*" in IEEE Int. Conf. on Intelligent Robots and Systems, 2001, pp. 37-42.

[5] P. Svestka and M. H. Overmars, "*Coordinated motion planning for multiple car-like robots using probabilistic roadmaps*", in IEEE International Conference on Robotics and Automation, volume 2, pages 1631–1636, Nagoya, Japan, May 1995.

[6] P. Svestka and M. Overmars, "*Motion Planning for Car-like Robots using a Probabilistic Learning Approach*", Technical Report UU-CS-1994-33, Dept. Comp. Sci., Utrecht Univ., Utrecht, the Netherlands, 1994

[7] R. Geraerts and M. Overmars, "*Sampling Techniques for Probabilistic Roadmap Planners*", 2003

[8] L. E. Kavraki and J.-C. Latombe, "*Randomized preprocessing of configuration space for fast path planning*", in IEEE International Conference on Robotics and Automation, volume 3, 1994, pp. 2138–2145.