The Corridor Map Method: Real-Time High-Quality Path Planning

Roland Geraerts and Mark H. Overmars

***Abstract*— A central problem in robotics is planning a collision-free path for a moving object in an environment with obstacles. Contemporary applications require a path planner that is fast (to ensure real-time interaction with the environment) and flexible (to avoid local hazards). In addition, paths need to be smooth and short. We propose a new framework, the *Corridor Map Method*, which meets these requirements.**

I. INTRODUCTION

Motion planning is one of the fundamental problems in robotics. The motion planning problem can be defined as finding a path between a start and goal placement of a robot in an environment with obstacles. The past fifteen years, efficient algorithms have been devised to tackle this problem. They are successfully applied in fields such as mobile robots, manipulation planning, CAD systems, virtual environments, protein folding and human robot planning. See the books of Choset *et al.* [3], Latombe [9] and LaValle [10] for an extensive overview.

Many algorithms require that the complete environment is known beforehand. However, in robotics, the environment is often partially unknown. Such environments frequently contain dynamic obstacles that can block a computed path. As a result, sensor information is required to avoid the new obstacles. Even if all information is available, e.g. in static virtual environments, methods can have difficulties dealing with the growing sizes of contemporary virtual environments. Often, only the large obstacles are taken into account to safe memory and to lower the CPU load. However, also the small obstacles have to be avoided in real-time.

An important question is how long the computation of a path may take to ensure real-time behavior. In a virtual environment, such as a game, very little processor time is scheduled for the path planner. Especially when many paths have to be planned simultaneously, only one (of a few) milliseconds per second CPU time per robot is allowed. Larger running times will lead to stalls in interactive environments.

In conclusion, interactive environments require a motion planner that is fast and flexible. Flexible planners, such as *Potential Field* methods, were introduced in the robotics community about 20 years ago [8], [13]. A Potential Field method directs the motion of the robot through an artificial potential field which is defined by a function over the free configuration space $\mathcal{C}_{\text{free}}$ (that is, the space of all possible placements for the robot in the environment). The robot is pulled toward the goal position as it generates a strong attractive force. In contrast, the obstacles generate a repulsive force to keep the robot from colliding with them. The path from the start to the goal can be found by following the direction of the steepest descent of the potential toward the goal. While this method has some flexibility to avoid local hazards (such as small obstacles/other moving objects), it is too costly for path planning in interactive virtual environments. In addition, the path will not always be found because the robot often ends up in a local minimum of the potential.

The *Probabilistic Roadmap Method* (PRM), developed in the nineties [1], [12], does not suffer from the local minima problem. This method consists of two phases. In the construction phase, a roadmap is created that captures the connectivity of $\mathcal{C}_{\text{free}}$ with a set of one-dimensional curves. In the query phase, the start and goal positions are connected to the graph, and the path is obtained by running Dijkstra's shortest path algorithm.

While the PRM has been successfully applied to a broad range of problems, the method generates ugly paths, i.e. the paths are only piecewise linear, they have many redundant motions, and they have little clearance to the obstacles, resulting in unnatural looking motions. While techniques exist for optimizing the paths [4], [5], [11], they are too slow to be applied in the query phase in real-time applications.

By shifting the optimization process to the off-line construction phase, high-quality paths can be computed in a small amount of time. In [6], we proposed a method that creates high-quality graphs from which relatively short paths and paths with a large amount of clearance can be extracted. While the method may be fast enough for an environment with one robot, the method will be still too slow for environments with many robots.

A disadvantage of these roadmap-based methods is that they output a fixed path in response to a query. This leads to predictable motions and lacks flexibility when the environment or robot changes.

Recently, the concept of path planning inside *corridors* has been introduced [7], [14]. By using corridors, the advantages of the techniques described above are combined. That is, global motions are directed by a high-quality roadmap, and local motions are controlled by potential fields inside corridors, providing local flexibility of the path. In [7], corridors have been exploited to find paths for coherent groups of robots. Also quantitative measures for the quality of corridors have been devised [14].

In this paper, we extend and generalize their results by proposing a general framework, called the *Corridor Map Method* (CMM). We show how the framework can be used to avoid dynamic obstacles and to create short paths. Then we conduct experiments with 2D problems and conclude that the framework is capable of creating smooth, short paths for robots avoiding dynamic obstacles in real-time, i.e. in less than 1 ms CPU time per second traversed time of the robot.

Part of this research has been funded by the Dutch BSIK/BRICKS Project.

R. Geraerts and M.H. Overmars are with Institute of Information and Computing Sciences, Utrecht University, 3508 TA Utrecht, the Netherlands. Email: {roland,markov}@cs.uu.nl

II. The Corridor Map Method

The Corridor Map Method (CMM) creates a system of collision-free corridors for the static obstacles in an environment. Paths can be planned inside the corridors for different types of robots while satisfying additional constraints such as avoiding dynamic obstacles. We assume that the robot can be modeled by a ball with radius r.

The CMM consists of an off-line construction phase and an one-line query phase. In the construction phase, a roadmap graph $G = (V, E)$ is built which serves as a skeleton for the corridors (see Fig. 1(a)). Each vertex $\nu \in V$ corresponds to a collision-free point in a D-dimensional environment (D is typically 2 or 3) and each edge $\epsilon \in E$ corresponds to a local path Π_ϵ. The path Π is defined as follows:

Definition 1 (Path). *A path Π for a point in a D-dimensional environment is a continuous map $\Pi \in [0,1] \rightarrow \mathbb{R}^D$ such that $\forall t \in [0,1] : \Pi[t] \in \mathcal{C}_{\text{free}}$.*

With each point $\Pi_\epsilon[t]$ on local path Π_ϵ, we associate the radius $R[t]$ of the largest empty ball (in the environment) centered at $\Pi_\epsilon[t]$. This clearance information and the graph are now used to define the *corridor map* (see Fig. 1(b)):

Definition 2 (Corridor map). *The corridor map is a graph $G = (V, E)$ with clearance information. That is, each edge $\epsilon \in E$ encodes a local path Π_ϵ together with the radii R of the corresponding largest empty balls in the environment.*

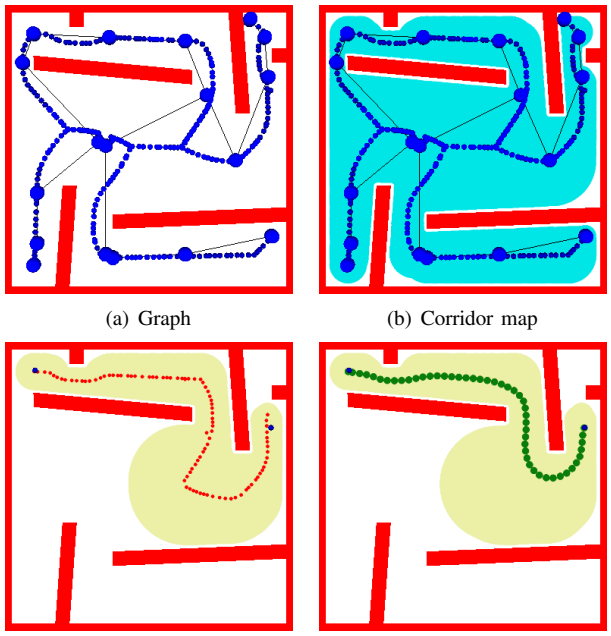
In the query phase, we have to find a path for a robot which connects the start position to the goal position. For now, we assume that these positions are vertices $\nu', \nu'' \in V$. By running Dijkstra's shortest path algorithm (while discarding passages that are too narrow, i.e. edges for which $\exists t : R[t] < r$), we extract the *backbone path* (if one exists) from G.

Definition 3 (Backbone path). *Let $\epsilon_1 \dots \epsilon_n$ be the sequence of edges extracted from G that connects ν' with ν''. The backbone path $B[t]$ is then defined as $\Pi_{\epsilon_1} \oplus \dots \oplus \Pi_{\epsilon_n}$ where the operator $\oplus$ concatenates the local paths Π_{ϵ_i}.*

The backbone path, together with the clearance information defines a *corridor* (see Fig. 1(c)):

Definition 4 (Corridor). *A corridor $C = (B[t], R[t])$ is defined as the union of the set of balls with radii $R[t]$ whose center points lie along its backbone path $B[t]$.*

If the start position s or goal position g is not equal to one of the vertices, we have to extend the corridor such that they are included (see Fig. 2). Let s' and g' be their closest points on local paths Π_{ϵ_s} and Π_{ϵ_g} whose balls include s and g, respectively. Edges ϵ_s and ϵ_g are split such that they include vertices s' and g', respectively. Finally, let Π_s be the straight-line local path between s and s' and Π_g be the straight-line local path between g' to g. Then, the backbone path is defined as path Π_s, concatenated with the shortest path between s' and g', and path Π_g. The radii corresponding to the positions p on Π_s (Π_g) are equal to the clearance corresponding to vertex s' (g') minus the Euclidean distance between s' (g') and p.



(a) Graph (b) Corridor map (c) Corridor and backbone path (d) Final path

Fig. 1. The construction phase (top) and the query phase (bottom) of the Corridor Map Method.

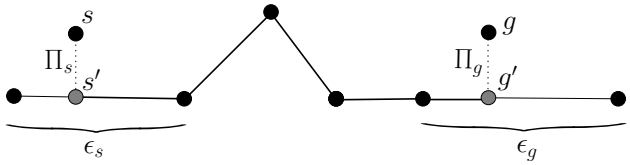


Fig. 2. Extending the corridor to include the start and goal positions.

Now that we have defined the corridor, which guides the global motions of the robot, its local motions are led by an *attraction point*, $\alpha(x)$, moving on the backbone path of the corridor from the start to the goal. The attraction point is defined such that making a step toward this point leads the robot toward the goal. In addition, the ball (with radius $R[t]$) corresponding to $\alpha(x)$ encloses the robot, ensuring a collision-free motion. If $R[t] \leq r$, there exists no attraction point, and, hence, no path.

Definition 5 (Attraction point). *Let x be the current position of the robot with radius r. The attraction point $\alpha(x)$ for the robot at position x is the point $B[t]$ on the backbone path B having the largest time index $t : t \in [0:1]$ such that Euclidean distance $(x, B[t]) < R[t] - r$.*

The attraction point attracts the robot with force $\mathbf{F}_0$. Let d be the Euclidean distance between the robot's position x and the attraction point $\alpha(x)$. Then $\mathbf{F}_0$ is defined as

$$\mathbf{F}_0 = f \frac{\alpha(x) - x}{||\alpha(x) - x||}, \text{where } f = \frac{1}{R[t] - r - d} - \frac{1}{R[t] - r}.$$

The scalar f is chosen such that the force will be 0 when the robot is positioned on the attraction point. In addition, f will be ∞ when the robot touches the boundary of the ball. (However, f will never reach ∞ since we require that the radii of the balls are strictly larger than r.)

Local hazards (such as small obstacles or other robots) can be avoided by adding repulsive forces to $\mathbf{F}_0$ toward the hazards. Hence, the final force $\mathbf{F}$ is dependent on the problem to be solved. We will show some choices in the Section III.

The final path Π is obtained by integration over time while updating the velocity, position and attraction point of the robot. In each iteration, we update the attraction point on the backbone path based on position x of the robot. Now we have all information needed to compute the force $\mathbf{F}$. By integrating $\mathbf{F}$, we compute the new velocity vector for x. In addition, by integrating the velocity vector, we compute the new position for the robot. We continue moving the robot until the robot has reached the goal. By using this time integration scheme, a *smooth* path is obtained.

Theorem 1 (C^1 continuity of the path)**.** *The* CMM *generates a path* Π *that is smooth, i.e.* C^1 *continuous.*

Proof: The path Π is obtained by integrating the force $\mathbf{F}$ two times, which adds two degrees of continuity to the path followed by the attraction point. Even though this path can be discontinuous, it can be easily shown that double integration leads to C^1 continuity. To proof that $\mathbf{F}$ can indeed be integrated, we have to show that the denominators in $\mathbf{F}$ are larger than 0: Since the attraction point $\alpha(x)$ is defined as the furthest point on the backbone path, the point lies always 'in front of' the robot's position x (except for the goal position), and, hence, the term $||\alpha(x) - x|| > 0$. In addition, the term $R[t] - r - d > 0$ because $R[t] > r$. Since these two terms stay positive, $\mathbf{F}$ can be integrated (two times), resulting in a path Π being C^1 continuous. ■

As an example, consider Fig. 1 which shows the stages of the CMM applied to a simple planar environment. For this environment, an input graph was created. Its nodes were sampled on the medial axis to ensure a locally maximum clearance of the nodes. Its edges (black lines) were retracted to the medial axis to provide high-clearance local paths (small discs). The graph, together with the clearance information, forms the corridor map which is displayed in the second picture. The covered area of the map is visualized in a light color. The next picture shows the corridor and its backbone path corresponding to a start and goal position of the query. The final path, displayed in the fourth picture, was obtained by applying the procedure described above.

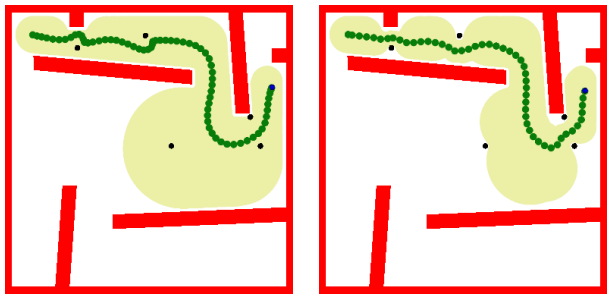
III. Specific choices

An important influence on the quality of the corridor map, and, hence, the quality of the resulting paths, is the quality of the input graph. In [6], we proposed the *enhanced Reachability Roadmap Method*, which is a technique that creates high-quality graphs satisfying the following four properties:

1. The graph is *resolution complete*. This means that a valid query (which consists of a start and a goal position) can always be connected to the graph. If there exists a path between the start and goal, then it can always be found (at a given resolution).
2. The graph is *small*. A small graph assures low query times and low memory consumption. In addition, when a graph must obey other criteria, a small graph eases manual tuning.
3. The graph contains *useful cycles*. These cycles provide short paths and alternative routes which allow for variation in the (global) routes that robots take.
4. The graph provides *high-clearance local paths*. As the local paths lie on the medial axis, each point on the corridor will have a locally maximum clearance. This will provide the most freedom for the robot to move.

In the remainder of this paper, we will use this technique for creating the input graphs.

We have seen that moving inside a corridor (instead of moving along a path) provides enough freedom to obtain a smooth path. Now we will describe how the framework can be used to avoid dynamic obstacles and to create shorter paths.



(a) Extending the force function (b) Updating the corridor

Fig. 3. Two techniques for obstacle avoidance. The left picture shows an unchanged corridor that includes five small obstacles. The right picture shows the updated corridor, swaying around the five obstacles.

A. Avoiding dynamic obstacles

Dynamic obstacles are the obstacles in the environment that are not present (or suppressed) when the corridor map is created. We consider two approaches for avoiding these obstacles: updating the force $\mathbf{F}$ (by adding a repulsive force toward the obstacles), and changing the corridor itself. We assume that the radii of the obstacles are known.

Adding forces: Our goal is to guide the robot around all dynamic obstacles inside the corridor. To ensure that the robot does not collide with the obstacles, repulsive forces are applied. Such a force is only applied if both the robot and the obstacle are located in the ball corresponding to the attraction point $\alpha(x)$. For each obstacle $O_i : i \in [1 : n]$, we compute a repulsive force $\mathbf{F}_i$. Let d_i be the Euclidean distance between the center of the robot at position x and the center of obstacle i with radius r_i, r be the radius of the robot, and $k : k > 0$ be a constant. Then $\mathbf{F}_i$ is defined as

$$\mathbf{F}_i = f\frac{x - O_i}{||x - O_i||}, \text{where } f = \frac{k}{d_i - r_i - r}.$$

The scalar f is chosen such that the force will be ∞ when the robot and obstacle touch. The larger the distance between them, the lower the force will be. The constant k is used to change the influence of the repulsive forces on the robot.

The final force $\mathbf{F}$ can now be calculated by adding the attractive force $\mathbf{F}_0$ and repulsive forces $\mathbf{F}_i$, i.e.

$$\mathbf{F} = \mathbf{F}_0 + \ldots + \mathbf{F}_n.$$

As an example, consider Fig. 3(a). It shows our running example, but now five small dynamic obstacles have been added. The final path is obtained after the force function has been extended. The figure shows that the path has only changed locally. While this method is flexible, it is hard to control the 'shape' of the path. In addition, future changes of the path (i.e. shortening the path) are hard since such change has to operate on a path instead of a corridor. By creating a sub-corridor inside the corridor, excluding the dynamic obstacles, we obtain more freedom.

Creating a sub-corridor: Our goal is to create a sub-corridor $C' = (B'[t], R'[t])$ which lies inside the original corridor and is absent from the dynamic obstacles $O_i : i \in [1:n]$. In the following procedure, we initially set $B'[t]$ and $R'[t]$ to $B[t]$ and $R[t]$, respectively, where $t : t \in [0:1]$ is the time index. Then we move the backbone path and update the corresponding radii of the balls, as follows. Let $d_i[t]$ be the Euclidean distance between the center of the ball positioned at $B[t]$ and the center of obstacle i, i.e. $d_i[t] = ||B[t] - O_i||$. Only if an obstacle O_i is in this ball, i.e. $d_i[t] < R[t]$, the sub-corridor is modified. The point $B[t]$ will be moved away in a straight line from obstacle O_i. The distance traveled by this point equals to

$$dist = \frac{R[t] - d_i[t]}{2}.$$

Hence, the position of ball $B'[t]$ is given by

$$B'[t] = B[t] + dist * \frac{B[t] - O_i}{d_i[t]},$$

and the radius $R'[t]$ of the ball equals to

$$R'[t] = R[t] - dist.$$

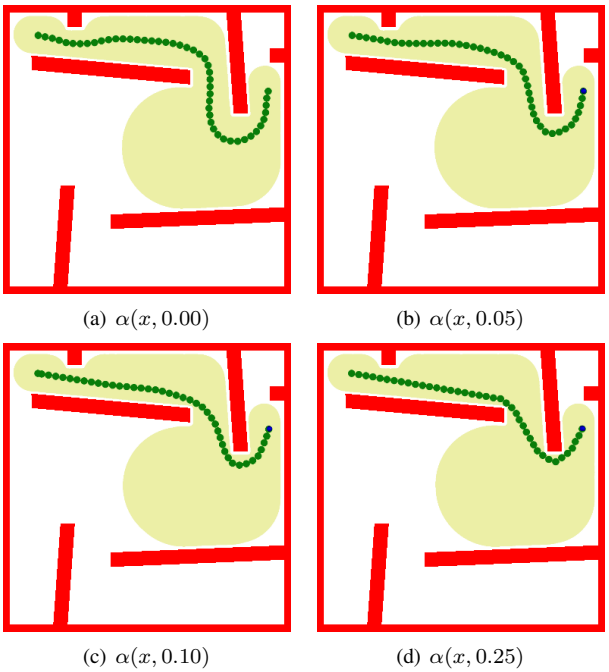
We refer the reader to Fig. 3(b) for an example of the method. The resulting smooth path lies in the updated corridor, being absent of the five dynamic obstacles. As a new corridor has been computed, the dynamic obstacles can be discarded when the path is processed even further, e.g. when the path is shortened.

B. Creating shorter paths

Our goal is to use the corridor to create shorter paths. In our standard framework, the robot at position x is attracted toward the attraction point $\alpha(x)$, corresponding to point $B[t]$. Shortcuts in the path can be made by creating a second *valid* attraction point $\alpha(x, \Delta t)$, corresponding to point $B[t + \Delta t]$: $\Delta t \geq 0 \wedge t + \Delta t \leq 1$, to which the robot is attracted. We say that an attraction point $\alpha(x, \Delta t)$ is *valid* if the robot, moved in a straight-line from its current position x to the attraction point $\alpha(x, \Delta t)$, stays inside the corridor.

Suppose now that we want to create a path using a certain value of Δt. If the attraction point $\alpha(x, \Delta t)$ is not valid, we have to lower Δt. We determine the highest Δt by decreasing Δt with small steps until $\alpha(x, \Delta t)$ is valid. Given Δt, the resulting force $\mathbf{F}_s$, which must be added to force $\mathbf{F}$, equals to

$$\mathbf{F}_s = \frac{\alpha(x, \Delta t) - x}{||\alpha(x, \Delta t) - x||}.$$



(a) $\alpha(x, 0.00)$ (b) $\alpha(x, 0.05)$ (c) $\alpha(x, 0.10)$ (d) $\alpha(x, 0.25)$

Fig. 4. Using the corridor to create shorter paths. Shorter paths are obtained by moving the attraction point $\alpha(x)$ along the backbone path toward to goal.

The influence of using different values for Δt on the path can be examined in Fig. 4. The first picture shows the path obtained by only attracting the robot to its attraction point $\alpha(x)$. The other pictures show the resulting paths for different values for Δt. Indeed, shorter paths are obtained for larger values of Δt at the cost of increased computation time.

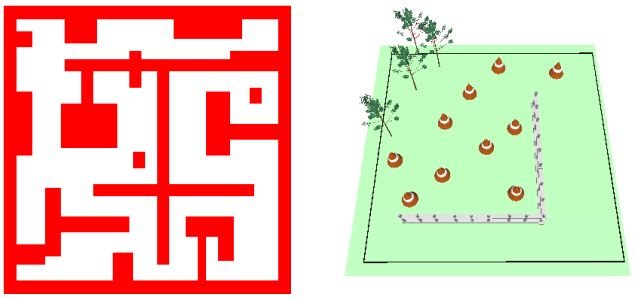
IV. Experiments

In this section, we test the *Corridor Map Method* (CMM) on two different environments. We will experimentally check whether the CMM can produce high-quality paths in real-time, i.e. the CPU load being less than 0.1%.

A. Experimental setup

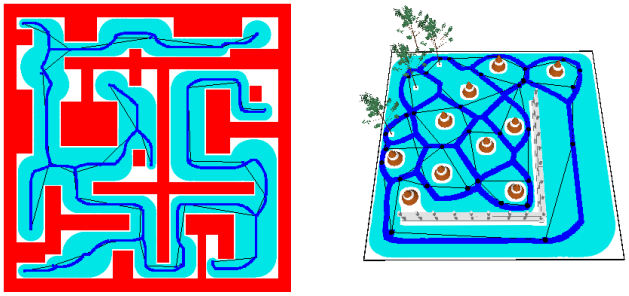
We integrated the techniques in a single motion planning system called SAMPLE (System for Advanced Motion PLanning Experiments), which we implemented in Visual C++ under Windows XP. All experiments were run on a 2.66 GHz Pentium 4 processor with 1 GB memory. We used Solid for collision checking [2].

We conducted experiments with the environments depicted in Fig. 5. The roadmaps, together with their corridor maps are displayed in Fig. 6. Since we focused on obtaining low query times and high-quality paths, much time for creating these graphs could be spent off-line by the *enhanced Reachability Roadmap Method* from [6]. This method discretized the



(a) Maze (b) Field

Fig. 5. The two test environments.



(a) Maze (b) Field

Fig. 6. The input graphs and corresponding corridor maps for two test environments.

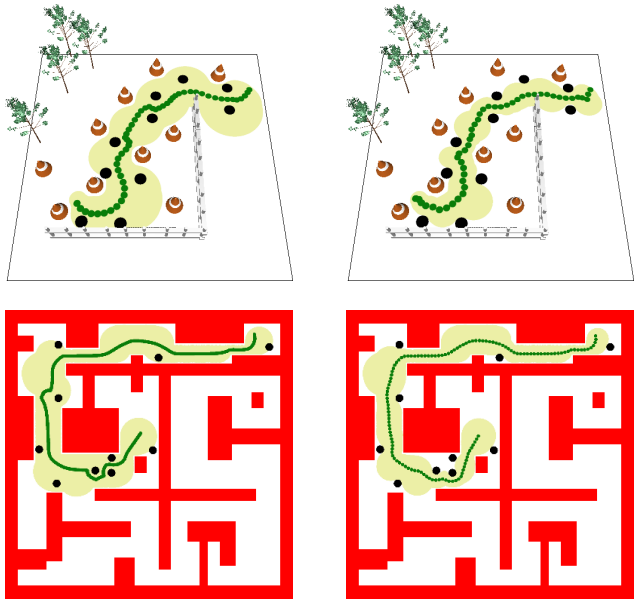
environments with 100x100 cells. The robot is modeled as a small disc. The environments have the following properties:

Maze This 2D environment is a simple maze composed of a small number of polygons. The input graph was created in 1.6 seconds. Since its local paths lie on the medial axis, the corridor map covers a large portion of the free space, providing the robot much freedom to move.

Field This 3D environment contains ten cones, two fences and four trees. Together, they are composed of 16,000 triangles. Consequently, much more time was needed to create the input graph (i.e. 20 seconds). There are many alternative routes. Again, the corridor map covers a large portion of the free space.

We performed three batches of experiments. In the first batch, we found paths for 100 random queries to get an idea of how long a query takes to compute. In the second batch, we defined one query. We added up to 10 dynamic obstacles close to the backbone path and observed the changes in running times. In the third batch, we studied the effect of choosing different values of Δt on the running time versus path length.

For each experiment, we provide the average integral running times (in ms) of the query phase. That is, connecting the query to the roadmap, computing the corridor and extracting the path. Often, only the computation for a part of the path is required/desired. Hence, we also provide the CPU load (which is defined as the CPU time / traversed time $*$ 100%). Note that this measure is rather subjective, i.e. increasing the robot's speed implies a lower CPU load while decreasing its speed implies a higher CPU load. Therefore, we also mention the traversed time.



(a) Extending the force function (b) Updating the corridor

Fig. 7. Obstacle avoidance in the two environments. Ten obstacles are avoided.

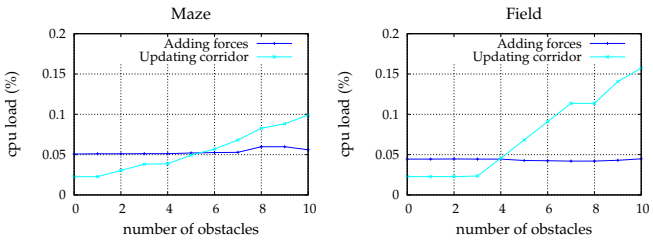


Fig. 8. The performance of the two techniques for avoiding obstacles.

B. Experimental results

1) Measuring the performance of creating smooth paths: For the Maze environment, a query was computed in 2.41 ms on average and was traversed in 9.25 seconds. The average CPU load was 0.026% which means that 0.26 ms CPU time (per second) was required for one second traversed time. For the Field environment, a query was computed in 0.84 ms on average and was traversed in 2.87 seconds. The average CPU load was 0.029%. We can conclude that the method is very efficient in producing smooth paths.

2) Dynamic obstacles: We considered two methods for obstacle avoidance. The first method extended the force function while the second method updated the corridor. Fig. 7 displays the corridors and paths obtained while avoiding ten obstacles. The average query times for the Maze environment ranged between 7.0 and 9.0 ms for the first method, and between 3.0 and 13.6 ms for the second method. The average query times for the Field environment ranged between 2.0 and 2.3 ms, and between 1.0 and 7.0 ms for the two methods, respectively. Fig. 8 shows the results. The figure makes clear that both techniques are efficient. Nevertheless, the technique that adds forces is more efficient when more dynamic obstacles are present.

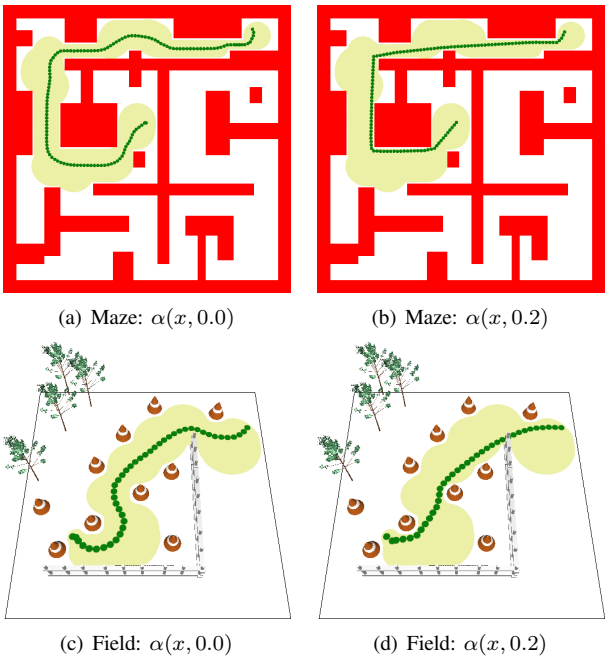
3) Short paths: Shorter paths were created by adding a force toward a new attraction point $\alpha(x, \Delta t)$ placed between $\alpha(x)$ and the goal. Fig. 9(a) and (c) show the paths which have not been shortened, i.e. no additional attraction point was used. Pictures (b) and (c) show the paths obtained by using an additional force attracting the robot toward $\alpha(x, 0.2)$. We performed preliminary experiments to study the relation between Δt and the path length. The results are shown in Fig. 10. We conclude for these curved paths, that short paths can be obtained for small values of Δt, i.e. $\Delta t = 0.2$. However, the running times were reasonably large due to the relative expensive computation of the second attraction point. For $\Delta t = 0.2$ they were about four times as large as the running times corresponding to not using the new attraction point. For $\Delta t = 1.0$ they were about twelve times as large. In conclusion, the technique is fast enough for real-time performance (for small values of Δt). For larger values however, the method will not be fast enough. We are currently investigating how to enhance the technique.

V. Conclusions and Future Work

We presented a new framework, called the *Corridor Map Method* (CMM), which can be used for path planning in real-time interactive applications. The CMM directs the global motions by a high-quality roadmap. Local motions are controlled by potential fields inside a corridor, leading to smooth and short paths. In addition, the corridor provides enough flexibility when dynamic obstacles (or other moving robots) have to be avoided. Experiments showed that such motions can be computed in real-time. As we have not optimized our code yet, we think that the running times can be improved even more.

The input graphs for the CMM were created by our Reachability Roadmap Method. Also other methods could be used to create these graphs. To ensure a high quality of the graphs, reasonably high computation times were required in the construction phase. We think that these can be improved dramatically by incorporating learning techniques. Nevertheless, these graphs ensured fast running times in the query phase.

The CMM can also be used for guiding the motions of a group of robots. In addition, tactical information could be incorporated in the corridor map to provide clever routes for the robots. While we focused on 2D problems, the framework is also applicable to higher-dimensional problems. In future work, we will extend the experiments with 3D problems. In addition, we will study how to select alternative corridors and how to create alternative paths within a corridor.



(a) Maze: $\alpha(x, 0.0)$ (b) Maze: $\alpha(x, 0.2)$

(c) Field: $\alpha(x, 0.0)$ (d) Field: $\alpha(x, 0.2)$

Fig. 9. Using the corridor to create shorter paths. Shorter paths are obtained by moving the attraction point $\alpha(x)$ along the backbone path toward to goal.

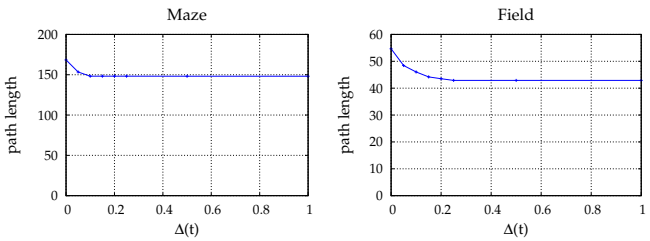


Fig. 10. The relation between the value of Δt and the path length.

References

[1] J. Barraquand, L. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan, "A random sampling scheme for path planning," *International Journal of Robotics Research*, vol. 16, pp. 759–744, 1997.

[2] G. Bergen, *Collision Detection in Interactive 3D Environments*. Morgan Kaufmann, 2003.

[3] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*, 1st ed. MIT Press, 2005.

[4] R. Geraerts and M. Overmars, "Clearance based path optimization for motion planning," in *IEEE International Conference on Robotics and Automation*, 2004, pp. 2386–2392.

[5] ——, "On improving the clearance for robots in high-dimensional configuration spaces," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 4074–4079.

[6] ——, "Creating high-quality roadmaps for motion planning in virtual environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 4355–4361.

[7] A. Kamphuis and M. Overmars, "Finding paths for coherent groups using clearance," in *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, 2004, pp. 19–28.

[8] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *International Journal of Robotics Research*, vol. 5, pp. 90–98, 1986.

[9] J.-C. Latombe, *Robot Motion Planning*. Kluwer, 1991.

[10] S. LaValle, *Planning Algorithms*. http://planning.cs.uiuc.edu, 2006.

[11] D. Nieuwenhuisen, A. Kamphuis, M. Mooijekind, and M. Overmars, "Automatic construction of roadmaps for path planning in games," in *International Conference on Computer Games: Artificial Intelligence, Design and Education*, 2004, pp. 285–292.

[12] M. Overmars, "A random approach to motion planning," Utrecht University, Tech. Rep. RUU-CS-92-32, 1992.

[13] E. Rimon and D. Koditschek, "Exact robot navigation using artificial potential fields," *IEEE Transactions on Robotics and Automation*, vol. 8, pp. 501–518, 1992.

[14] R. Wein, J. Berg, and D. Halperin, "Planning near-optimal corridors amidst obstacles," in *International Workshop on the Algorithmic Foundations of Robotics*, 2006.