

# Occlusion-Free Path Planning with a Probabilistic Roadmap

Matthew A. Baumann, Donna C. Dupuis, Simon Léonard, Elizabeth A. Croft and James Little.

**Abstract**—We present a novel algorithm for path planning that avoids occlusions of a visual target for an “eye-in-hand” sensor on an articulated robot arm. We compute paths using a probabilistic roadmap to avoid collisions between the robot and obstacles, while penalizing trajectories that do not maintain line-of-sight. The system determines the space from which line-of-sight is unimpeded to the target (the visible region) using the method described in [11]. We assign penalties to trajectories within the roadmap proportional to the distance the camera travels while outside the visible region. Using Dijkstra’s algorithm, we compute paths of minimal occlusion (maximal visibility) through the roadmap. In our experiments, we compare a shortest-distance path to the minimal-occlusion path and discuss the impact of the improved visibility.

## I. INTRODUCTION

An important criteria for planning robot motions is to achieve a goal configuration while maintaining a line-of-sight from a sensor to a target. In the case of vision-guided robotics, the goal is to plan a path to a goal that simultaneously allows the sensor to observe a target while avoiding environment obstacles. Sensors such as cameras or laser rangefinders rely on a clear line-of-sight to the target to obtain data. If the robot moves in such a way that an object or structure lies between the sensor and the target, then the sensor can no longer collect data and progress towards the goal is impeded (see Fig. 1). Uninterrupted data collection relies on avoiding occlusions to the line-of-sight.

Formally, the problem considered herein is to compute a series of motions in the robot’s configuration space ( $C$ -space) such that those motions avoid occluding the line-of-sight from an onboard sensor to a target while still reaching a goal configuration. Additionally, we use existing probabilistic roadmap techniques to avoid collisions between the robot and obstacles in the environment. In practice, the algorithm will find a minimum-occlusion path within the probabilistic roadmap, which represents a sampling of possible configurations in the  $C$ -space.

For the purposes of this paper, the configuration space of the robot will be denoted as  $C$ . In our case this is a 6-dimensional space representing the joint angles of an articulated industrial robot arm. The probabilistic roadmap (PRM) consists of an undirected graph where each node denotes a single configuration or point in  $C$ . Each node has been tested for collisions and determined to lie within  $C_{free}$ , the

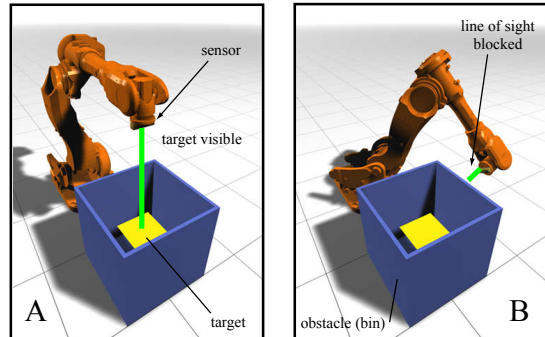


Fig. 1. An example setup: A robot observes objects in a bin using a visual sensor. The sensor has a clear line-of-sight in configuration A, but its line of sight is occluded by the bin wall in configuration B.

collision-free subset of  $C$ . The nodes are linked with edges that represent linear interpolations between configurations. We test the interpolated motions for collisions using the collision-detection algorithm from Schwarzer et al. [8]. If the interpolated motion lies entirely within  $C_{free}$ , then it is collision-free and safe to include in the roadmap. Our goal is to plan paths that remain not only collision-free, but maintain a line-of-sight to the target. The region of space from which an unoccluded line-of-sight exists to the target will be referred to as the “visible region” and denoted  $C_{vis}$ . Like  $C_{free}$ , it is difficult to compute  $C_{vis}$  directly. Instead we will evaluate each edge in the roadmap to assess if it causes the sensor to travel outside of  $C_{vis}$ .

## II. RELATED WORK

Kavraki, Svestka, Latombe & Overmars [5] introduced the probabilistic roadmap (PRM), using a sampling technique to discover a sparse representation of obstacles in the configuration space. This allows efficient computations of collision-free paths.

Our work extends the PRM concept to include visibility as well collision avoidance by adding to the penalties, based on interpolation distance, assigned to edges in the roadmap.

Schwarzer, Saha & Latombe [8] present adaptive dynamic collision checking, where a trajectory in joint space is checked for collisions by adaptively refined sampling. The key computation is the comparison of the lower bound of the distance between the moving objects and the upper bound of the arc length traced by each object. If the moving object’s maximum arc length (for a given trajectory segment) is less than the distance to any other object, then that segment can be declared collision-free. This allows segments of the

This work was supported by NSERC, Precarn Inc., and Braintech Inc.

M. Baumann, D. Dupuis, S. Léonard and E. Croft are with the Collaborative Advanced Robotics and Intelligent Systems Laboratory, Institute for Computing, Information, and Cognitive Systems, University of British Columbia, Vancouver, Canada. mabauman@cs.ubc.ca

J. J. Little is with the Computer Science Department, University of British Columbia, Vancouver, CA little@cs.ubc.ca

trajectory that are far from any collisions to be quickly tested and declared safe, reserving computation for trajectory segments with potential collisions.

Not only does this method improve the initial PRM construction step, but the same sampling method will be used herein to speed up the visibility computations as well.

Yong and Gupta [12] detail the construction of the probabilistic roadmap of an initially unknown environment using an arm-mounted laser rangefinder. Dynamic roadmap construction is desirable for unknown environments, but requires a high degree of efficiency if the robot is expected to respond to the newly explored environment in a timely manner.

Jaillet and Simeon [4] present a PRM-based path planner that can accommodate moving obstacles. Their approach uses a precomputed “multi-query” PRM to avoid static obstacles, but supplements that with a “single-query” online modification of the path planning to take the dynamic obstacles into account. This method combines the efficiency of the pre-computed PRM with the flexibility of the online-computed single-query system.

Tarabanis, Tsai & Kaul [11] demonstrate a method to compute the boundary of the region of space from which a planar polygonal target is visible amid polyhedral objects. We use this method to compute the meshes which describe the boundary  $B$  separating the visible region from the occluded region.  $B$  subdivides task space  $P$  into  $P_{vis}$ , the space from which a camera can see the target, and  $P_{occ}$ , the space from which view of the target is blocked by an object.

Tarabanis, Tsai, Abrams & Allen [9] describe a method for describing a number of additional constraints on visual sensors. The constraints include the limited field-of-view, the desired image resolution and depth-of-field. These constraints allow the MVP system [10] to compute optimal viewpoints that conform to the sensor’s limitations.

LaValle, González-Baños, Becker and Latombe [6] present a trio of motion planning strategies for maintaining visibility of a moving target for a mobile observer. For targets whose motion is predictable, the strategy can compute optimal motions for observers with low-dimensional configuration spaces.

Briggs and Donald [1] demonstrate a system to compute the necessary motion to move a mobile sensor into a vantage point from which it can monitor a region of interest for a visually recognized target. Their approach uses a method similar to the one described by Tarabanis et al. [11] to determine the visibility region.

Marzouqi and Jarvis [7] solve the opposite problem, planning motions to minimize visibility of a robot to hostile observers. Their approach uses a dense 2D grid-based environment map to compute minimum-visibility paths across a cluttered environment. While effective in 2D, the grid-based approach may not be scalable to high-dimensional configuration spaces.

### III. THE ALGORITHM

We can test any point in task space  $P$  to determine if a line of sight exists from the point to the target. This is

accomplished by dividing the task space  $P$  into  $P_{vis}$ , the “visible region” from which the target is visible, and  $P_{occ}$ , the “occluded region”, from which an obstacle occludes the line of sight to the target. In practice, the algorithm presented by Tarabanis et al. [11] provides a triangle-mesh representation of  $B$ , the boundary that separates  $P_{vis}$  and  $P_{occ}$ .

The occlusion-avoidance algorithm computes the occluded arc length of the camera-center motion. Roadmap penalties are assigned based on this value. We compute the minimum-occlusion path within the roadmap by minimizing this penalty over multiple successive trajectories from the start configuration to the goal. (See Fig. 2)

#### Algorithm OCCLUSION-PENALTIES

```

Let  $R$  be a probabilistic roadmap.
Let  $N$  be the set of all nodes in  $R$ .
Let  $E$  be the set of all edges in  $R$ .
Let  $B$  be the boundary between  $P_{occ}$  and  $P_{vis}$ .
Let  $traj(e)$  be the camera-center trajectory of edge  $e$ .

for each edge  $e$  in  $E$ 
  let  $sum = 0$ 
  compute intersections  $I$  between  $traj(e)$  and  $B$ 
  let  $S$  be the set of segments of  $traj(e)$  defined by
  splitting it at intersections  $I$ 
  for each segment  $s$  in  $S$ 
    if the midpoint of  $s$  is inside  $P_{occ}$ 
      add  $length(s)$  to  $sum$ 
    end if
  end for
  Apply penalty to  $e$  proportional to  $sum$ .
end for

```

Fig. 2. The Occlusion-Penalties Algorithm.

#### A. Input Data

The robot’s environment determines the region from which the target can or cannot be seen. The objects in the environment are represented as a closed, manifold, oriented triangle mesh, usually derived in practice from CAD models of the robot’s workspace. (See Fig. 3) The system can only account for obstacles that are described in this mesh, that is, objects that are detected offline.

The target itself is represented as a planar polygon in task space, as in [11]. This would typically be a planar polygon derived from the image in which the target is first detected, or a region (such as the center of the bin) in which targets are always to be found.

The boundary of the visibility region is an open, manifold, oriented triangle mesh computed from the environment obstacle mesh and the target polygon using the algorithm described in [11]. (See Fig. 3)

Given a probabilistic roadmap of the environment, the algorithm will compute visibility penalties for each edge in the roadmap, allowing us to compute minimum-occlusion paths through the roadmap using Dijkstra’s algorithm. We

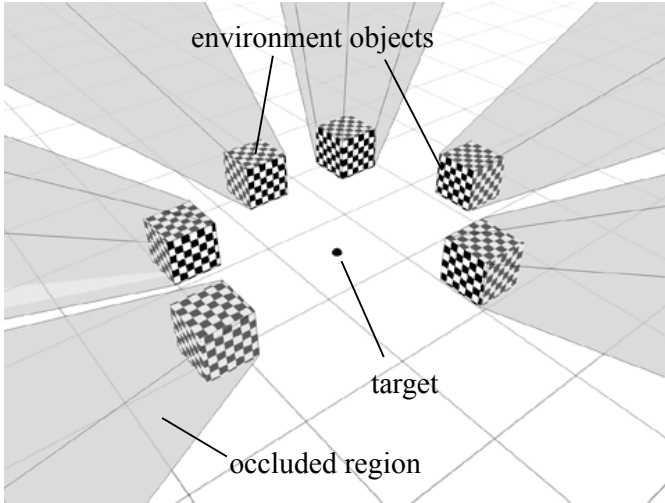


Fig. 3. The environment objects (chequered) occlude the line of sight to the target. The target is only visible from within  $P_{vis}$ .  $P_{vis}$  is represented by its boundary (light grey).

ensure that the target always lies in the sensor’s field of view, by only allowing nodes in the roadmap that fulfill the field-of-view constraint. Roadmap points could also be selected to enforce other constraints such as a range limit of the sensor, but in our case this was not necessary.

Finally, the algorithm requires a forward-kinematics routine allowing it to compute the position of the arm-mounted sensor given a point in the configuration space.

### B. Mesh-Trajectory Intersection

The key step within the occlusion avoidance algorithm is to compute the intersections of a single trajectory with the visibility boundary.

The boundary  $B$  is represented as a triangle mesh. For the purposes of this step, we can treat this as a simple list of triangles. The trajectory is represented as two points in the configuration space  $C$ . The trajectory is a linear interpolation between these points, which generates a nonlinear end-effector trajectory in task space  $P$ . The interpolation is parameterized with parameter  $t \in [0, 1]$

The algorithm performs a binary search along the trajectory, interpolating  $t$  and using the forward kinematics routine to compute the position of the arm-mounted camera. At each sampled point, we compute the minimum distance from the camera position to the boundary, using the method detailed below. If the distance is less than a threshold  $\epsilon$ , an intersection is reported.

It is important to note that if the trajectory passes through two triangles that are less than  $2\epsilon$  apart, they may be detected as a single intersection. This will cause the algorithm to ignore the small distance between the two intersections, be it either an occluded or nonoccluded segment.  $\epsilon$  must thus be chosen to be small enough that such errors have no significant effect on the outcome. This is particularly a problem if there are triangles that meet at highly acute angles in the mesh, as no matter how small  $\epsilon$  is, the trajectory could

pass close enough to the edge connecting the triangles to miss one (or both) of the intersections.

Following the adaptive-bisection method described by Schwarzer et al. [8], we also compute an upper bound for the arc length, called  $displacement_{max}$  of the camera’s motion for each trajectory segment. If  $displacement_{max}$  is less than the distance from the camera position at either end of a trajectory segment to the nearest triangle, then that segment cannot contain an intersection. Thus no further subdivisions of that segment are necessary (see Fig. 4). We compute  $displacement_{max}$  in the following manner:

let  $Q_1$  and  $Q_2$  be the two configurations,  
 let  $\theta_{1-d}$  be angular differences between  $Q_1$  and  $Q_2$   
 for robot joints  $1-d$ ,  
 and let  $r_{1-d}$  be the radii of the robot links in the plane of each link’s rotation.

Then, we can compute the maximum displacement as follows:

$$displacement_{max} = \sum_{i=1}^d (\theta_i \times \sum_{j=i}^d r_j). \quad (1)$$

The maximum arc length due to a given joint in the arm is the the angular displacement  $\theta$  of the joint multiplied by the maximum possible length of the arm above it. Since this is an upper bound, the worst-case scenario is considered where all the higher links are positioned so that they form the largest possible radius. Thus, the worst-case radius for a given joint is the sum of the radii of all links above that joint. The maximum displacement for the whole arm is thus the sum of all joint angles multiplied by their respective worst-case radii, which is the maximum possible end-effector arc length,  $displacement_{max}$ .

Accurately bounding the running time of the Mesh-Trajectory intersection algorithm is difficult due to its adaptive nature. An absolute worst-case scenario would involve a trajectory whose entire length lies at a distance minutely greater than  $\epsilon$  from the mesh surface. The algorithm would thus exhaustively subdivide the trajectory until the  $displacement_{max}$  value for every segment is less than the distance to the mesh. Thus, we can state that the algorithm will terminate when  $displacement_{max} < \epsilon$  for all segments.  $displacement_{max}$  monotonically decreases as the trajectory segments are subdivided, and  $\epsilon$  is a constant, so we can be certain that the algorithm will terminate in a finite time for any finite  $\epsilon$ . In the described worst-case scenario, the subdivision ends when there are  $displacement_{max}/\epsilon$  leaf nodes in a perfect binary tree, indicating that the trajectory has been subdivided until all the segments must terminate. Thus we can compute that there are  $2 \times (displacement_{max}/\epsilon) - 1$  nodes in the tree, which means the computation time is linear with respect to the ratio of  $displacement_{max}$  and  $\epsilon$ .

Fortunately, for articulated robots, the curved trajectories are unlikely to lie so precisely along the visibility boundary. Actual running times will depend heavily on the geometry of the robot and visibility boundary.

The intersection detection process returns the list  $I$  of intersection points in task space. Additionally, it returns the associated parameter value  $t$  for each intersection.

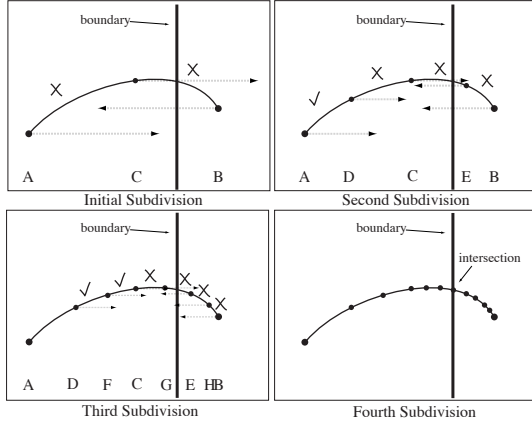


Fig. 4. Trajectory-boundary intersection. The trajectory  $AB$  is subdivided into two segments  $AC$  and  $CB$ .  $displacement_{max}$  is computed for  $AC$  and  $CB$  (shown as dotted arrows). If the distances from each end of a segment to the boundary do not both exceed  $displacement_{max}$ , then that segment must be further subdivided. In the second subdivision we see that both  $A$  and  $D$  are further than  $displacement_{max}$  from the boundary. Thus  $AD$  is “checked off”: there can be no intersections in  $AD$ , so no further subdivision of  $AD$  is necessary. Subdivision continues for two more iterations. The midpoint of  $GE$  is within  $\epsilon$  of the boundary, and is reported as an intersection. All other segments will (eventually) be “checked off” as free of intersections in a similar manner to segment  $AD$ .

### C. Trajectory Segment Classification

The intersection points  $I$  divide the trajectory into a set of segments  $S$  (See Fig. 5). Each segment must be tested to determine whether it lies inside or outside the visible space  $P_{vis}$ . In practice, the parameter values  $t$  of the intersections are used for further computations. Both the classification and arc length steps rely on that parameter.

The boundary is again represented by a triangle mesh which divides  $P$  into two halves. Since the boundary itself has no explicit inside or outside, a point in  $P$  must be supplied to act as a reference for which half of  $P$  is  $P_{vis}$ . This point is referred to as the “safe point” and corresponds to a point known to have a clear line of sight to the target. A convenient safe point is the target’s location. The trajectory is again represented as a pair of points in  $C$ , whose interpolation is parameterized by  $t$ , and subdivided by the intersection points  $S$ .

Let  $a$  be the size of set  $I$ . We split the trajectory at each intersection point in  $I$ , producing  $a + 1$  trajectory segments. For each segment we compute a midpoint  $m$  by linear interpolation of the parameter  $t$ . We compute the task-space position of  $m$  and then test whether it is inside or outside  $P_{vis}$  by computing the valence of a line segment connecting  $m$  to the safe point. The valence is simply the number of intersections between the line segment and the boundary. An even valence indicates that  $m$  is inside  $P_{vis}$ , as the line segment has re-entered the volume as many times as it has exited it [3].

The classification outputs a boolean value for each segment, indicating if the segment is occluded or not.

Each line segment valence test has a complexity of  $O(F)$ , where  $F$  is the number of triangulate faces in the boundary mesh. Thus, for a trajectory of  $s$  segments, the overall complexity is  $O(sF)$ . This can be improved by using an acceleration structure to reduce the number of faces that must be checked.

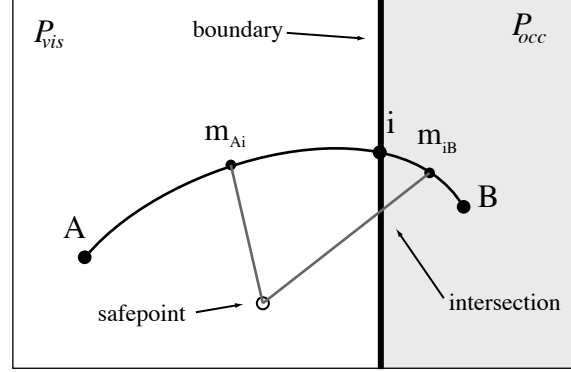


Fig. 5. Trajectory segment classification. Trajectory  $AB$  is divided by intersection point  $i$  into two segments:  $Ai$  and  $iB$ . We compute the midpoint of each segment  $m_{Ai}$  and  $m_{iB}$ . We then create a line segment from each midpoint to the “safe point”: a point which is known to be in  $P_{vis}$ . We test each line segment for intersections with the visibility boundary. If the line segment has an even number of intersections (valence), then the associated point is in  $P_{vis}$ . If the line segment has an odd valence, then the associated point is in  $P_{occ}$ . Thus we can classify  $Ai$  as visible and  $iB$  as occluded.

### D. Trajectory Task-space Arc Length

To establish the length of a trajectory in task space, we sample a series of points along the trajectory and compute the length of the piecewise-linear approximation of the trajectory formed by connecting the sampled points with line segments. While only approximate, this method does not rely on any knowledge of the trajectory’s shape, which can be complex due to the nonlinear nature of the forward kinematics.

### E. Penalty Assignments

The path planner has the twin goals of avoiding occlusions and computing an efficient path. The penalty assigned to each edge in the PRM is thus a linear combination of the occlusion penalty and the  $C$ -space distance between the nodes.

let  $D(e)$  be the joint-space distance between the ends of edge  $e$ .

let  $V(e)$  be the occluded distance between the ends of edge  $e$ .

The penalty on edge  $e$  is thus:

$$P(e) = D(e) + \lambda * V(e). \quad (2)$$

Where  $\lambda$  is a positive parameter which controls the strength of the occlusion avoidance. If  $\lambda$  is zero, then the occlusion information is ignored and the system acts as a typical shortest-distance PRM planner.

### F. Path Planning using the PRM

Once the penalties are assigned to the PRM edges, the rest of the path planning is identical to a typical PRM. We compute the lowest-penalty path through the roadmap using Dijkstra’s Algorithm, which solves the lowest cost route from a given node to all other nodes. If the parameter  $\lambda$  is high enough, the planner will avoid visual occlusions, as occluded paths add significantly to the penalties. If occlusions are unavoidable, then the algorithm will still seek to minimize the (and thus the time) that the camera is unable to see the target.

Using the modified Dijkstra’s algorithm, the complexity of a single source-node solution to the lowest-penalty-path problem for the roadmap is  $O((|E| + |N|)\log|N|)$  [2], where  $E$  is the number of edges in the graph, and  $N$  is the number of nodes. This can be precomputed for every node in the roadmap if sufficient storage is available.

## IV. EXPERIMENTS

The goal of the following experiments is to test the ability of the algorithm to avoid configurations that cause visual occlusions, either at the waypoints or during interpolation. The experimental setup simulates a real-world situation where a 6-DOF articulated arm robot must observe objects in an open-topped bin. The robot must move from an initial configuration (typically where the object is first observed) to a configuration from which it can triangulate the object’s position accurately, as indicated in the introduction. In this application continuous tracking is essential. If the robot loses sight of the target object while traversing to the new position, there is no way to guarantee that the same object can be reacquired, particularly if there are many similar-looking objects in the bin. The target is represented by a square, which is used to compute the visibility boundary. If the target position is unknown when the roadmap is created, then visibility penalties must be computed for multiple target regions, and the appropriate values selected once the target position is known. If that is not feasible, then a region of interest within which the target is known to reside can be represented by a large polygon, and the system will keep the entire polygon in view as much as possible.

### A. Simulation 1

The first simulation involves the robot attempting to move across the top of the bin, avoiding occlusions by the bin walls, and by a light fixture that overhangs one half of the bin. (See Fig. 6). The light fixture is represented by a bounding box, and overhangs the wall opposite the robot. The visual target was a small square in the exact center of the bin. The starting configuration placed the sensor near the far-right corner of the bin from the robot’s perspective. The goal configuration placed the sensor above the far-left corner.

The algorithm produces two markedly different paths for values of parameter  $\lambda$  (see Fig. 7). We ran Dijkstra’s algorithm with  $\lambda = 0$ , and the robot moves in a relatively short path over top of the light fixture, as expected. The robot moves the end-effector up to avoid a collision with the

light fixture. This path leads to an occlusion of the target in the bin interior. The occlusion-aware path travels around the light fixture, and maintains the line-of-sight, at the cost of a longer path length.

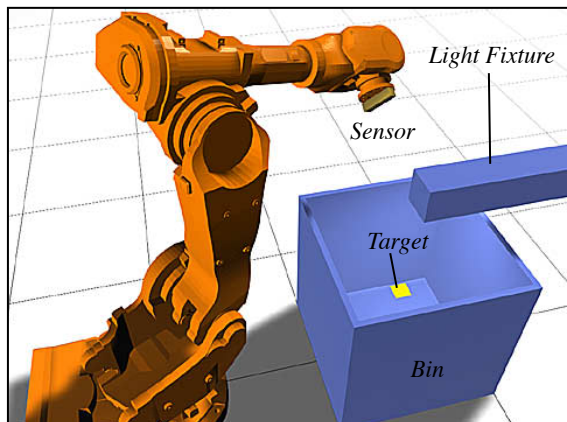


Fig. 6. Simulation 1 setup. The target is a small square region inside the bin. Environmental obstacles include the bin itself and a light fixture which overhangs the side of the bin opposite the robot.

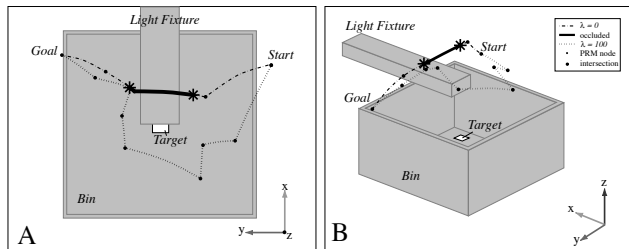


Fig. 7. Simulation 1 results. In top view A and isometric view B two paths are compared. The  $\lambda = 0$  path (dash-dotted line) is the shortest-distance solution, and travels up and over the light fixture to avoid a collision. This path causes the light fixture to occlude the visual target. The occluded segment of the path is marked as a solid black line. The  $\lambda = 100$  path (dotted line) takes the occlusion penalties into account. It avoids collisions but maintains a clear line-of-sight to the target inside the bin at all times.

### B. Simulation 2

The second simulation involves a similar task, but two visual obstacles in the form of support struts have been added. The shortest-distance path crosses above one of the struts, occluding the target. The occlusion-aware path again moves around the visual obstacle (see Fig. 8).

## V. SUMMARY

We have shown that a probabilistic roadmap path planner can be modified to take visibility into account. The framework of applying sensor-related penalties to particular motions is well-suited to the occlusion problem. This method avoids occlusions where possible, but does will still function even if an occlusion is unavoidable.

The system can compute the minimum-occlusion path through the roadmap, which is only an approximation of the actual minimum-occlusion path in the configuration space.

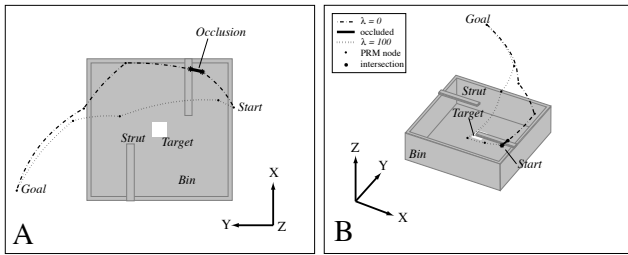


Fig. 8. Simulation 2 results. In the top view A and isometric view B two paths are compared. As in simulation 1, the  $\lambda = 0$  path is the shortest-distance path in configuration space. The target is occluded briefly by one of the struts. The  $\lambda = 100$  path avoid that occlusion, skirting carefully around the strut to maintain a line of sight.

However, the approximation will be sufficient for most applications if a dense roadmap is employed.

The main downside to this technique is its high computational cost. For simulation 1, the occlusion penalty computations took 3 hours 56 minutes in and unoptimized MATLAB implementation on a 1.80 GHz Intel Core 2 Duo. The simulation 2 penalties took 12 hours and 24 minutes. In the case of articulated robots in known, static workcells, slow computation is only a minimal problem, as the entire roadmap can be precomputed. If dynamic updates are desired, techniques that combine static and dynamically-computed roadmaps such as in [4] may improve performance. Adding new obstacles will change the shape of the visibility boundary in a wide-reaching but predictable manner. If updates to the occlusion penalties could be limited to only the affected subset of the roadmap, then a dynamic solution may be possible.

The algorithms described are generic, and can be adapted to many problems other than static work cells. We use sampling rather than specific symbolic solutions as sampling can provide approximate results for any trajectory, regardless of whether there is a closed-form expression for the trajectory's shape. This generality comes at the cost of performance. To alleviate this, it may be possible to perform application-specific improvements to one or more components. The geometric tests on the mesh would benefit from acceleration structures. Symbolic solutions for the mesh-trajectory intersection or trajectory arc-length would reduce the cost. Computing a more accurate lower bound for the maximum displacement would make the sampling process more efficient. If high accuracy overall is not necessary (i.e. the sensors are robust to small occlusions) then increasing  $\epsilon$  can greatly reduce the computation time.

Future work could improve the speed of this system, but the real potential is in dynamically creating both the roadmap and the visibility penalties in an unknown environment. Dynamically constructed roadmaps such as is seen in [12] would allow the system to function in initially unknown environments. A robot could use the algorithm presented in this work to reason about lines of sight within an environment during the exploration process. A mixed model of offline and online computation as is seen in [4] could allow the system to respond to a partially dynamic environment.

This system is a step towards robots that can intelligently examine an object, reasoning about the best data-gathering actions. With the ability to keep its eye on the proverbial ball, an occlusion-avoiding robot can perform tracking, triangulation, and many other tasks more accurately and efficiently.

## VI. ACKNOWLEDGMENTS

The authors gratefully acknowledge the contribution of the National Sciences and Engineering Research Council of Canada, Precarn Inc., Braintech Inc., The Institute for Computing, Information and Cognitive Systems and reviewers' comments.

## REFERENCES

- [1] A. J. Briggs and B. R. Donald. Visibility-Based Planning of Sensor Control Strategies. *Algorithmica*, 26:364–388, 2000.
- [2] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*, pages 527–531. The MIT Press, 1997.
- [3] Eric Haines. *Point in Polygon Strategies: Graphics Gems IV*, pages 24–46. Academic Press, 1994.
- [4] L. Jaillet and T. Simeon. A prm-based motion planner for dynamically changing environments. *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, 2:1606–1611 vol.2, 28 Sept.-2 Oct. 2004.
- [5] Lydia E. Kavraki, P. Svestka, J. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12:566–580, 1996.
- [6] S.M. LaValle, H.H. Gonzalez-Banos, C. Becker, and J.-C. Latombe. Motion strategies for maintaining visibility of a moving target. *Robotics and Automation, 1997. Proceedings. 1997 IEEE International Conference on*, 1:731–736 vol.1, 20-25 Apr 1997.
- [7] Mohamed S. Marzouqi and Ray A. Jarvis. New visibility-based path-planning approach for covert robotic navigation. *Robotica*, 24:759–773, 2006.
- [8] Fabian Schwarzer, Mitul Saha, and Jean-Claude Latombe. Adaptive dynamic collision checking for single and multiple articulated robots in complex environments. *IEEE Transaction on Robotics and Automation*, 21(3):338–353, 2005.
- [9] Konstantinos Tarabanis, Roger Y. Tsai, and S. Abrams. Planning viewpoints that simultaneously satisfy several feature detectability constraints for robotic vision. In *Proc. ICAR, Fifth Int'l Conference on Advanced Robotics*, volume 2, pages 1410–1415, 1991.
- [10] Konstantinos Tarabanis, Roger Y. Tsai, and P. K. Allen. The mvp sensor planning system for robotic vision tasks. *IEEE Transactions on Robotics and Automation*, 11:72–85, 1995.
- [11] Konstantinos Tarabanis, Roger Y. Tsai, and Anil Kaul. Computing Occlusion-Free Viewpoints. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18:279–292, 1996.
- [12] Yu Yong and K. Gupta. Sensor-based roadmaps for motion planning for articulated robots in unknown environments: some experiments with an eye-in-hand system. In *IEEE/RSJ Int'l Conference on Intelligent Robots and Systems*, volume 3, pages 1707–1714, 1996.