Stealth Tracking of an Unpredictable Target among Obstacles

Tirthankar Bandyopadhyay[1], Yuanping Li[1], Marcelo H. Ang Jr.[1], and David Hsu[1,2]

[1] National University of Singapore, Singapore
[2] Singapore-MIT Alliance, Singapore

Abstract. This papers introduces the *stealth tracking* problem, in which a robot, equipped with visual sensors, tries to track a moving target among obstacles and, at the same time, remain hidden from the target. Obstacles impede both the tracker's motion and visibility, but on the other hand, provide hiding places for the tracker. Our proposed tracking algorithm uses only local information from the tracker's visual sensors and assumes no prior knowledge of target motion or a global map of the environment. It applies a local greedy strategy, and chooses the best move at each time step by minimizing the target's escape risk, subject to the stealth constraint. The algorithm is efficient, taking $O(n)$ time at each step, where n is the complexity of the tracker's visibility region. Simulation experiments show that the algorithm performs well in difficult environments.

1 Introduction

Target tracking is an important task for mobile robots. In this work, we investigate motion strategies for an autonomous mobile robot to track a moving target among obstacles and, at same time, remain hidden from the target. We call this problem *stealth tracking*. It has many applications. For example, in surveillance, a tracker follows a target to observe the target's action and may fail to acquire useful information or endanger itself if exposed. Other examples include observing wildlife and gaming. Stealth tracking is a common tracking behavior. In all these applications, a tracker must not only follow the target, but also avoid detection by the target.

More specifically, both the tracker and the target are equipped with visual sensors. The tracker has two objectives: *tracking*—keeping the target inside the field of view—and *stealth*—staying outside the target's field of view. Obstacles in the environment complicate tracking, as they introduce two types of constraints. Motion constraints prevent the tracker and the target from crossing the obstacles. Visibility constraints prevent the tracker and the target from seeing each other.

We assume that the tracker has no prior knowledge of target motion or a global map of the environment. This assumption is valid in many real-life situations. The lack of prior information limits the use of off-line pre-computation to plan the tracker's motion. The tracker must rely on its visual sensors to acquire information on the target and the local environment, and decide the necessary motion on-line.

Ideally the tracker always moves to keep the target in the middle of its visibility region so that the target cannot escape easily. At the same time, the tracker stays outside the target's visibility region to remain hidden. However, this is impossible to achieve, if the target has the same sensing capability as the tracker, *i.e.*, the visibility relationship is symmetric: if the tracker sees the target, the target sees the tracker. This apparent dilemma can be resolved by allowing some asymmetry. Although the tracker and the target have the same visual sensors, the visibility relationship may not be exactly symmetric. The target is initially unaware of the tracker and may not detect it even if the line of visibility between the target and the tracker is clear. So we assume that the tracker can operate safely near the boundary of the target's visibility region and quickly run outside if it detects risk of exposure. This provides the tracker the "slack" needed to achieve the dual objectives of tracking and stealth.

The stealth tracking problem introduced here is related to the more common tracking problem without the stealth requirement [8]. However, the stealth requirement makes the problem more difficult. We have already mentioned the conflict between the dual objectives of tracking and stealth. In addition, good tracking strategies may not work for stealth tracking at all. For example, in a star-shaped environment, without the stealth requirement, the tracker can simply stay in the middle of the environment, but this does not work for stealth tracking. Fortunately, as we will see, a suitable formulation of the problem translates the stealth requirement as a constraint on the motion of the tracker's motion. The resulting algorithm is almost as efficient as that for tracking without the stealth requirement.

In the following, Section 2 reviews previous work. Section 3 formulates the stealth tracking problem. Section 4 describes our tracking algorithm. Section 5 gives an efficient algorithm for computing the target's visibility region. Section 6 presents experimental results. Section 7 discusses limitations of our approach and possible solutions. Section 8 summarizes main results.

2 Previous Work

The objective of tracking is to find the motion for the tracker to keep the target within the tracker's field of view. Different aspects and variants of the problem have been studied in control theory [5], computer vision [10], and robotics [9]. In comparison, our goal here is to keep the target inside the field of view by moving the tracker actively to strategic locations under motion and visibility constraints.

The stealth tracking problem is related the work in [2,6,8,13,15,16], which studies tracking without the stealth requirement. The earlier work is based on two main approaches. The off-line approach assumes that the target's trajectory or behavior is known in advance. This allows one to derive stronger theoretical guarantees of the tracking algorithm [2,6,13]. The on-line approach assumes very little knowledge about the target's behavior. This is more realistic in many applications. Our work takes the second approach.

3 Problem Formulation

We assume that the robot tracker and the target operate in a planar environment $\mathcal{W}$ cluttered with obstacles (Figure 1*a*). The free space $\mathcal{F}$ is the subset of $\mathcal{W}$ not occu-

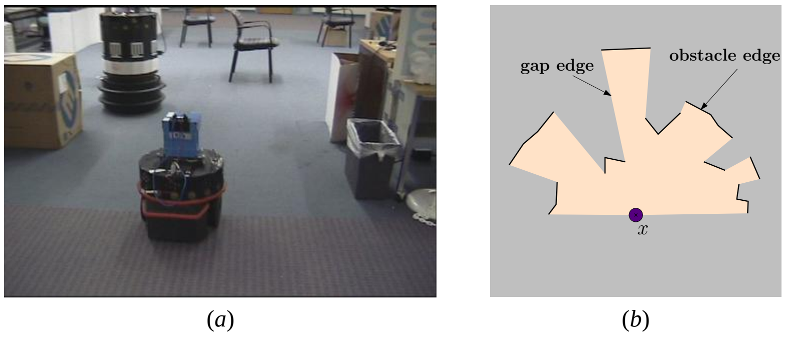

Fig. 1. A robot tracker operating in a planar environment. (*a*) A robot mounted with a laser range finder tracks another robot. Courtesy of H.H. González-Baños. (*b*) The visibility region from a sensor at x.

pied by obstacles. To simplify the presentation, assume for now that the each robot is a point in $\mathcal{W}$. The extension to the usual cylindrical robots is straightforward.

Both the robot tracker and the target have visual sensors. The sensor's field of view depends on the robot's position. Let $\mathcal{V}(x) \subset \mathcal{W}$ denote the visibility region of a robot located at x. The set $\mathcal{V}(x)$ contains all the points in $\mathcal{W}$ that have a direct line of sight to the sensor, unobstructed by the obstacles in $\mathcal{W}$. The visibility region can be modeled in various ways, depending on sensor capability. Here we assume that the sensor has an omni-directional field of view. Constraints, such as limited viewing angle and range, can be added if necessary. On real robots, the visibility region is obtained by processing sensor data. In simulation, the visibility region can be computed with a rotational plane sweep algorithm [3]. Either way, after the processing, we represent the visibility region as a polygon. The boundary of the visibility polygon consists of two types of edges: *obstacle* edges and *gap* edges (Figure 1*b*). Obstacle edges are part of obstacle boundaries. They directly block the sensor's line of sight. Gap edges result from occlusion of the line of sight elsewhere.

The tracker has two objectives, tracking and stealth. Let x_{s} and x_{t} denote the positions of the tracker and the target, and let $\mathcal{V}_{\mathrm{s}}$ and $\mathcal{V}_{\mathrm{t}}$ be the visibility regions of the tracker at x_{s} and the target at x_{t}, respectively. To track the target, the tracker must keep the target inside the field of view, meaning $x_{\mathrm{t}} \in \mathcal{V}_{\mathrm{s}}$. To maintain stealth, the tracker must stay outside the target's field of view, meaning $x_{\mathrm{s}} \notin \mathcal{V}_{\mathrm{t}}$. If the visual sensor of the tracker is stronger than that of the target, *e.g.*, by having a longer range, then the tracker's task becomes easier. It can stay at a distance from the target to remain outside $\mathcal{V}_{\mathrm{t}}$ while keeping the target inside $\mathcal{V}_{\mathrm{s}}$ at the same time. We assume, however, that the tracker and the target have similar sensors. In this case, the visibility relationship seems symmetric: $x_{\mathrm{t}} \in \mathcal{V}_{\mathrm{s}}$ if and only if $x_{\mathrm{s}} \in \mathcal{V}_{\mathrm{t}}$. Our two objectives are then contradictory and cannot be achieved simultaneously.

The visibility relationship, however, is not exactly symmetric. The target is initially unaware of the tracker and may not detect the tracker even if the tracker is inside the visibility region $\mathcal{V}_{\mathrm{t}}$. So we allow the tracker to operate in a region within a distance L of some gap edge of $\mathcal{V}_{\mathrm{t}}$ (Figure 2). Since there is an upper

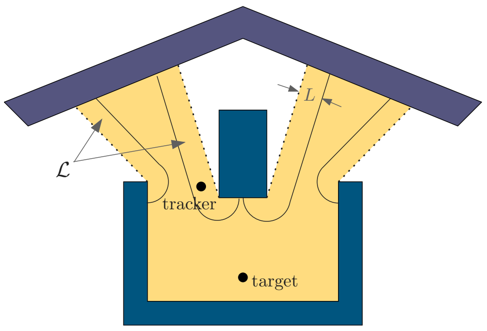

Fig. 2. The lookout region $\mathcal{L}$.

bound on the tracker's maximum velocity, the threshold L reflects the tracker's estimate on the amount of time during which it can remain safely in $\mathcal{V}_{\mathrm{t}}$ without detection, and quickly run out of $\mathcal{V}_{\mathrm{t}}$ via a gap edge if it detects risk of exposure.

More precisely, let the distance from a point $p \in \mathcal{V}_{\mathrm{t}}$ to a gap edge g be the minimum Euclidean distance between p and all points in g that are visible to p via an unobstructed line of sight. We define the *lookout* region of a gap edge g as

$$\mathcal{L}_g = \{p \in \mathcal{V}_{\mathrm{t}} \mid \text{The distance between } p \text{ and } g \text{ is less than } L\}.$$

The total lookout region $\mathcal{L}$ is then the union of $\mathcal{L}_g$ over all gap edges of $\mathcal{V}_{\mathrm{t}}$.

Now the objective of the tracker is to move in $\mathcal{L}$ so that the target remains in the field of view. The motion of the robot tracker and the target is described with a simple discretized model. Let $x(t)$ denote the position of a robot at time t. If it chooses a velocity $v(t)$ at time t, its new position $x(t+1)$ after a fixed time interval Δt is given by

$$x(t+1) = x(t) + v(t)\Delta t.$$

Here we implicitly assume that sensing occurs every Δt time. This discretized model is effective as long as Δt remains small. As we will see, our tracking algorithm is very efficient. Based on the experience of previous work [8], we expect it to run at the rate of 10 Hz, sufficient for keeping Δt small in many common tasks.

There is a maximum bound V on the robot's velocity, but no other constraint. So the robot can reach anywhere inside a circle with radius center $x(t)$ and $V\Delta t$, unless it is obstructed by obstacles. Let V_{s} and V_{t} be the maximum velocity bounds of the tracker and the target, respectively. The bounds may be different, but we must have $V_{\mathrm{s}} \geq V_{\mathrm{t}}$. Otherwise, the target can easily escape by simply running straight ahead with maximum velocity, and the tracking problem is uninteresting.

We are now ready to state the problem formally:

Problem 1. A stealth tracking algorithm must compute a sequence of actions—in this case, the velocities $v(t), t = 0, 1, \ldots, T$—for the tracker so that at any time t, $x_{\mathrm{t}} \in \mathcal{V}_{\mathrm{s}}$ and $x_{\mathrm{s}} \in \mathcal{L}(x_{\mathrm{t}})$, where $\mathcal{L}(x_{\mathrm{t}})$ is the lookout region for a target at x_{t}.

For some environments, losing the target is unavoidable. We may then try to keep the target in sight for as long as possible and maximize the target's escape time. In addition, it may be advantageous to choose actions to optimize certain criteria, for example, minimizing the distance travelled by the tracker or minimizing the distance between the tracker and the target.

Our formulation assumes that the target is unpredictable. The tracker does not know the target motion in advance, and does not have a prior map of the environment. It knows only $\mathcal{V}_\mathrm{s}$, the part of the environment visible to its sensor. It also knows the position of the target in $\mathcal{V}_\mathrm{s}$ and thus can compute the subset of the target's visibility region within $\mathcal{V}_\mathrm{s}$. These assumptions certainly limit the tracker's ability to make the best decisions. It must rely on purely local information and choose the necessary motion in an on-line fashion. However, the assumptions are realistic in many applications. They may also simplify the problem in some respects: if the tracker does not rely on a global map, it does not need to localize with respect to the map, a difficult problem in itself [17].

Our formulation also assumes that the tracker can estimate the risk of exposure and take evasive action if necessary. In some applications, the target has directional sensors and must turn to look in the direction of the tracker. Upon observing this, the tracker can then take evasive action. If the target has omni-directional sensors, the best action for the tracker is nevertheless to stay close the boundary of the target's visibility region, but it is more difficult to decide when to take the evasive action. One possible solution is to learn a model of the target's typical movement patterns. Deviation from the model may indicate that the target is trying to look for the tracker. This is an interesting problem in itself, but is beyond the scope of this paper.

4 The Stealth Tracking Algorithm

Our stealth tracking algorithm uses a local greedy strategy. It defines a function that measures the risk for the target to escape. At each step, the tracker moves to a location that minimizes the risk function, subject to the stealth constraint.

4.1 The Risk Function

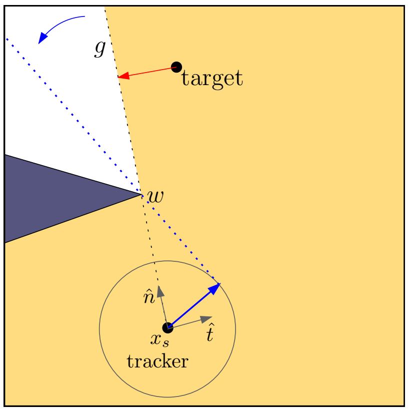

Fig. 3. The tracker's best move in one time step.

In our visibility model, the target can only escape from the tracker's visibility region through one of its gap edges. Let us focus on a single gap edge g for now (see Figure 3). If the target plans to escape through g, the best action is to move towards g along a straight-line path perpendicular to g. The length of this path gives the minimum distance d_esc that the target must travel to escape through g. Since the target's velocity is bounded, this also gives an estimate on the minimum time to escape. To prevent the target from exiting g, the tracker should move to maximize d_esc. If ℓ is the occlusion line containing g and w is the obstacle vertex abutting g, we can think of such a move as rotating ℓ about w so that the g is as far away from the target as possible. During a time step of length Δt, the tracker can only move inside a circle of radius $V\Delta t$, where V is the tracker's maximum velocity bound. Clearly the best move for the tracker is to go until ℓ is tangent to this circle (Figure 3). This move can be decomposed into two orthogonal directions $\hat{n}$ and $\hat{t}$, where $\hat{n}$ is a unit vector pointing from x_{s} to w and $\hat{t}$ is a unit vector orthogonal to $\hat{n}$. Both directions contribute to increased d_{esc}. Moving along $\hat{t}$ gives the maximum instantaneous increase of d_{esc}. Moving along $\hat{n}$ brings the tracker much closer to the pivot vertex w so that a future step along $\hat{t}$ gives even larger increase of d_{esc}. Intuitively $\hat{t}$ gives the direction to reduce current escape risk of the target, and $\hat{n}$ gives the direction to reduce future escape risk. These considerations lead to the risk function

$$f = (1/2)(r/e)^2 \tag{1}$$

where r is the distance between w and the tracker position x_{s}, and e is the target's minimum escape distance with respect to g. The risk f grows, as r increases and the tracker is further away from w, or as e decreases and the target is closer to the gap edge g. The gradient of f with respect to the tracker position x_{s} gives the direction to increase f at the maximum rate:

$$\nabla f = (r/e^2)\nabla r - (r^2/e^3)\nabla e. \tag{2}$$

The vectors ∇r and ∇e are oriented along $\hat{n}$ and $\hat{t}$, respectively. So the combined direction ∇f balances the current and the future escape risk. There are certainly alternative ways of combining r and e to form the risk function. They basically change the relative weight of ∇r and ∇e in ∇f. It is not clear which choice is the best. So we have decided to use the simplest one.

Given the tracker position and a gap edge, ∇f can be computed in constant time. To minimize f, the tracker follows the negated gradient $-\nabla f$.

So far, we have considered the risk with respect to a particular gap edge. In the worst case, the target would run towards the nearest gap edge $\hat{g}$. We must find $\hat{g}$ and minimize the risk with respect to $\hat{g}$.

To find $\hat{g}$, we compute the shortest path in $\mathcal{V}_{\mathrm{s}}$ that goes from x_{t} to each of the gap edges. In general, this can be done in $O(n^2 \lg n)$ time using the visibility graph method [12], where n is the number of vertices in $\mathcal{V}_{\mathrm{s}}$. However, since $\mathcal{V}_{\mathrm{s}}$ is a visibility region and thus star-shaped, a better algorithm is available and achieves the $O(n)$ time [8]. Hence the following lemma.

Lemma 1. *The direction for minimizing the worst-case risk can be computed in $O(n)$ time, where n is the number of vertices in the tracker's visibility region $\mathcal{V}_{\mathrm{s}}$.*

4.2 Feasible Regions

Having determined the direction of the tracker's motion $-\nabla f$, we now consider how much the tracker should move along $-\nabla f$. Suppose that the tracker's current position is x_{s}. The tracker's new position x_{s}' in one time step must satisfy three conditions:

- ${x_\mathrm{s}}' \in R$, where R is a disc with radius $V\Delta t$ and center x_s, because of the maximum velocity bound.
- ${x_\mathrm{s}}' \in \mathcal{V}_\mathrm{s}$. The new position must lie inside the tracker's current visibility region $\mathcal{V}_\mathrm{s}$, because the tracker has only a local map from the sensor and has no information outside $\mathcal{V}_\mathrm{s}$. This condition also guarantees that ${x_\mathrm{s}}'$ is collision-free, ${x_\mathrm{s}}' \in \mathcal{F}$.
- ${x_\mathrm{s}}' \in \mathcal{L}(x_\mathrm{t})$, where x_t is the target's current position. The tracker's new position must remain in the lookout region to maintain stealth.

So ${x_\mathrm{s}}'$ lies in the intersection $R \cap \mathcal{V}_\mathrm{s} \cap \mathcal{L}(x_\mathrm{t})$, which we call the *feasible region*.

To obtain the feasible region, we first compute the intersection $I = \mathcal{V}_\mathrm{s} \cap R$. Let $p_i, i = 1, 2, \ldots, n$ be the vertices of $\mathcal{V}_\mathrm{s}$ in counter-clockwise order. Since $\mathcal{V}_\mathrm{s}$ is star-shaped, we represent it as a list of triangles $\Delta x_\mathrm{s} p_i p_{i+1}$ for $i = 1, 2, n-1$. We then intersect each triangle with R and obtain a new convex shape in constant time. So I can be computed in $O(n)$ time.

The set I is basically the visibility region $\mathcal{V}_\mathrm{s}$ clipped by the boundary circle of R. Usually the time step Δt is small. Thus R is also small and contains only a small constant number of obstacle vertices and edges, if any. By merging consecutive convex shapes in I whenever possible, the resulting I has only constant size.

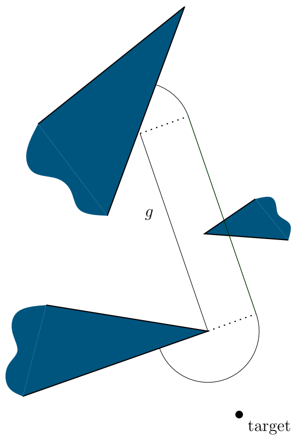

Fig. 4. Comparing $\mathcal{L}_g$ and $\mathcal{L}'_g$. The obstacle in the middle right is ignored in $\mathcal{L}'_g$.

Next, we compute the target's visibility region in order to get $\mathcal{L}(x_\mathrm{t})$. The tracker has no information outside of its own visibility region $\mathcal{V}_\mathrm{s}$. It can only compute the subset of the target's visibility region within $\mathcal{V}_\mathrm{s}$. Since there is no confusion, we use $\mathcal{V}_\mathrm{t}$ to denote this subset from now on. Given x_t, there is a simple algorithm that computes $\mathcal{V}_\mathrm{t}$ in in $O(n)$ time, because $\mathcal{V}_\mathrm{s}$ is star-shaped. We defer the discussion of this algorithm until Section 5.

For each gap edge g of $\mathcal{V}_\mathrm{t}$, there is a lookout region $\mathcal{L}_g$, which may be quite complex if many obstacles are close together. We compute a simpler set $\mathcal{L}'_g$, which consists of a possibly clipped rectangle adjacent to g and two circular sectors that cap the rectangle (Figure 4). The width of the rectangle and the radius of the circular sectors are both L, which is the distance threshold for satisfying the stealth requirement. Compared with $\mathcal{L}_g$, the set $\mathcal{L}'_g$ ignores all the obstacles except the two at the end points of g. The set $\mathcal{L}'_g$ is sufficient for computing the feasible region, because $\mathcal{L}_g \cap I = \mathcal{L}'_g \cap I$. A point in the obstacle is certainly outside $\mathcal{V}_\mathrm{s} \supset I$.

It is important to observe that $\mathcal{L}'_g$ can be represented as a union of at most three convex shapes, each of which has constant size. Assuming I has constant size, we can intersect every convex shape in I with $\mathcal{L}'_g$ in constant time. There are at most $O(n)$ gap edges, and thus we can compute the feasible region in $O(n)$ time. The feasible region is represented as a list of convex shapes approximated by convex polygons, all having constant sizes. Again, if we assume that R is small enough, then R intersects only a constant number of lookout regions, and the feasible region has a constant size. We summarize the result in the lemma below.

Lemma 2. *The feasible region for the tracker's new position can be computed in $O(n)$ time, where n is the number of vertices in the tracker's visibility region $\mathcal{V}_s$. Assuming that the time step Δt is small enough, the feasible region has $O(1)$ size.*

4.3 Locally Optimal Action

To choose the best move, the tracker should minimize the risk f over the feasible region. Consider the linear approximation of f at x_s:

$$f(x_s + \Delta x) \approx f(x_s) + \nabla f(x_s) \cdot \Delta x. \tag{3}$$

Minimizing function (3) is equivalent to

$$\min_{\Delta x} \nabla f(x_s) \cdot \Delta x \qquad \text{subject to } \Delta x \text{ lying in the feasible region.}$$

As the feasible region consists of a list of convex polygons, the problem reduces to linear programming [3]. The minimum solution $\Delta\hat{x}$ must lie at the vertex of one of the convex polygons. By projecting all the vertices in the feasible region along $\nabla f(x_s)$, we can find the minimum in $O(c)$ time, where c is the number of vertices describing the feasible region. We now set the tracker's new position $x_s{}' = x_s + \Delta\hat{x}$.

Combining the above result with Lemmas 1 and 2 gives the following theorem.

Theorem 1. *Let n be the number of vertices in the tracker's visibility region $\mathcal{V}_s$. The best action for the tracker to minimize the risk function f within the feasible region can be computed in $O(n)$ time at each time step.*

Without preprocessing, we cannot hope to achieve running time better than $O(n)$. However, as the tracker's visibility region changes at each time step, there is little opportunity for preprocessing.

4.4 Emergency Actions

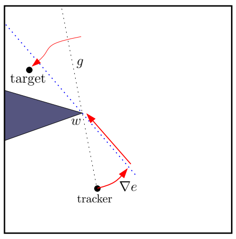

Fig. 5. Emergency action.

Our tracking algorithm also handles two emergency situations. In the first case, the tracker loses the target. Suppose that the target exits a gap edge g and that the obstacle vertex w abuts g (Figure 5). When the tracker detects that the target has exited g, it tries to rotate the occlusion line ℓ that contains g as fast as possible by moving along ∇e for a fixed number of steps, and hopes to regain the target quickly. If this fails, the tracker then goes directly towards w in order to eliminate the occlusion line ℓ. In the second case, the tracker is exposed. It computes a shortest path to each of the lookout regions and goes towards the nearest one.

5 Computing the Target's Visibility Region

In this section, we describe how to compute $\mathcal{V}_\mathrm{t}$, the subset of the target's visibility region inside the tracker's visibility region $\mathcal{V}_\mathrm{s}$. Since $\mathcal{V}_\mathrm{s}$ is a simple polygon, $\mathcal{V}_\mathrm{t}$ can be computed in optimal $O(n)$ time [7,14,11]. However, a simpler algorithm is possible here, because $\mathcal{V}_\mathrm{s}$ is star-shaped.

Our basic idea is to walk along the boundary of $\mathcal{V}_\mathrm{s}$ and compute the vertices of $\mathcal{V}_\mathrm{t}$ incrementally. To initialize, we use the line that goes through x_s and x_t to divide $\mathcal{V}_\mathrm{s}$ into two halves, P and P' (see Figure 6). We now describe the algorithm for P, the left half.

Let p_0 the point where the ray $\overrightarrow{x_\mathrm{s}x_\mathrm{t}}$ intersects the boundary of $\mathcal{V}_\mathrm{s}$. We number the vertices $p_0, p_1, p_2, \ldots$ of P in counter-clockwise order, starting from p_0. Every vertex of P is visible to x_s, because P is a subset of the visibility region of x_s. The first vertex of the target's visibility region $\mathcal{V}_\mathrm{t}$ is p_0, which is visible to x_t because x_t lies on the line segment $\overline{x_\mathrm{s}p_0}$. Now we walk along the boundary of P and visit the vertices $p_1, p_2, \ldots$ in this order. Let p_i be the latest vertex of P that is visible to x_t. For every new vertex p_j encountered, where $j > i$, if the ray $\overrightarrow{x_\mathrm{t}p_j}$ lies to the right of $\overrightarrow{x_\mathrm{t}p_i}$, we simply move to p_{j+1}, because p_j must be blocked by edges adjacent to p_i. If $\overrightarrow{x_\mathrm{t}p_j}$ lies to the left of $\overrightarrow{x_\mathrm{t}p_i}$, we claim that p_j is visible to x_t. To see this, we have to show that no boundary edge of $\mathcal{V}_\mathrm{s}$ intersects $\overline{x_\mathrm{t}p_j}$. Suppose, for the purpose of contradiction, that some edges of P intersect $\overline{x_\mathrm{t}p_j}$. Let e be the intersecting edge closest to x_t along $\overline{x_\mathrm{t}p_j}$. If e belongs to the polygonal chain between p_0 and p_i, the chain must cross $\overline{x_\mathrm{t}p_i}$. This is impossible, because p_i is visible to x_t. If e belongs to the polygonal chain between p_i and p_j, then some vertex on the chain (excluding p_i and p_j) must be visible to x_t. This contradicts the fact that p_i is the latest vertex visible to x_t. If e belongs to the polygonal chain after p_j, the chain must cross $\overline{x_\mathrm{s}p_j}$. This is also impossible, because p_j is visible to x_s. Of course, none of the edges in P' can intersect $\overline{x_\mathrm{t}p_i}$. Otherwise, they would block the visibility line through x_s and x_t. Hence, p_j is visible to x_t. We compute the intersection of the ray $\overrightarrow{x_\mathrm{t}p_i}$ and the edge $\overline{p_{j-1}p_j}$ and add both the intersection point and p_j to $\mathcal{V}_\mathrm{t}$ as new vertices of $\mathcal{V}_\mathrm{t}$. We then move to p_{j+1} and continue until all vertices are visited.

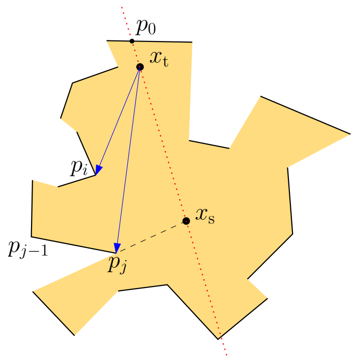

Fig. 6. Computing the target's visibility region.

The right half of $\mathcal{V}_\mathrm{s}$ can be processed similarly, except that now, we walk along the boundary in clockwise order. We then merger the two halves of $\mathcal{V}_\mathrm{t}$.

Our initialization step takes $O(n)$ time, where n is the number of vertices in $\mathcal{V}_\mathrm{s}$. When walking along the boundary of $\mathcal{V}_\mathrm{s}$, we encounter each vertex exactly once and process it in constant time. Hence the following theorem.

Theorem 2. *Let n be the number of vertices of the tracker's visibility region $\mathcal{V}_{\mathrm{s}}$. Given a target position x_{t}, the visibility region of x_{t} within $\mathcal{V}_{\mathrm{s}}$ can be computed in $O(n)$ time.*

6 Experimental Results

To test the effectiveness of our tracking algorithm, we implemented it and ran it with different environments in simulation. In Figures 7–11, we show five representative experiments to illustrate the behavior of the algorithm. In these figures, dark blue regions indicate obstacles. Red crosses mark the target's trajectory. Blue boxes mark the tracker's trajectory. Two circles mark the target's and the tracker's current positions. The tracker's visibility region is marked with thick blue lines. The target's visibility region is shaded in light red. The lookout region is marked with thin black lines.

Corridor (Figure 7) This example shows the tracker's behavior when the target turns round a corner in a corridor. Initially the tracker stays near the lower right corner of the obstacle inside a lookout region to maintain stealth (Figure 7*a*). When the target makes a turn, a new lookout region develops, and the line of sight pivots about the upper right corner of the obstacle. The tracker follows the gradient of the risk function to take advantage of this. It swings out to improve its visibility (Figure 7*b*). Finally the tracker goes towards the upper right corner and stays there to maintain stealth (Figure 7*c*).

The tracker's success in this environment depends on the target's turning radius. If the target makes a sharp turn, the line of sight may pivot too fast for the tracker to follow, and the tracker loses the target. In this case, the tracker executes the emergency action (Section 4.4) to try regaining the target.

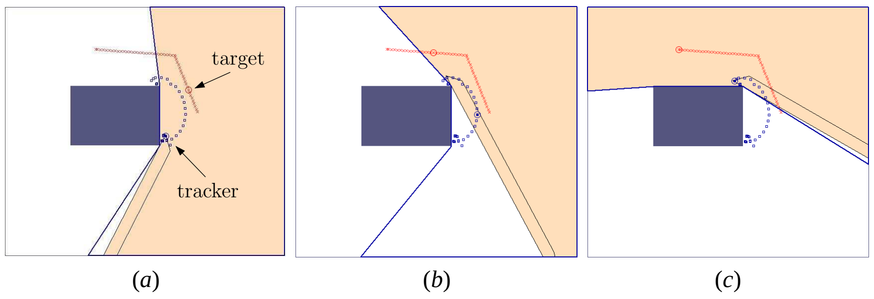

Fig. 7. The target turns around a corner.

Forest (Figure 8) Imagine a target going straight along a road passing through dense forests. If the tracker follows behind the target on the road, it risks exposure. The figures show the path that the tracker chooses when faced with such a situation. In general, the tracker stays on side of the road near the obstacles to avoid the risk of exposure. It tries to trail the target as closely as possible, given the constraint that it must stay inside the lookout region of some gap edge (Figure 8*a*). When the

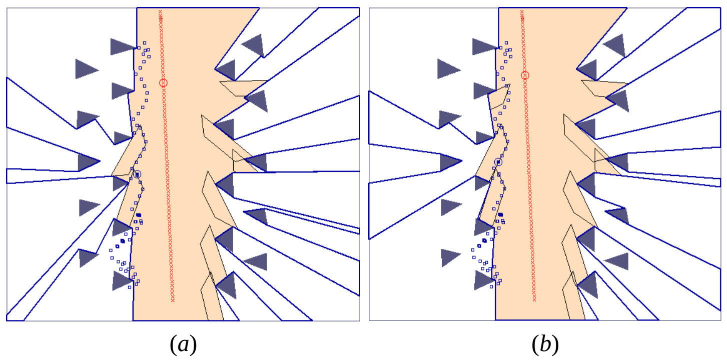

Fig. 8. The tracker switches lookout regions.

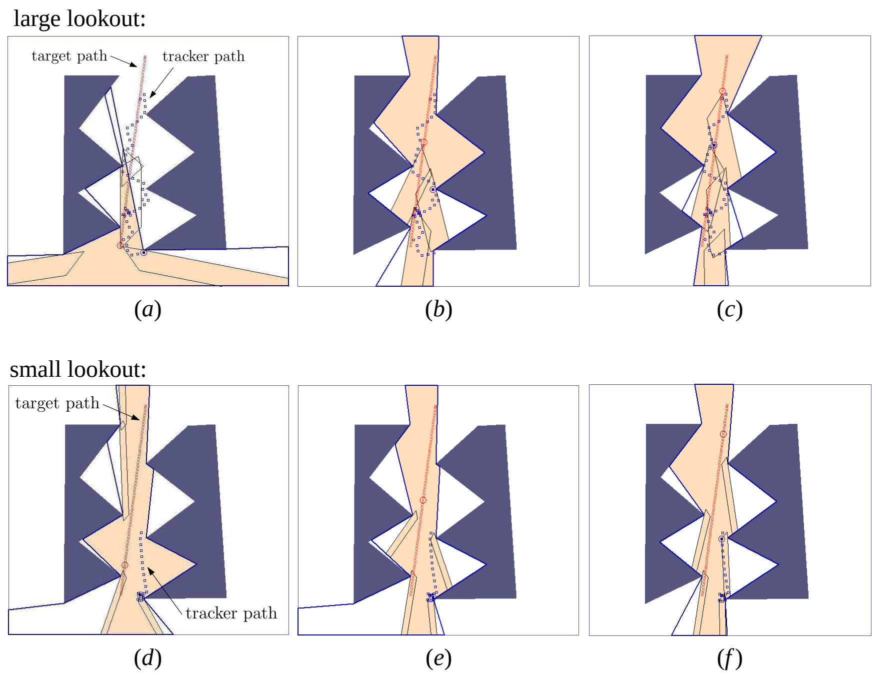

Fig. 9. The tracker's behavior changes due to different sizes of lookout regions.

lookout regions of two gap edges merge, the tracker immediately switches to the new lookout region to further reduce the risk function (Figure 8*b*).

Zigzag Pathway (Figure 9) The tracker's behavior and success may change drastically, depending on how much risk of exposure that it is willing to take. Recall that the distance threshold L controls the size of lookout regions and reflects the tracker's estimate or willingness to take risk of exposure. If L is large, the tracker has more room to maneuver, usually resulting in more successful tracking.

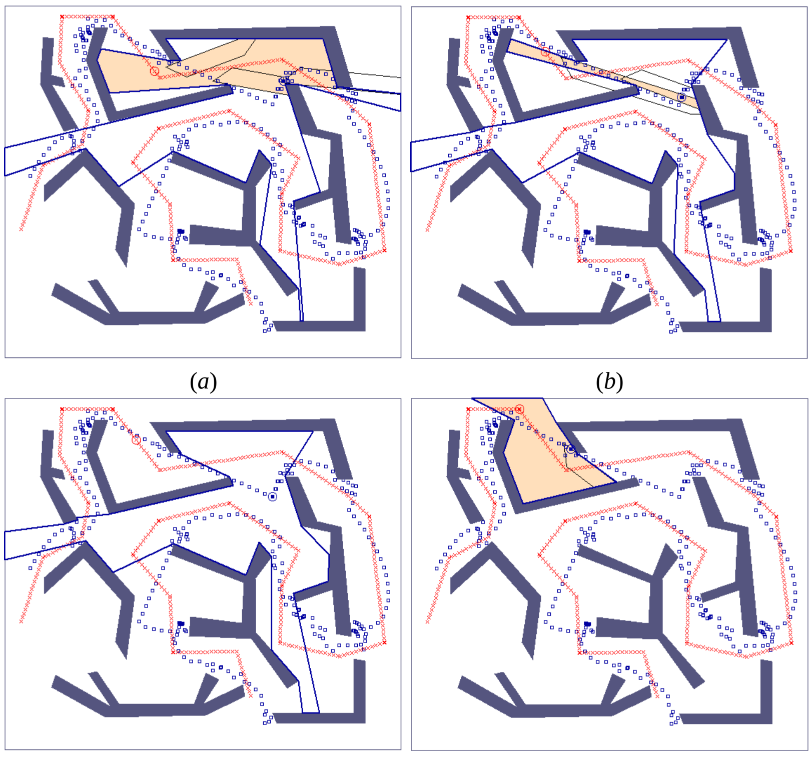
(*a*) (*b*) (*c*) (*d*)

Fig. 10. Losing and regaining the target.

Consider the environment shown in Figure 9. The target moves roughly along a straight path through the zigzag pathway. When L is large, lookout regions from two sides of the pathway often merge and the tracker can jump from one side to the other, using obstacles on both sides for cover (Figures 9*a*–*c*). It thus follows the target very closely.

For the same environment and target motion, the tracker's performance worsens when L is small. It is unable to move from one side to the other, because the lookout regions are small and do not merge. It is stuck in one lookout region and has to wait for another lookout region to come close as a result of target motion (Figures 9*d*–*e*). However, by then, it falls behind the target by a large distance (Figure 9*f*). Although the tracker does not lose the target here, the target can escape easily by making a turn at the end of its straight-line motion. No simple emergency action can recover the target, because the distance between the target and the tracker is too large.

Maze (Figure 10) The geometry of this environment is more complex than the others. The target also takes along a long and winding path. The tracker is able to follow the target till the end, with the help of emergency actions. In Figure 10*a*–10*b*, the tracker moves almost entirely in the direction ∇e, because the target is

Table 1. Tracking performance.

Environment	Total No. Steps	No. Steps Visible	No. Times Lost & Regained
Corridor	102	102	0
Forest	214	214	0
Zigzag (large L)	130	130	0
Zigzag (small L)	130	130	0
Maze	730	663	8
City Blocks	468	466	2

very close to the gap edge and the current escape risk is high. At this moment, the tracker does not have the luxury to move along ∇r to reduce the future escape risk. Unfortunately the tracker still loses the target (Figure 10*c*). Taking the emergency action (see Section 4.4), the tracker runs to the vertex that abuts the gap edge from which that the target has escaped and regains the target (Figure 10*d*). The tracker loses the target eight times during the tracking, but every time it regains the target after taking the emergency action.

City Blocks (Figure 11) This environment resembles urban areas with roads and housing blocks and gives another example with many obstacles. The target takes a long path, but the tracker successfully follows the target till the end, with a little help from emergency actions.

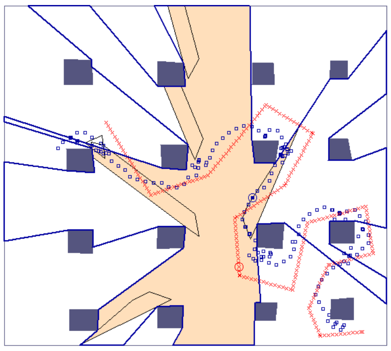
Fig. 11. Tracking a target among many obstacles.

Table 1 shows the performance statistics of the tracker in the above environments. Column 2 lists the total time length of target motion. Column 3 lists the length for which the target is visible to the tracker. Column 4 shows how many times that the target is lost and regained. From the table, we see that the tracker loses the target in the two environments where there are many obstacles. In the City Blocks environment, the tracker loses the target two times, for a duration of one step each. Every time that the tracker loses the target, it recovers it in the next step by taking the emergency action. In the Maze environment, which is more complex, the tracker loses the target eight times. It always recovers the target, but it takes more time than that needed in City Block. When there are many obstacles causing occlusions, the tracker is more likely to lose the target. The emergency action, despite it simplicity, seems quite effective in such environments.

7 Discussion

We now discuss some limitations of our approach and possible solutions.

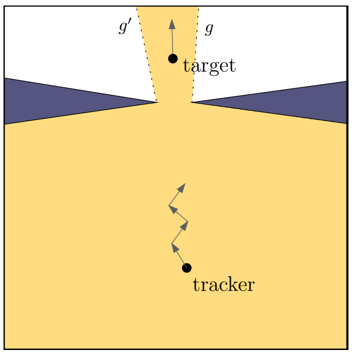

Fig. 12. Changes in the escape edge causing the robot tracker to follow an oscillatory path.

Discontinuity in the risk function. Our tracking algorithm currently minimizes the worst-case risk. The risk function f is defined with respect to the gap edge closest to the target, which we call the *escape edge*. The escape edge may suddenly change, leading to a discontinuity in f. Figure 12 shows an example. The target is moving forward about equally distant to two gap edges g and g'. Initially g is the escape edge. To minimize the escape risk with respect to g, the tracker takes a step to the left. This step increases the target's distance to g, but reduces the target's distance to g'. As the result, g' becomes the escape edge. Now the tracker takes a step to the right, which reduces the distance to g', but increases the distance to g. The escape edge switches back to g. This causes the robot tracker to oscillate and is undesirable. To deal with this issue, we can define the overall risk as the average of all gap edges' risks [8], instead of the worst-case risk.

"Discontinuity" in the lookout regions. Sometimes a lookout region may suddenly disappear. Consider the example shown in Figure 13. Initially the tracker is at x_s, safely inside the lookout region of a gap edge g. As it moves to the new position, $\mathcal{V}_\mathrm{s}$ is reduced. Although the target has not moved at all, g now lies outside $\mathcal{V}_\mathrm{s}$. The tracker can no longer ascertain the existence of g and the associated lookout region, and assumes that it is exposed. This happens, because the tracker uses only local information from the sensor and knows only the environment within $\mathcal{V}_\mathrm{s}$. If the tracker had fused sensor data through history to produce a global map, it would know that it is inside the lookout region of g. The use of only local information is clearly a limitation here.

This situation occurs only if the tracker is close to the obstacle vertex w that supports the line ℓ containing the gap edge g (Figure 13). If the tracker crosses ℓ, the gap edge disappears from $\mathcal{V}_\mathrm{s}$. To avoid this situation, we truncate the lookout region $\mathcal{L}_g$ so that the tracker does not cross ℓ.

Nonholonomic constraints. Our current model of the tracker's motion is very simple. Often the tracker robot is wheeled and is subject to nonholonomic or dynamic constraints. To deal with these additional constraints, we can use a control system to describe the robot's motion [1,4]. We no longer use the disc R for computing the feasible region, but replace it with a reachable set, the set of points reachable from the tracker's current position within time Δt according to the motion model. In fact, the disc R is simply the reachable set for our simple motion model. We

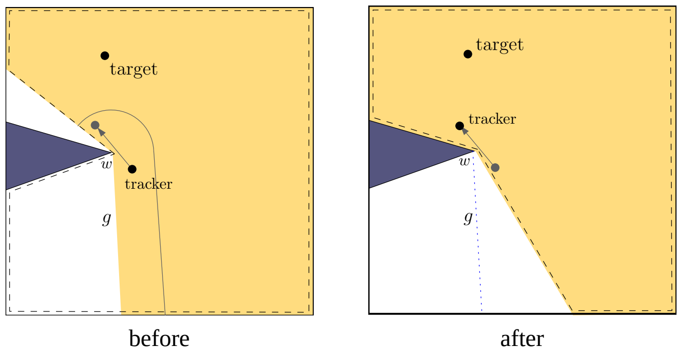

Fig. 13. The target's visibility regions before and after the tracker's move. The dashed lines indicate $\mathcal{V}_\mathrm{s}$. The shaded region indicates $\mathcal{V}_\mathrm{t}$.

can pre-compute the reachable set and approximate it using convex polygons. The computation of the feasible region can then proceed as before.

8 Conclusion and Future Work

We have introduced the stealth tracking problem, in which a robot tries to track a moving target among obstacles and remain hidden from the target at the same time. Our tracking algorithm uses only local information from the tracker's visual sensors and assumes no prior knowledge of target motion or a global map of the environment. It defines a function that measures the target's escape risk and tries to minimize the risk function, subject to the stealth constraint, in order to achieve the dual objectives of tracking and stealth. The algorithm is efficient, taking $O(n)$ time at each time step, where n is the complexity of the tracker's visibility region. Experiments in simulation show that the algorithm performs well in difficult environments.

For future work, we plan to test our algorithm on real robots to further validate the approach. We also plan to introduce a limited amount of global information to improve tracking performance, due to the limitations of purely local information, as we have discussed in Section 7. With global information, more sophisticated tracking strategies can be applied. For example, instead of following a target in a long corridor, which leads to high risk of exposure, the tracker can simply move to the other end of the corridor and wait there for the target. Global information also allows us to use multiple trackers to further improve performance.

Acknowledgements We thank Ho-Lun Cheng and Jean-Claude Latombe for helpful discussions and thank Hector González-Baños for providing the image in Figure 1.

References

1. J. Barraquand and J.C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10(2-4):121–155, 1993.
2. C. Becker, H.H. González-Baños, J.C. Latombe, and C. Tomasi. An intelligent observer. In *Int. Symp. on Experimental Robotics*, pages 153–160, 1995.
3. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, 2nd edition, 2000.
4. B.R. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066, 1993.
5. K. Donald. *Optimal Control Theory: An Introduction*. Prentice Hall, Englewood Cliffs, NJ, 1970.
6. A. Efrat, H.H. González-Baños, S.G. Kobourov, and L. Palaniappan. Optimal strategies to track and capture a predictable target. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 3789–3796, 2003.
7. H. ElGindy and D. Avis. A linear algorithm for computing the visibility of polygon from a point. *J. Algorithms*, 2:186–197, 1981.
8. H.H. González-Baños, C.Y. Lee, and J.C. Latombe. Real-time combinatorial tracking of a target moving unpredictably among obstacles. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 1683–1690, 2002.
9. S.A. Hutchinson, G.D. Hager, and P.I. Corke. A tutorial on visual servo control. *IEEE Trans. on Robotics & Automation*, 12(5):651–670, 1996.
10. M. Isard and A. Blake. Condensation—conditional density propagation for visual tracking. *Int. J. Computer Vision*, 29(1):5–28, 1998.
11. B. Joe and R.B. Simposon. Corrections to Lee's visibility polygon algorithm. *BIT*, 27:458–473, 1987.
12. J.C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.
13. S.M. LaValle, H.H. González-Baños, C. Becker, and J.C. Latombe. Motion strategies for maintaining visibility of a moving target. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 731–736, 1997.
14. D.T. Lee. Visibility of a simple polygon. *Computer Vision, Graphics, & Image Processing*, 22:207–221, 1983.
15. T.Y. Li and T.H. Yu. Planning tracking motions for an intelligent virtual camera. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 1353–1358, 1999.
16. R. Murrieta-Cid, González-Baños, and B. Tovar. A reactive motion planner to maintain visibility of unpredictable targets. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 4242–4248, 2002.
17. S. Thrun. Probabilistic algorithms in robotics. *AI Magazine*, 21(4):93–109, 2000.