Real-time Combinatorial Tracking of a Target Moving Unpredictably Among Obstacles

Héctor H. González-Baños Cheng-Yu Lee Jean-Claude Latombe

Abstract— **Many applications require continuous monitoring of a moving target by a controllable vision system. Although the goal of tracking objects is not new, traditional techniques usually ignore the presence of obstacles and focus on imaging and target recognition issues. For a target moving among obstacles, the goal of tracking involves a complex motion problem: a controllable observer (e.g., a robot) must anticipate that the target may become occluded by an obstacle and move to prevent such an event from occurring. This paper describes a strategy for computing the motions of a mobile robot operating in a 2-D workspace without prior knowledge of the target's intention or the distribution of obstacles in the scene. The proposed algorithm governs the motion of the observer based on current measurements of the target's position and the location of the local obstacles. The approach is combinatorial in the sense that the algorithm explicitly computes a description of the geometric arrangement between the target and the observer's visibility region produced by the local obstacles. The algorithm computes a continuous control law based on this description. The new tracking strategy has been implemented in a real-time robotic system.**

Keywords— **Target tracking, visibility constraints, robotics, autonomous observers, escape paths.**

I. Introduction

Various types of applications may benefit from mobile sensors capable of autonomously monitoring targets moving unpredictably in environments cluttered by obstacles. For instance, in [1], [2] a robot equipped with a camera —called an "autonomous observer" (AO)— helps geographically distributed teams debug robotic software. The AO continuously tracks a second robot (the target) executing an independent task. The information acquired by the AO is sent over to remote workstations, where a 3-D graphic rendering of the target and its environment allows the programmers to detect and correct bugs in the target's software. Target-tracking techniques in the presence of obstacles have also been also proposed for the graphic animation of digital actors, in order to select the successive viewpoints (positions of a virtual camera) under which an actor is to be displayed as it moves in its environment [3]. In surgery, controllable cameras could keep a patient's organ or tissue under continuous observation, despite unpredictable motions of potentially obstructing people and instruments. In an airport, mobile robots available to travelers for carrying bags could autonomously follow the displacements of their "clients." The military domain offers many other potential applications as well. As noted in [4], a key distinction between the above applications and standard tracking problems (e.g., missile control, pure visual tracking [5], [6], [7]) is the introduction of obstacles that occlude the field of view of the sensor and obstruct the motions of the sensor and the target. The sensor must then use its ability to move to prevent undesirable occlusions from happening.

In this paper, we consider the case where a controllable vision sensor is mounted on an indoor mobile robot acting as the observer. It is assumed that any obstacle in the environment that obstructs the field of view of the sensor also constrains the motions of both the robot and the target, and vice-versa (no transparent objects or smoke screens). The target (as well as the observer) operate on a plane, but the trajectory of the target is not known in advance. Furthermore, no prior model (map) of the environment is available. These constraints imply that off-line techniques cannot be used. Moreover, the lack of a prior map severely limits the use of pre-computed calculations in order to enhance the on-line execution of the algorithm. The robot must rely upon a "local" map computed on-the-fly from the measurements produced by its sensors.

Our on-line algorithm redirects the observer (i.e. the robot) several times per second based on the differential change in a measure ϕ of the risk that the target escapes its field of view. ϕ is computed deterministically, and is a function of how quickly can the target escape and how easy it is for the observer to react to such an event. For example, if the target can escape by moving around a corner, ϕ grows with the distance between the observer and such corner. This encodes the fact that an occlusion is difficult to clear if the observer is far away from the corner producing it.

The evaluation of the change in ϕ at each time step is based on two key geometric computations. One computation yields the observer's visibility region in the environment; this region is constrained by the sensor's characteristics (e.g., minimal and maximal range), the observer's position, and the view-obstructing obstacles. The second computation is the construction of a *escape-path tree* (EPT), which is a data structure containing all the locally worst-case paths that the target may use to escape the observer's visibility region.

In general, a differential change in the position of the observer produces a differential change in the structure of the EPT, which in turn produces a differential change in the risk ϕ. This change can be computed analytically, and thus the gradient of ϕ can be used

Héctor H. González-Banos is with Honda's Fundamental Research Labs, in Mountain View, CA. E-mail: hhg@hra.com

Cheng-Yu Lee and Jean-Claude Latombe are with the CS Robotics Laboratory, Stanford University, in Stanford, CA. E-mail: {chengyu,latombe}@robotics.stanford.edu

to direct the motion of the robot. The result is not only an on-line strategy, but a *differential* one (i.e., a feedback controller).

The rest of the paper is organized as follows: Section II gives a formal statement of the tracking problem and describes previous work on this topic. Section III defines escape paths and their connection to the tracking problem. The notion of escape risk as a tracking criterion is also introduced here. In Section IV, we present an algorithm for target tracking that does not require a prior map. Section V describes the implementation of our techniques into a robotic platform and reports on the experiments with this system. Finally, Section VI suggests topics for future research.

II. Problem Formulation and Background

Below, we follow the mathematical formulation used in [4].

Suppose the observer and the target move in a bounded Euclidean subspace $\mathcal{W} \subset \Re^2$ (the workspace). The observer and the target are assumed to be rigid bodies, and their free configuration spaces are denoted $\mathcal{C}^o$ and $\mathcal{C}^t$, respectively. Let $\mathcal{X}$ be the state space of the problem, which is the Cartesian product of the individual state spaces of both the observer and the target. The Cartesian product $\mathcal{C}^o \times \mathcal{C}^t$ is equal to $\mathcal{X}$ in the absence of dynamics. In general, however, $\mathcal{C}^o \times \mathcal{C}^t$ is a subspace of the state space.

Define $\boldsymbol{q}^o(t) \in \mathcal{C}^o$ as the observer's configuration at time t, and $\boldsymbol{x}^o(t)$ as its state. Let f^o be the transition equation for the states of the observer: $\dot{\boldsymbol{x}}^o(t) = f^o(\boldsymbol{x}^o, \boldsymbol{u})$, where $\boldsymbol{u}(t)$ is the control or action selected from a control set $\mathcal{U}$ at time t. The function f^o models the observer's dynamics, and may encode nonholonomic restrictions or other type of constraints.

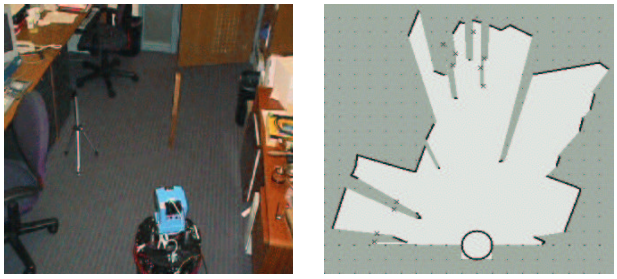
Similarly, let $\boldsymbol{q}^t(t) \in \mathcal{C}^t$ be the configuration of the target at time t and $\boldsymbol{x}^t(t)$ its state. The transition equation for the target is given by $\dot{\boldsymbol{x}}^t(t) = f^t(\boldsymbol{x}^t, \boldsymbol{\theta})$, with the action $\boldsymbol{\theta}(t)$ selected from a target control set Θ. For a reactive target with knowledge about the observer's actions, f^t depends on $\boldsymbol{x}^o(t)$ or $\boldsymbol{q}^o(t)$. An important case, treated extensively in [4], is when the target is predictable. In this case, the target's transition equation simplifies to $\dot{\boldsymbol{q}}^t = f^t(\boldsymbol{q}^t(t))$.

Together, f^o and f^t define a state transition equation $\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\theta})$, where $\boldsymbol{x}(t) = (\boldsymbol{x}^o(t), \boldsymbol{x}^t(t))$. The state can be mapped into a configuration pair by some function $(\boldsymbol{q}^o, \boldsymbol{q}^t) = H(\boldsymbol{x})$, where $H : \mathcal{X} \to \mathcal{C}^o \times \mathcal{C}^t$ is a mapping that is not injective in general.

A. Visibility regions

The workspace geometry plays an important role in the target-tracking problem. In addition to obstructing motion, obstacles determine the area of the workspace that is visible to the observer.

The observer's configuration determines the field of view of its sensors. Let $\mathcal{V}(\boldsymbol{q}^o) \subseteq \mathcal{W}$ be the set of all locations where the target is visible to an observer located at $\boldsymbol{q}^o$. The set $\mathcal{V}(\boldsymbol{q}^o)$ is the *visibility region* at the observer position $\boldsymbol{q}^o$ and can be defined in a number of ways [8]. For example, the observer may have a 360-deg field of view and the target may be a point in $\mathcal{W}$. In this case, the target is said to be visible *iff* the line-of-sight to the observer is un-obstructed. In other examples, visibility may be limited to some fixed cone or restricted by lower- and upper-bounds on the sensor range.



(a) (b)

Fig. 1. Measuring the visibility region with a laser range-finder.

The visibility region can be computed from a synthetic model or from sensor measurements. In the former case, a ray-sweep algorithm can be used to compute this region for polygonal models (see [9] for a survey of methods). For the latter, the visibility region can be measured with a laser range-finder using the technique proposed in [10] (see Fig. 1).

B. Tracking strategies

In essence, the target-tracking problem consists of computing a function $\boldsymbol{u}^*(t)$ — a *strategy* — such that the target remains in view for all $t \in [0, T]$ (where T is the *horizon* of the problem). Additionally, it may be important to optimize additional criteria based on the total distance traversed by the observer, the distance to the target, or a quality measure of the visual information. Sometimes losing track of the target is unavoidable, in which case an optimal strategy may consist of maximizing the target's *escape time* (t_{esc}) — the time when the observer first loses the target.

If the target action $\boldsymbol{\theta}(t) \in \Theta$ is known in advance for all $t \leq T$, then the target is said to be *predictable*. In this case, the optimal strategy can be calculated *off-line* before the observer begins to track the target. Because the location of the target is known for all t, it is possible to re-acquire the target when it is lost. Therefore, for cases where it is impossible to track the target for all $t \leq T$, we may instead maximize the *exposure* — the total time the target is visible to the observer — as an alternative to maximizing the escape time.

If $\boldsymbol{u}^*(t)$ is computed as a function of the state $\boldsymbol{x}(t)$, then the strategy operates in *closed loop*. Otherwise, the strategy runs in *open loop*. Closed-loop strategies are preferred over open-loop ones even for the predictable case, unless there is an absolute guarantee that the motion models and position measurements are exact (e.g., as is the case in [3]).

When the target actions are unknown, the target is said to be *unpredictable*. This is a significantly more complex problem. Following the framework proposed in [11], the unpredictable case can be analyzed in two ways: If the target actions are modeled as *nondeterministic uncertainty*, then it assumed that we know

Θ but not a specific $\boldsymbol{\theta}(t) \in \Theta$. That is, we know the action set but not the specific action selected by the target. In this case, one can design a strategy that performs the best given the worst-case choices for $\boldsymbol{\theta}(t)$. Alternatively, if a *probabilistic uncertainty* model is available — i.e., the probability density function $p(\boldsymbol{\theta}(t))$ is given — then it is possible to compute a motion plan that is the best in the expected sense.

In any event, the unpredictable case has to be solved on-line. Unless there is a mechanism for re-acquiring the target [12], a good tracker seeks to maximize t_{esc} as opposed to maximize the exposure. A strategy designed for the worst-case scenario will anticipate the target's most adverse action for a future horizon T, execute a small (possibly differential) initial portion of the computed strategy, and repeat the entire process again [2]. On the other hand, a strategy designed to anticipate the expected target's action will seek to maximize t_{esc} by maximizing the probability of future target visibility [1], [4], [13].

C. Robot localization issues

In practice, the tracking problem is often interwoven with that of robot self-localization. This is true when the tracking algorithm uses a prior map of the environment to calculate the observer's actions. Self-localization is typically done by using landmarks. In the systems described in [1], [2], the landmarks are artificial ceiling landmarks scattered through the workspace. The observer localizes itself with good precision if a landmark is visible; otherwise, it navigates by dead-reckoning and the observer's position uncertainty increases until the next landmark observation.

The techniques in [1], [2], [4] do not explicitly acknowledge the need for the observer to see landmarks for self-localization. This issue is specifically addressed in [14], where the need to re-localize is explicitly considered by the tracker. The algorithm decides at each stage whether it is preferable to perform the best tracking motion, or deviate from this motion in order to see a landmark and achieve better self-localization.

Of course, if the tracker does not require a prior map and makes all its decisions based on a local map computed from its sensor inputs, then the localization problem is solved implicitly and no other self-localization mechanism is required. This is the design philosophy behind the system described in Section V.

III. Escape paths and Escape Time

Under the line-of-sight visibility model, the region $\mathcal{V}(\boldsymbol{q}^o)$ inside a polygonal workspace is also a polygon (a star polygon in fact). This visibility polygon has linear complexity (see [9]), and its boundary is composed of *solid* and *free* edges. A solid edge represents an observed section of the workspace (i.e., it is part of a physical obstacle). A free edge is caused by an occlusion, and it is contained inside the workspace (i.e., it is an element of $\mathcal{C}_{free} \subseteq \mathcal{W}$). Fig. 2 shows the free and solid edges for an example of a visibility region.

Suppose that the target is visible to the observer (i.e., $\boldsymbol{q}^t \in \mathcal{V}(\boldsymbol{q}^o)$). An *escape path* for a target located at $\boldsymbol{q}^t$ is any collision-free path connecting $\boldsymbol{q}^t$ to a point in $\mathcal{W}$ outside $\mathcal{V}(\boldsymbol{q}^o)$. The *escape point* of this path is the point where the path intersects the boundary of $\mathcal{V}(\boldsymbol{q}^o)$, which always occur along a free edge. It is clear that there exists an infinite number of escape paths for a particular configuration of target and observer. Note, however, that for a particular escape point there exists a path of minimal length. Moreover, for any free edge e, there exists an escape path of minimal length among all escape points along that edge (see Fig. 2). Such path $\text{SEP}(\boldsymbol{q}^t, e)$ is called the target's *shortest escape path* through the free edge e. The length of $\text{SEP}(\boldsymbol{q}^t, e)$ is the *shortest distance to escape* through e ($\text{SDE}(\boldsymbol{q}^t, e)$).

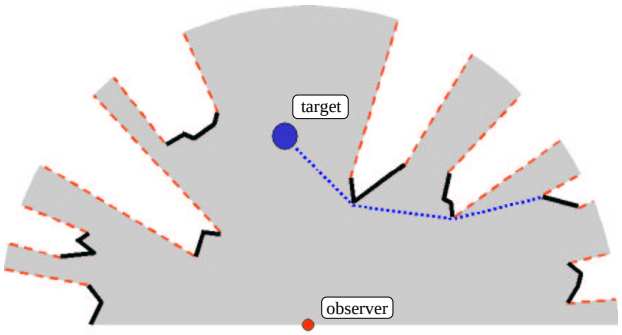


Fig. 2. Visibility region example. Free edges are shown in red (dashed lines) and solid ones in black (solid lines). Also shown is the shortest escape path through one free edge (dotted path).

The shortest time in which the target may traverse an escape path is the *escape time* for that path. Again, for any free edge e and a target location $\boldsymbol{q}^t$, there exists a path of minimal escape time, and in general this is not equal to the $\text{SEP}(\boldsymbol{q}^t, e)$. These two paths are equivalent only if the target is holonomic and has negligible inertia. We reserve the term $t_{esc}(\boldsymbol{q}^o, \boldsymbol{q}^t)$ to denote the minimum escape time among all escape paths leaving $\mathcal{V}(\boldsymbol{q}^o)$ and originating in $\boldsymbol{q}^t$.

Given $\boldsymbol{q}^t \in \mathcal{V}(\boldsymbol{q}^o)$, we can compute $\text{SEP}(\boldsymbol{q}^t, e)\ \forall\ e$ bounding $\mathcal{V}(\boldsymbol{q}^o)$. Thus, if $\mathcal{V}(\boldsymbol{q}^o)$ is bounded by n_f free edges, there are n_f "shortest" escape paths. The shortest $\text{SEP}(\boldsymbol{q}^t, e)$ over all e bounding $\mathcal{V}(\boldsymbol{q}^o)$ is the shortest escape path $\text{SEP}(\boldsymbol{q}^t, \boldsymbol{q}^o)$ for the configuration $(\boldsymbol{q}^o, \boldsymbol{q}^t)$. Its length is $\text{SDE}(\boldsymbol{q}^t, \boldsymbol{q}^o)$. If the target is holonomic and has negligible inertia then $\text{SDE}(\boldsymbol{q}^t, \boldsymbol{q}^o)$ equals $t_{esc}(\boldsymbol{q}^o, \boldsymbol{q}^t)$ multiplied by the target's maximum speed.

A. Properties of escape paths

Suppose $\boldsymbol{q}^t \in \mathcal{V}(\boldsymbol{q}^o)$, and let $l(\boldsymbol{q}^o, \boldsymbol{q}^t)$ be the line passing through the target and the observer. For polygonal workspaces, and assuming the target is a point, the path $\text{SEP}(q^t, e)$ satisfies the following basic properties (stated here without proof):

Property 1: $\text{SEP}(q^t, e)$ is a polygonal line connecting $\boldsymbol{q}^t$ to a point in a free edge e bounding $\mathcal{V}(\boldsymbol{q}^o)$. Each vertex of this polygonal line, if any, is a vertex of $\mathcal{V}(\boldsymbol{q}^o)$.

Property 2: The path $\text{SEP}(\boldsymbol{q}^t, e)$ cannot strictly cross the radial line $l(\boldsymbol{q}^o, \boldsymbol{q}^t)$. The path either lies fully on a single side (right or left) of $l(\boldsymbol{q}^o, \boldsymbol{q}^t)$ or is contained in $l(\boldsymbol{q}^o, \boldsymbol{q}^t)$.

Property 3: The path $\text{SEP}(\boldsymbol{q}^t, e)$ cannot strictly cross any radial line $l(\boldsymbol{q}^o, v)\ \forall\ v \in \mathcal{V}(\boldsymbol{q}^o)$ more than once.

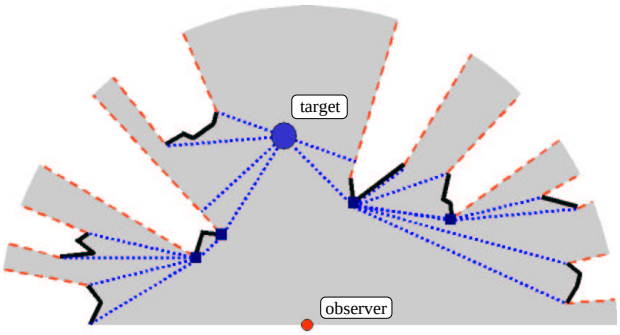


Fig. 3. Escape tree for the example shown in Fig. 2. The tree is shown in blue (dotted lines), and the nodes of the tree are drawn as little squares. The root of the tree is the target.

Let L_l be the list of vertices of $\mathcal{V}(\boldsymbol{q}^o)$ that lie to the left of $l(\boldsymbol{q}^o, \boldsymbol{q}^t)$, sorted in counter-clockwise order. Similarly, define L_r as the list of vertices of $\mathcal{V}(\boldsymbol{q}^o)$ to the right of $l(\boldsymbol{q}^o, \boldsymbol{q}^t)$, sorted in clockwise order. If $l(\boldsymbol{q}^o, \boldsymbol{q}^t)$ passes through a vertex of $V(\boldsymbol{q}^o)$ then let this vertex be included in both L_l and L_r.

The next theorem is the basis of the ray-sweep algorithm in Section IV:

Theorem 1: If the shortest path from $\boldsymbol{q}^t$ to an obstacle vertex v in L_l (or L_r) does not pass through a previous vertex u in L_l (or L_r), then neither the shortest path from $\boldsymbol{q}^t$ to any vertex w appearing after v in L_l (or L_r) passes through u.

Proof: If the shortest path from $\boldsymbol{q}^t$ to w passes through u, then the shortest path from $\boldsymbol{q}^t$ to v will intersect the shortest path from $\boldsymbol{q}^t$ to w at a point other than $\boldsymbol{q}^t$. Therefore, one of the two paths could be made shorter. □

Theorem 1 implies that $\text{SEP}(\boldsymbol{q}^t, e)$ can be constructed incrementally with a ray-sweep algorithm. It is only necessary to remember the shortest path from $\boldsymbol{q}^t$ to the most recently visited obstacle vertex during the scanning of L_l (or L_r).

B. Escape-path trees

Computing escape paths is important because a tracking strategy based on expecting the worst-case scenario assumes that the target will escape by taking the quickest route. One such strategy is to move the observer to a position that (locally) minimizes $\text{SDE}(\boldsymbol{q}^t, \boldsymbol{q}^o)$ [4]. This, of course, assumes that an algorithm to compute the shortest escape path is given [2].

The SDE is a worst-case measure of the likelihood that the target abandons the observer's field of view (the larger the SDE, the more likely will the target remain in view). An alternative is to minimize the average length over all paths $\text{SEP}(\boldsymbol{q}^t, e)$, or optimize a similar function operating over all the individual paths.

If all the escape paths are computed for a configuration $(\boldsymbol{q}^o, \boldsymbol{q}^t)$, these form a tree structure (see Fig. 3). We call this structure the *escape-path tree*. The root of this tree is the target, and each branch in the tree terminates in a free edge. The complexity of this tree is linear, since each node in the tree is a vertex in the visibility polygon (Prop. 1).

It is evident from the tree that many paths share the same initial structure (Fig. 3). This property reveals a fundamental problem with a strategy that minimizes the average distance over all paths $\text{SEP}(\boldsymbol{q}^t, e)$. Escape paths along the same branch are over-represented by taking the global average at the expense of solitary escape paths. This is often the case in an implementation, where chairs, table legs and similar small obstacles produce many escape paths along the same branch. In our system, we lessened this problem by computing a recursive average from the tree's children backwards into the parent node. Children of the same node are first averaged between each other, and the result is then back-propagated to the previous node.

C. Tracking by minimizing the escape risk

Solving the target-tracking problem on-line can be computationally expensive. In practice, it is usually necessary to settle for strategies that plan for a small time horizon Δt in order to have a sufficiently fast algorithm.

The algorithms in [2], [4] consists of discretizing the problem in stages of some small duration Δt. For a given stage k, the algorithm finds a control action $\boldsymbol{u}_k$ by solving the following equation:

$$\boldsymbol{u}_k^* \quad = \quad \arg \sup_{\boldsymbol{u}_k \in \mathcal{U}} t_{esc}(\boldsymbol{q}_{k+1}^o(\boldsymbol{q}_k^o, \boldsymbol{u}_k), \boldsymbol{q}_k^t), \qquad (1)$$

where the target is assumed to remain at the same location until the next stage. The key in Eqn.(1) is the calculation of t_{esc} (an expensive computation).

For any target under kino-dynamic constraints, t_{esc} is upper bounded by a factor proportional to the SDE, which is a lot easier to compute. In [2], the solution to Eqn.(1) is approximated by solving:

$$\boldsymbol{u}_k^* \quad = \quad \arg \sup_{\boldsymbol{u}_k \in \mathcal{U}} \text{SDE}(\boldsymbol{q}_{k+1}^o(\boldsymbol{q}_k^o, \boldsymbol{u}_k), \boldsymbol{q}_k^t), \qquad (2)$$

which essentially uses the SDE as a proxy function of the escape time. In practice, this strategy produces poor results except for simulated experiments and holonomic observers without dynamics.

There are two problems with Eqn.(2). One is due to the nature of the SDE function. As the observer moves, new occlusions form and old ones disappear, and new paths become the shortest escape path. As a result, the value of $\boldsymbol{u}_k^*$ changes abruptly from one stage to the next producing a *shattering* effect on the control signal [15]. Un-modeled observer dynamics will be excited by a shattering signal, producing very erratic and unpredictable observer motions.

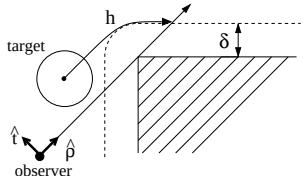
The second problem is that the SDE is not a good proxy function for t_{esc}. The relationship between SDE and t_{esc} is not linear. In fact, a large SDE makes it increasingly harder for the target to escape. To understand this, imagine a situation where the SDE becomes increasingly larger. Tracking becomes easier because the target has to travel a longer distance in order to escape *and* the observer has time to improve its future ability to track.

In this paper, we attempt to solve both problems with a new proxy function for t_{esc}. This function is the *escape risk*, defined for every free edge e as follows:

$$\phi^e \quad = \quad c\, {r_o}^2 \left(\frac{1}{h}\right)^{m+2}, \qquad (3)$$

where $h = \text{SDE}(e, \boldsymbol{q}^t)$, r_o is the distance from $\boldsymbol{q}^o$ to the corner causing the occlusion at e, d is the distance between $\boldsymbol{q}^t$ to this corner, $c > 0$ is a constant and $m \geq 0$ is a given integer. We call the numerator of the fraction the *look-ahead* component, because its minimization increases the future ability of the observer to track. The denominator is the *reactive* component, because its maximization decreases the likelihood that the target escapes the current visibility region.

It is possible to compute the gradient of Eqn.(3) with respect to the observer's position in closed form. The gradient is given by different formulas depending on the way the escape path $\text{SEP}(e, \boldsymbol{q}^t)$ exits $\mathcal{V}(\boldsymbol{q}^o)$ (e.g., through a vertex of $\mathcal{V}(\boldsymbol{q}^o)$ or through a corner). For instance, in the following arrangement (for $m = 0$):



,

the gradient is:

$$\nabla\phi^e = \left[2\, c\, r_o \left(\tfrac{1}{h}\right)^2 \hat{\boldsymbol{\rho}}\,, \quad -2\, c\, \delta\, r_o \left(\tfrac{1}{h}\right)^3 \hat{\boldsymbol{t}}\right]. \qquad (4)$$

Here $(\hat{\boldsymbol{\rho}}, \hat{\boldsymbol{t}})$ is a coordinate system attached to the observer, and δ is the target's radius (assuming it is a circle). Other cases are dealt similarly, but they are omitted for the sake of space. It is important to note that the gradient computation can be degenerate when the target is a point. But when the target is a circle with $\delta > 0$, the gradient exists for all cases. Because $\nabla\phi^e$ is computed analytically, only current sensor information is required.

Our tracking strategy consists of moving the observer in the opposite direction of the average of $\nabla\phi^e$ over all free edges e in $\mathcal{V}(\boldsymbol{q}^t)$. Let $\nabla\bar{\phi}$ denote this average. $\nabla\bar{\phi}$ can be computed using the escape-path tree as explained before. However, other ways of aggregating the individual risks are possible.

IV. A Combinatorial Target Tracker

We now describe our target-tracking algorithm. Its basic structure is the following:

Algorithm TRACK (no prior map):
Repeat:
1. Extract a local map from measurements.
2. Determine the target's position.
3. Compute the escape-path tree.
4. Compute $\nabla\phi^e$ for each free edge e in the map.
5. Compute $\nabla\bar{\phi}$ using the escape-path tree.
6. Steer robot using $-\nabla\bar{\phi}$ (robot control).

The algorithm is presented sequentially for clarity purposes. In an actual implementation it will be more efficient to intermingle some of these steps.

Steps 1, 2 and 6 are implementation dependent (see next section). Step 4 consists on evaluating the gradient of Eqn.(3) for every escape path. Step 5, as explained before (Section III-B), consists of computing a recursive average of the risks of the various paths in the escape path tree.

Step 3 is the missing piece. In order for this algorithm to work in real-time, the computation of the escape-path tree has to be done efficiently. This can be accomplished with a ray-sweep algorithm.

A. Computation of escape paths using ray-sweep

Suppose $\mathcal{V}(\boldsymbol{q}^o)$ is represented by a list of vertices ordered counter-clockwise. We split its contents into the lists L_l and L_r, where L_l is the list of all vertices of $\mathcal{V}(\boldsymbol{q}^o)$ to the left of $l(\boldsymbol{q}^o, \boldsymbol{q}^t)$, and L_r is the list of all vertices to the right of $l(\boldsymbol{q}^o, \boldsymbol{q}^t)$. We reverse the order of L_r so its contents are ordered clockwise.

The ray-sweep algorithm consists of computing the shortest path from $\boldsymbol{q}^t$ to every vertex in $\mathcal{V}(\boldsymbol{q}^o)$ by performing a sequential scan of L_l followed by a similar scan on L_r. We describe here the scan for L_l.

The algorithm visits each $v_i \in L_l$ and updates a *pivot* list π_i: the list of vertices that define the shortest path from $\boldsymbol{q}^t$ to $v_i \in L_l$. The update operation is as follows:

Pivot List UPDATE:
Repeat until `size_of`$(\pi_i) < 3$ or `Step` 2 fails:
1. Let u_{r-1}, u_r, and u_{r+1} be the last 3 elements in π_i, with $u_{r+1} = v_i$.
2. If u_{r+1} lies to the right of the line (u_{r-1}, u_r) then remove u_r from π_i.

The above algorithm derives from the fact that once a vertex is no longer a member of an escape path, it will never become one again. This is a consequence of Theorem 1.

A path $\text{SEP}(\boldsymbol{q}^t, e)$ is computed from a pivot list π_i at the currently visited vertex v_i. There are three mutually exclusive cases for v_i and the algorithm acts differently in each case:
1. If v_i isn't in a free edge then π_i isn't an escape path.
2. If v_i is an endpoint of a free edge and the segment (v_{i-1}, v_i) is an obstacle edge, then π_i represents a new escape path $\text{SEP}(\boldsymbol{q}^t, e)$.
3. If v_i is an endpoint of a free edge, but the segment (v_{i-1}, v_i) lies in free space, then it might be possible to shorten the newly-found escape path by displacing the escape point along the free edge preceding v_i. This can be easily calculated in constant time.

B. Run-time analysis

Each vertex in L_l and L_r is appended to the pivot list exactly once. At the same time, each removed vertex is never re-inserted into the list. Hence, if the input list representing $\mathcal{V}(\boldsymbol{q}^o)$ is pre-sorted, the computational cost of the ray-sweep algorithm is proportional to the number of vertices stored in L_l and L_r. The cost for computing all the escape paths is thus $\text{O}(n)$. This is also the cost for computing the escape-path tree, since each node in the tree is a vertex in $\mathcal{V}(\boldsymbol{q}^o)$.

V. Implementation and Experiments

We implemented our tracking strategy on a Nomad SuperScout robot. The robot is equipped with a laser sensor from Sick OpticElectronic. This sensor measures distances to objects in the workspace based on a time-of-flight technique. Measurements are done along evenly-spaced rays in a horizontal plane. The sensor operates at a frequency of 32 scans/s, and each scan consists of 360 points with 0.5-deg spacing, for a total field of view of 180 deg. The sensor's range is 8 meters. A Nomad 200 robot acts as the target.

Our visibility region is limited to the half-plane in front of the robot. Since our tracking algorithm relies only on the local map to generate a new motion, the observer cannot move backward without risking a collision. Therefore, in our experiments we restricted the target to move away from the observer. This assumption, however, can be easily removed by mounting two range finders on the observer.

The software runs in a 410 MHz Pentium II workstation, with the exception of the low-level robot and sensor drivers (running on-board the robot). Sensory data is transmitted to the workstation through an IEEE 802.11b wireless link. The workstation computes the local visibility polygon as well as the escape-path tree, followed by a velocity command (v, ω) that is then sent to the robot for execution. The software was developed in C^{++} under Linux using functions from the LEDA 3.8 library [16]. The implemented TRACK algorithm operates at approximately 10 Hz.

The implementation is entirely based on the methods described previously, with the exception of a few additional functions listed below.

A. Some important peripheral functions

The algorithm TRACK requires a few support functions in order to run: local map construction, target detection, and robot control.

Local map construction

The sensor readings are captured sequentially and stored into a list L_p of points in the reference frame local to the sensor. The technique described in [10] is used to transform L_p into a collection of polylines.

The technique first segments L_p into sub-lists by detecting gaps between successive points, and fits a polyline to each of the sub-lists. There are two kinds of gaps: occlusion and out-of-range gaps. An occlusion gap occurs whenever two successive points in L_p are further apart than a certain threshold. Out-of-range gap correspond to a series of consecutive measurements in L_p that have saturated to the sensor's maximum range. Gaps are detected as L_p is captured.

The visibility region $\mathcal{V}(\boldsymbol{q}^o)$ is obtained by closing the contour formed by the computed polylines. Two consecutive polylines are connected by an occlusion edge if they are separated by an occlusion gap. If they are separated by an out-of-range gap, they are connected by a sequence of three free edges: an occlusion edge, a range edge, and another occlusion edge.

Target detection

Our detection algorithm is deliberately simple. A Nomad 200 is a fairly cylindrical object. We detect its shape by finding a cluster of points in L_p that matches the target's circular contour. If the algorithm identifies several sequences of points that can plausibly match the target, it retains the one that provides the best fit.

The target creates a shadow in the local visibility region. For the effect of computing the escape paths in $\mathcal{V}(\boldsymbol{q}^o)$, we assume that no obstacle lie in this shadow.

Robot control

The gradient $-\nabla\bar{\phi}$ should be interpreted as the desired direction of the observer's motion. The speed of the observer should be proportional to $|\nabla\bar{\phi}|$.

A *trajectory-follower* [15] can be used to generate observer trajectories that follow $-\nabla\bar{\phi}$. The choice of a trajectory-follower depends on the dynamic model of the observer. In our implementation, we assume a very simple kinematic model for a non-holonomic observer:

$$\dot{x} = v\cos(\theta), \quad \dot{y} = v\sin(\theta), \quad \dot{\theta} = \omega. \tag{5}$$

Here, v is the translational speed of the observer and ω is the angular speed.

A crude but effective way of steering the robot is to use the following controller:

$$v = k_v\ \bar{n}\cdot(-\nabla\bar{\phi}), \qquad \omega = k_\omega \frac{\bar{n}\times(-\nabla\bar{\phi})}{|\nabla\bar{\phi}|}, \tag{6}$$

where k_v and k_ω are positive constants, and $\bar{n} = [\cos(\theta), \sin(\theta)]$ is a unit vector representing the current direction of the observer. One can prove that for a slow-varying $\nabla\bar{\phi}$ this controller is stable: v and $\tan(\alpha)$ converge exponentially to a steady-state (where α is the angle between $\bar{n}$ and $-\nabla\bar{\phi}$).

Eqn.(6) has to be complemented with a second regulator to maintain the target at a preferred distance to the observer. This is required because we ignored the sensor's range and angular restrictions in our current implementation of $\nabla\bar{\phi}$. This second regulator is a simple proportional feedback of the target's deviation from the center of the field of view, and would be unnecessary if the computation of $\nabla\bar{\phi}$ explicitly acknowledged the sensor's range limits.

B. Experiments

We present here the results of three experiments: an example of the influence of the escape-path tree in the computation of $-\nabla\bar{\phi}$ (*chair example*), a test of the transient response of the target tracker (*cardboard box example*), and a long tracking tour around the Robotics Lab. at Stanford University (*tour example*).

Chair example

In this example, the observer is surrounded by several chairs (Fig. 4), and is forced to remain motionless while the target is parked in the background. One of the chairs was pushed past the target and toward the observer, and we observed the change that this produces on $-\nabla\bar{\phi}$. The moving chair is enclosed with a red square in the pictures, and its corresponding shape in $\mathcal{V}(\boldsymbol{q}^o)$ is the shaded region in the plots.

From the bottom plot in Fig. 4, it is clear that most of the escape paths to the left of the chair share a common branch. If $-\nabla\bar{\phi}$ is computed using the average of $-\nabla\phi^e$ over all free edges e in $\mathcal{V}(\boldsymbol{q}^o)$ this results on the vector shown in Fig. 5(a). The problem with this vector is that it points in a direction behind the chair.

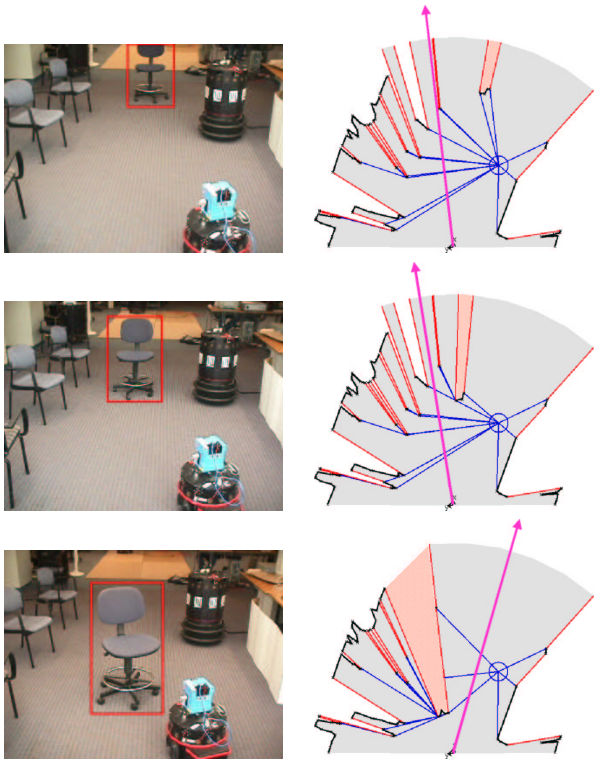


Fig. 4. Evolution of an escape-path tree for a scene with chairs. The long vector shown in the figures to the right is $-\nabla\bar{\phi}$.

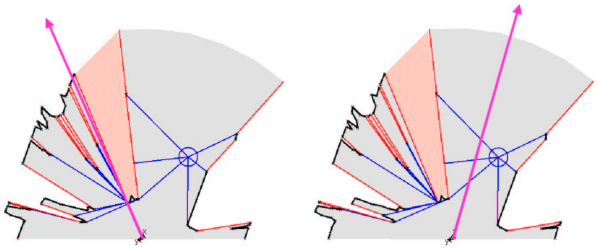


Fig. 5. Example of $\nabla\bar{\phi}$ computation: (a) standard average; (b) average using the escape-path tree.

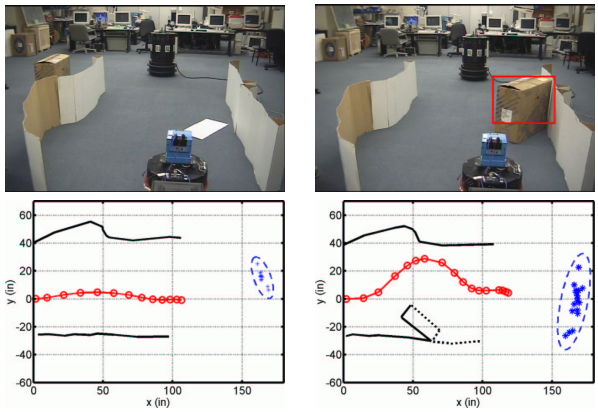
This occurs because the escape paths to the left of the chair are over-represented.

Fig. 5(b) shows the value of $-\nabla\bar{\phi}$ computed as the recursive average over the nodes of the escape-path tree. $-\nabla\phi^e$ is first averaged among all the branches to the left of the chair before it is averaged with those to the right. The result points in the desired direction.

Cardboard box example

This example was used to test the transient response of the target tracker. The scenes shown in Fig. 6(a) and Fig. 6(b) differ in that a cardboard box is present in the latter but not in the former. In both experiments the observer was initially located at a distance of 165 in. aiming towards a stationary target, and the tracking program was activated afterwards.

The tracking paths for both scenarios are shown with Matlab plots in Fig. 6. The plot in (a) is a very



(a) (b)

Fig. 6. Scenes (a) and (b) differ by the presence of a cardboard box. In (b), the observer must swerve around the box in order not to lose the target. Both plots show the walls detected by the laser (black curves), and the noise of the target detector (the scattered points to the right of each plot). Scale in inches.

straight-forward path, but not the one in (b). In (b), a straight path becomes a high-risk strategy due to the presence of the box. Therefore, the observer should swerve around the box in order to decrease the risk, and this maneuver must occur almost immediately after the observer becomes aware of its situation. The position data was captured from the observer's encoders. The walls detected by the sensor are shown as solid curves in the plots.

The noise of the target detector is also shown in Fig. 6 (the scattered points to the right of each plot). It is interesting to note that the tracker's motion is relatively smooth in spite of this noise. We reached the empirical conclusion that a precise detection algorithm is not extremely important, as long as the detector runs at a fast rate. More critical is the rate of false positives (instances when the detector incorrectly identifies a target as such). A long stream of false positives will confuse the tracker.

Tour example

An experimental run is shown in Fig. 7. The tracker followed the target through the Robotics Lab. at Stanford U. The path is long, and only snapshots are shown in the figure. The tour started outside the office of one of the co-authors. The observer chased the target down the North corridor of the lab, through a lounge area cluttered with chairs, and into one of the offices in the South corridor.

The accumulated paths for both the observer and the target are shown with Matlab plots in Fig. 7. The path for the observer is drawn with red circles and the one for the target with blue triangles. The observer's path is plotted using the data from the encoders. The target's path is computed from the output of the target detector in combination with the observer's encoders.

Video of the experiments The tour example is difficult to appreciate in pictures. Please visit our web-site at `http://underdog.stanford.edu/` to see a video of this experiment and other examples.

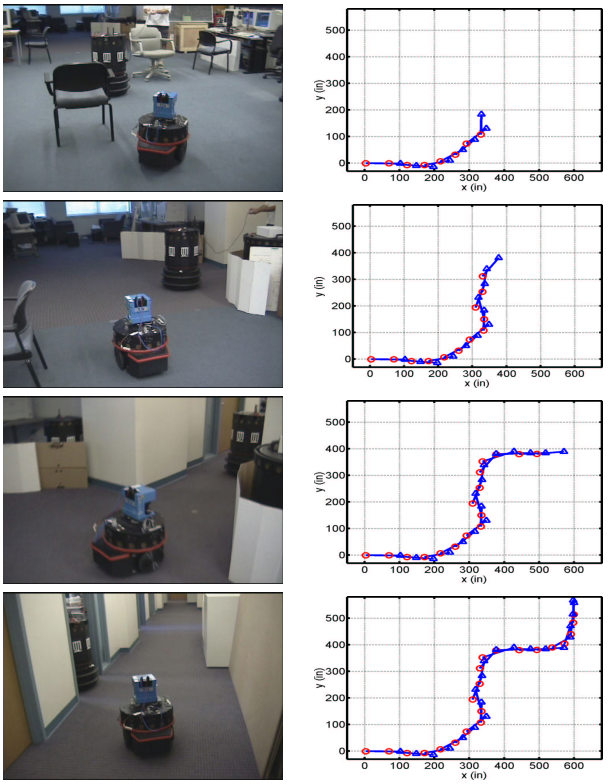


Fig. 7. Tracking the target around the Stanford Lab. The path of the observer is shown in red (circles) and the one for the target in blue (triangles). Scale in inches.

VI. Conclusion

Several applications require the continuous tracking of a target moving in a cluttered environment. This paper introduces a new tracking algorithm for the case when the target moves unpredictably and no prior map of the environment exists.

Our algorithm computes a motion strategy based exclusively on current sensor information — no global map or historical sensor data is required. The algorithm is based on the notion of escape risk and the computation of an escape-path tree. This tree is a data structure storing the most effective escape routes that a target may follow in order to escape the observer's field of view. This paper also shows how an escape-path tree can be computed in linear time from range-finder data using a ray-sweep technique.

We have implemented an experimental robotic observer equipped with a range finder as its sole sensor. Our experiments show that the observer is able to keep a moving target in view by continuously steering in the direction minimizing the escape risk. Most failures were due to the shortcomings of our simple target detector. An improvement would be to equip the observer with an additional vision system to make target detection and localization more reliable and precise.

Several interesting extensions and variations can be considered in future investigations. Currently, our theory is limited to a 2-D workspace. In the future, we would like to extend our algorithm to consider problems in 3-D space. This will have an immediate impact in several applications. Another extension is to integrate our target-tracking technique with the map-building system described in [17]. This hybrid system will map the environment while the target is being tracked. The computed map could then be used to enhance the behavior of the observer by extending its reasoning beyond the local vicinity.

Acknowledgements We wish to thank Vicky Chao for her help in running the experiments and capturing the results on video.

References

[1] H. González-Banos, J.L. Gordillo, D. Lin, J.C. Latombe, A. Sarmiento, and C. Tomasi, "The autonomous observer: A tool for remote experimentation in robotics," in *Telemanipulator and Telepresence Technologies VI*, Matthew Stein, Ed. November 1999, vol. 3840, SPIE Proc.

[2] H. González-Banos, *Motion Strategies for Autonomous Observers*, Ph.D. thesis, Stanford University, March 2001.

[3] T.Y. Li, J.M. Lien, S.Y. Chiu, and T.H. Yu, "Automatically generating virtual guided tours," in *Proc. of the 1999 Computer Animation Conference*, May 1999, pp. 99–106.

[4] S.M. LaValle, H. González-Banos, C. Becker, and J.C. Latombe, "Motion strategies for maintaining visibility of a moving target," in *Proc. 1997 IEEE Int'l Conf. Robotics & and Automation*, April 1997, pp. 731–736.

[5] S. Hutchinson, G. D. Hager, and P. I. Corke, "A tutorial on visual servo control," *IEEE Trans. Robotcis & Automation*, vol. 12, no. 5, pp. 651–670, Oct. 1996.

[6] D.P. Huttenlocher, J.J Noh, and W.J. Rucklidge, "Tracking non-rigid objects in complex scenes," in *Proc. 4th Int. Conf. on Computer Vision*, 1993, pp. 93–101.

[7] N.P. Papanikolopoulos, P.K. Khosla, and T. Kanade, "Visual tracking of a moving target by a camera mounted on a robot: A combination of control and vision," *IEEE Trans. Robotics & Automation*, vol. 9, no. 1, pp. 14–35, Feb. 1993.

[8] T. Shermer, "Recent results in art galleries," *Proc. IEEE*, vol. 80, no. 9, pp. 1384–1399, Sept. 1992.

[9] J. O'Rourke, "Visibility," in *Handbook of Discrete and Computational Geometry*, J.E. Goodman and J. O'Rourke, Eds., pp. 467–479. CRC Press, Boca Raton, FL, 1997.

[10] H. González-Banos and J.-C. Latombe, "Robot navigation for automatic model construction using safe regions," in *Lecture Notes in Control and Information Sciences, 271*, D. Russ and S. Singh, Eds. 2001, pp. 405–415, Springer.

[11] S. M. LaValle, *A Game-Theoretic Framework for Robot Motion Planning*, Ph.D. thesis, University of Illinois, Urbana, IL, July 1995.

[12] L.J. Guibas, J.C. Latombe, S.M. Lavalle, D. Lin, and R.Motwani, "Visibility-based pursuit-evasion problem," *Int. J. of Computational Geometry and Applications*, vol. 9, no. 5, pp. 471–494, 1999.

[13] C. Becker, H. González-Banos, J.-C. Latombe, and C. Tomasi, "An intelligent observer," in *Proc. 4th International Symposium on Experimental Robotics*, 1995, pp. 153–160.

[14] P. Fabiani and J.C. Latombe, "Dealing with geometric constraints in game-theoretic planning," in *Proc. Int. Joint Conf. on Artif. Intell.*, 1999, pp. 942–947.

[15] J.-J.E. Slotine and W. Li, *Applied Nonlinear Control*, Prentice–Hall, Englewood Cliffs, NJ, 1991.

[16] K. Mehlhorn and St. Nahër, *LEDA: A Platform of Combinatorial and Geometric Computing*, Cambridge University Press, Cambridge, UK, 1999.

[17] H. González-Banos, A. Efrat, J.C. Latombe, E. Mao, and T.M. Murali, "Planning robot motion strategies for efficient model construction," in *Robotics Research - The Ninth Int. Symp.*, J. Hollerbach and D. Koditschek, Eds., Salt Lake City, UT, 1999, Springer-Verlag.