

ON-LINE PLANNING FOR AN INTELLIGENT OBSERVER IN A VIRTUAL FACTORY

Tsai-Yen Li

Computer Science Department
National Chengchi University,
Taipei, Taiwan, R.O.C.
E-mail: li@cs.nccu.edu.tw

Tzong-Hann Yu

Computer Science Department
National Chengchi University,
Taipei, Taiwan, R.O.C.
E-mail: g8801@cherry.cs.nccu.edu.tw

Yang-Chuan Shie

Computer Science Department
National Chengchi University,
Taipei, Taiwan, R.O.C.
E-mail: s8536@cherry.cs.nccu.edu.tw

Abstract

In tele-present or tele-operation applications, an intelligent observer is a virtual camera that can perform autonomous motions to assist a user in accomplishing desired observing tasks. In this paper we consider the problem of generating and modifying the motions of an intelligent observer tracking a moving target in an online manner. Given the target's trajectory, we first plan the observer's motion to maintain visibility with the target while optimizing certain time-dependent camera-specific criteria. During the execution of the motion, we allow a user to interactively command the observer to deviate from the planned path while guaranteeing visibility with the target in the remaining motion. The online nature of the problem imposes a constraint on the planning time allowed in each execution cycle. Efficiency of the motion planner becomes the greatest concern. In this work we formulate the problem in the relative configuration-time space (CT-space) of the observer and adopt a best-first algorithm to search for a feasible path. In addition, we propose an incremental backward search algorithm in the CT-space to handle dynamic modification of the planned path. Simulation experiments show that the planner can generate a feasible tracking path in fractions of a second and maintain validity of the planned path for each execution cycle in tens of milliseconds. This planning efficiency is achieved partly due to an efficient visibility detection algorithm that we have developed for planning tracking motions. This implemented intelligent system is part of an interactive 3D auto-navigation system for virtual factories. We believe that these experimental results can inspire a new line of potential applications that could benefit from active, intelligent observations.

Keywords: Motion Planning, Intelligent Observer, Tele-present, On-line Planning, Intelligent Tracking, Visibility Planning, Virtual Reality, and Virtual Factory.

1. Introduction

Tracking a moving object by maintaining a continuously non-obstructive view presents a great challenge for computers as well as for human. Human professionals usually follow common cinematographic idioms to create certain artistic effects. Before a film is taken, the professionals need to plan their camera motions ahead of time in order to maintain good visibility with the subject. If the spatial reasoning model behind the planning process can be successfully created in the computer, we can potentially open up a new dimension of interesting applications. For example, a computer-controlled camera can then present a sequence of non-obstructive image or a video segment to a surgeon during a medical operation. An active vision system with such a tracking capability can be mounted on a mobile robot to provide intelligent surveillance services.[4] In tele-presence or virtual-world applications, by delegating motion tracking to the computer, we can greatly increase the level of control required from a remote user.

The on-line motion-tracking problem considered in this paper was motivated by a virtual presence application for virtual factories.[13] An intelligent tracking module is part of our work on building an auto-navigation system that can generate customized guided tours in an online manner. In this system, a user expresses his/her viewing preferences and points of interests by directly clicking on a 2D-layout map of the virtual world. The system will then generate an

appropriate tour path led by a virtual human guide. The motion of a virtual camera will be planned to track the human motion. In this context, the object to be tracked, called *target*, is the tour guide, whose motion is known, while the tracking agent, called *observer*, is the virtual camera in the virtual world.

Such an auto-navigation system alleviates a user from low-level control of 3D-navigation interface with a 2D mouse. Nevertheless, the planner-generated observing motion may not satisfy the user all the time. It is highly desirable to allow the user to interactively modify the generated camera motion during run-time execution. However, visibility with the target under such user intervention may no longer be guaranteed. One can evoke the planner on-line whenever the user changes the camera configuration in order to generate a new tracking motion. However, the efficiency of motion planners reported in the literature is not good enough for such an on-line demand (within the cycle time of a execution loop or update time of a graphical interface).

In this paper, we present our approaching to making on-line planning for tracking motion feasible. We report improvement on the efficiency of our previously proposed work on planning intelligent tracking motion and extend the planner to handle the case of on-line modification of observer's configuration. Specifically, we have developed an efficient collision check routine to speed up visibility detection and proposed an incremental search algorithm at run-time to distribute planning into each execution cycle. Our experiments show that the efficiency of the planner allow the user to safely interact with the camera in an on-line manner that guarantees visibility with the moving target.

In the next section, we describe previous researches pertaining to this work. In Section 3, we give a formal description of the basic and extended tracking problems we consider. Several user-specified criteria used in searching for a tracking motion are also described in details. In Section 4, we propose our approach that uses an incremental backward search to ensure satisfaction of visibility constraint under user intervention. Implementation details of this planner and experimental results are given in Section 4. We conclude the paper by discussing possible extensions of this planning system in the last section.

2. Related Work

Visibility computation has attracted many attentions in computer graphics and robotics. In computer graphics, visibility information can be used for geometry culling in interactive 3D graphics such as building walkthrough [19] or it can be used to facilitate illumination computation [18]. In computer animation, a camera can be controlled to satisfy constraints such as fixing points in a image space[2] or be

directed in real time according to the film idioms (heuristics) commonly used in Cinematography[6]. The issue of intelligent camera control has also been addressed in [2][13] to enable task-level control of a virtual camera in a virtual world. In robotics, the general motion planning problems are described in [9]. On-line motion planning for two robot arms has also been considered in [12] but not for visibility computation. Visibility planning considers the problem of placing sensors at appropriate locations to ensure the visibility of certain objects. Installing a minimal number of sensors in a static manner is the so-called *art gallery problem* [1][16] while allowing the sensors (usually cameras) to move actively is the so-called *pursuit-evasion problem* [5]. Camera motions are usually controlled via an active vision system, and the motion can also be integrated into control servo loops [7][17].

Similar planning problems about maintaining object visibility were considered in [10] for robotics applications. A general dynamic programming approach was used to generate motions for a mobile robot (as an observer) to keep track of a moving target with a (partially) known trajectory. If the motion of the moving target is predictable, an optimal tracking motion for the observer can be generated in an off-line manner. If the target's motion is only partially predictable, on-line motion strategies are used instead. However, the reported planning time of the off-line planner is too long (about 20 sec. on a modern workstation) to be used in an interactive application even for a simplified view model. In [14], a problem similar to the off-line planning case but with more practical considerations for on-line uses is modeled in the configuration-time space relative to the target. The improved planning time for finding a feasible tracking motion ranges from fractions of a second to a few seconds.

3. The Planning Problem

In this section, we first describe our view of the basic motion-planning problem of maintaining visibility with a moving target, as presented in [14]. Then we extend the formulation to consider the problem of allowing dynamic modification of the observer's motion in an on-line manner.

3.1. The basic tracking problem

We assume that a moving *target* r and an *observer* o (or *camera* interchangeably) exist in a bounded 2D Euclidean workspace W cluttered with obstacle regions B . Although the target and observer all exist in a 3D environment, it is reasonable in our application to confine their motions in a 2D plane to reduce the computational complexity of the problem. The *configurations* of r and o are parameterized by $q^r=(x^r, y^r, \mathbf{q}^r)$ and $q^o=(x^o, y^o, \mathbf{q}^o)$, respectively. A configuration of an object is *collision-free* if and only if regions

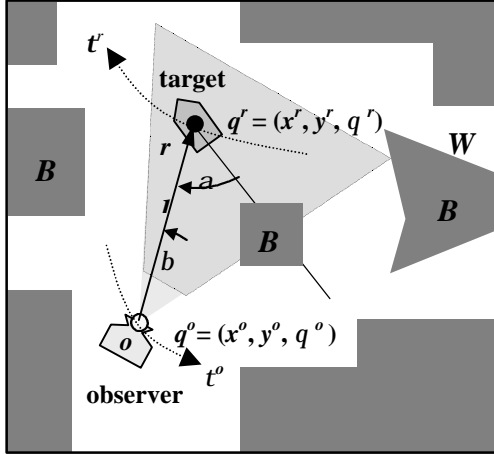


Figure 1. Workspace and observer (camera) parameters in a configuration space relative to the target

occupied by the object do not interfere with B . The *configuration space* (C -space for short) of r and o , denoted by C^r and C^o , respectively, are the spaces of all possible configurations. Let C_{free}^r and C_{free}^o denote the collision-free portions of their C -spaces, respectively. We suppose that the target's trajectory t' is given as a function of time t and that all q^r 's in the trajectory lie in C_{free}^r . With this given trajectory, we try to find a collision-free path for o in the *configuration-time space* of $CT(t, x^o, y^o, q^o) = (x^r(t), y^r(t), q^r(t), x^o, y^o, q^o)$, as formulated in [8][12]. However, not all configurations in CT are legal configurations since a legal configuration must also satisfy the visibility constraints of the observer. That is, in a legal configuration, the observer must be able to remain collision-free from the obstacles, and see portion of the target model with its view model (e.g. a view cone with near and far clipping planes). In addition, the neighboring configurations along a path must also satisfy the maximal velocity constraint imposed on the observer.

Since the observer's goal is to track a moving target, it makes sense to specify its configuration with respect to the target's coordinate system. Therefore, as proposed in [14], we specify the observer's configuration q^o relative to r by the set of parameters $(l, \mathbf{a}, \mathbf{b})$ as shown in Figure 1. This reparameterization equivalently transforms C^o into another space C^o relative to the target's coordinate system. Similarly, $CT(t, x^o, y^o, q^o)$ can be transformed into $CT'(t, l, \mathbf{a}, \mathbf{b})$. These three parameters $(l, \mathbf{a}, \mathbf{b})$ are called *view distance*, *tracking angle*, and *view angle*, respectively.

In order to reduce the size of search space for our planning problem, we further drop the dimension, \mathbf{b} , in CT' by making the following assumption. We assume that the observer can rotate as fast as the target so that we can temporarily lock the view angle \mathbf{b} to a fixed value, say 0 degree,

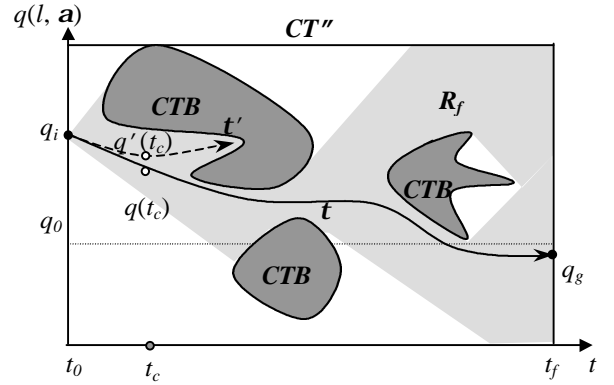


Figure 2. Obstacles (CTB , in dark gray), reachable region (R_f , in light gray) and a sample feasible path t in the conceptual representation of the configuration-time space (For clarity, the two-dimensional q is shown with one dimension only.)

and then remove this parameter from our search space CT' . The resulting configuration-time space $CT''(t, l, \mathbf{a})$ is what we will consider in the remaining of this paper. After the observer path in CT' is found, the parameter \mathbf{b} will then be accounted for and added into $q^o(l, \mathbf{a}, \mathbf{b})$ in a post-processing step. In order to simplify notation, we will drop the superscript o in symbols for observer configurations whenever it does not cause confusions.

3.2. Planning with viewing criteria

In [14], we described a Best-First search algorithm to find a collision-free path, t , in CT'' , connecting the given initial configuration q_i to any legal goal configuration $q_g(t_f, *, *)$ in the final time slice t_f , as shown in Figure 2. Any points inside the CTB region are illegal due to geometry or visibility constraints. The search starts from q_i and continuously explores neighbors of the current configuration that is evaluated to be the "best" according to some cost function. The neighboring configurations also must not violate the maximal velocity constraint v_{max} imposed on the observer. The cost function used in the Best-First search is defined as a linear combination of the weights on several viewing criteria as shown in the following equation.

$$f(t, l, \mathbf{a}, dir) = w_1 f_1(t) + w_2 f_2(l) + w_3 f_3(\mathbf{a}) + w_4 f_4(l, \mathbf{a}, dir) \quad (1)$$

$$f_1(t) = t_e - t$$

$$f_2(l) = |l - l_0|$$

$$f_3(\mathbf{a}) = |\mathbf{a} - \mathbf{a}_0|$$

$$f_4(l, \mathbf{a}, dir) = dist(p(l, \mathbf{a}, 0), p(l, \mathbf{a}, dir)),$$

where

- w_i : weights for individual cost functions,
- t : current time,
- l : current distance between the viewpoint and the target,
- \mathbf{a} : current tracking angle,
- dir : an integer indicating the direction where the current

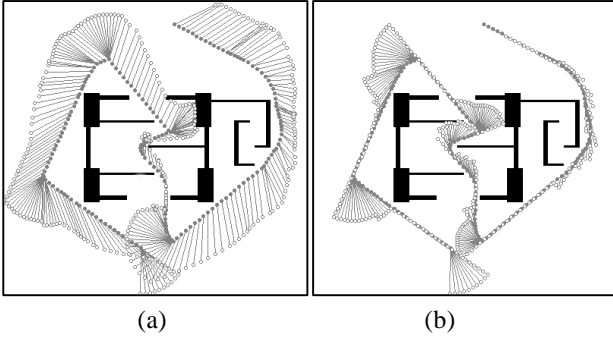


Figure 3. An example of viewpoint tracking motion with emphases on different criteria: preferring keeping (a) a good view distance, l , (b) a good tracking angle, α . (dot: target, circle: observer)

configuration was created (0 means unchanged),

f_1 : cost function for the distance between the current and the ending time slices (t_e is the ending time),

f_2 : normalized cost function for the view distance (l_0 is a preferred neutral view distance),

f_3 : normalized cost function for the tracking direction (α_0 is a preferred neutral tracking angle),

f_4 : normalized cost function for the Euclidean distance moved from the parent configuration,

p : returns the previous position of the observer for the given approaching direction,

$dist$: returns the distance between two positions.

In this cost function, l_0 and α_0 are the preferred view distance and tracking angle specified by the user. In addition, the coefficients w_i 's ($i = 1$ to 4) represent the user's preference on the weights of the viewing criteria defined by the four individual cost functions. Since these criteria might conflict with each other, deciding the importance of each individual function is a subjective matter. Examples of the tracking path generated by the planner with different weights are shown in Figure 3. The set $P=(l_0, \alpha_0, w_1, w_2, w_3, w_4)$, defined as the *viewing preference set*, determines the search result and is specified by a user for a given scenario. A user may specify different P 's for different time windows when tracking a moving target performing different tasks. For example, in tele-present applications, a user may prefer to keep a good view distance from the target when the target transits from one room to another but once the target enters a room, the preference may be changed to avoid unnecessary moves whenever possible.

3.3. On-line modification of observer's motion

The observer's path \mathbf{t} is sent for execution after it has been generated by the planner. Visibility with the target is guaranteed if the path is followed exactly as planned. However, this first found path by no means is the optimal path according to the given cost function. Therefore, it is highly

highly desirable to allow the user to voluntarily modify the planned motion with some provided user interface. For example, at a given time t_c , the observer is supposed to be at some configuration $q(t_c)$ if it follows \mathbf{t} . However, the user may command the observer to move to a different configuration $q'(t_c)$. The problem becomes how to determine if the observer, when moved to $q'(t_c)$, can still maintain visibility with the target for the remaining execution time. One can imagine the situation where the modified path gradually deviates from the original one up to a point that the target will inevitably disappear from the observer's view. For example, the modified path \mathbf{t}' in Figure 2 is inside the reachable region R_f from q_i under the velocity constraint, but there exist no feasible continuing paths that can bring the observer to the final time t_f without bumping into the obstacle regions.

4. The Proposed Approach

In this section we present our approach to incorporating user interactions into the planning of the observer's motion. In order to generate tracking motions that are close to what a user expects, we allow the user to interactively specify different viewing preference sets (P 's) at different periods of time in the target's trajectory. After the observer's motion has been generated by the planner, we use an on-line planning algorithm in the observer's execution loop to examine any dynamic modifications of the observer's motion before the motion is sent for execution. We will also describe an efficient visibility check routine that enables the on-line planning.

4.1. Maintaining visibility at run-time

We would like the user to modify the observer's path \mathbf{t} in run-time while it is executed. Because the user input is not predictable, we have to ensure that the modified configuration remains valid at the sampling rate of the observer's execution loop. At any time step t_c , we get a modification command u from the user. This command is supposed to bring the observer to a new configuration $q'(t_n)$ for the next time step t_n . However, the guaranteed visibility for the remaining execution time is lost. Since the time dimension is not reversible, one cannot back up to the original point in CT'' either.

One way to ensure this validity is by evoking planning in each execution loop to find a new feasible path connecting $q'(t_n)$ back to the original path \mathbf{t} or directly to the goals. Currently, the planning times of most known planners range from fractions of a second to a few seconds in average cases. In the worst case, where no paths exist and the whole search space needs to be traversed, the planning time will be too long for on-line uses. Actually, the worst case may happen

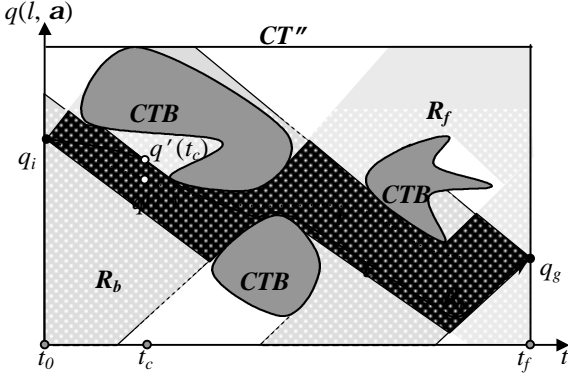


Figure 4. Forward reachable region R_f , backward projection region R_b , and their cross section R_c in CT'' . The planned path \mathbf{t}^o can be transformed to \mathbf{t}^o' only inside R_c .

very often since the purpose of on-line planning in our case is to monitor and eliminate such potential violations.

We approach this problem by incrementally searching the CT'' space backward in time along \mathbf{t} in order to find a guaranteed path connecting the modified configuration back to \mathbf{t} . We define $\mathbf{t}^*(t_n)$ as the subpath of \mathbf{t} for $t \geq t_n$. Instead of trying to connect $q'(t_n)$ forward to some configuration $q(t_r)$ in $\mathbf{t}^*(t_n)$, we connect $q(t_r)$ backward to $q'(t_n)$. The configuration $q(t_r)$ is a connecting point in $\mathbf{t}^*(t_n)$ that brings the observer back to \mathbf{t} . The new subpath connecting $q'(t_n)$ and $q(t_r)$ is denoted by $\mathbf{t}_r(t_n)$. As shown in Figure 4, R_f denotes the region reachable from q_i in CT'' . R_b denotes the region reachable from some point in \mathbf{t} . Equivalently, any points in R_b can be connected to \mathbf{t} under the v_{max} constraint. The intersection of R_f and R_b , denoted by R_c , is the safe region where the modified observer path must lie.

Although conceptually R_c is the region in which one needs to confine the observer's motion, it may not be a good idea to explicitly compute R_c . If the 3D CT'' space is represented in a discrete form of some resolution as in our planner, the number of points to be marked in this region could be very large, especially when the time dimension is long. However, only the portion of R_c that belongs to the vicinity of the modified path \mathbf{t}' might need to be considered. If the planned path is executed without any modifications, there are even no needs to compute this region. Therefore, we take an on-demand approach that incrementally builds portion of R_c as needed at run-time. Any points in R_c are marked in CT'' after they have been visited.

The algorithm for safely executing the observer's tracking motion with dynamic modification from a user is outlined in Figure 5. Starting from the initial time t_0 , we have to ensure that at any time t_c , there exists a safe returning sub-path $\mathbf{t}_r(t_c)$ connecting $q'(t_c)$ back to some configura-

1 procedure ExecuteSafePath

```

2  $t_c \leftarrow 1$  // ticker
3  $\mathbf{t}' \leftarrow \mathbf{t}^*(t_c)$  //  $\mathbf{t}'$  is the current valid path
4  $q' \leftarrow \text{First}(\mathbf{t}')$ ; MoveTo( $q'$ ); // update observer's config
5 while( $t_c \leq t_f$ ) { // loop until the end of execution
6    $u \leftarrow \text{UserCommand}()$  // get user input
7    $q'' \leftarrow q' + u$  // compute new config
8    $t_s \leftarrow \text{StartTime}(q'', \mathbf{t}^*(t_c))$  // when search should start
9    $\mathbf{t}_r \leftarrow \text{FindPath}(q(t_s), q'')$  // search for a legal path
10  if ( $\mathbf{t}_r$  is not NULL) then
11     $\mathbf{t}' \leftarrow \mathbf{t}_r + \mathbf{t}^*(t_s)$  // update the modified path
12  else
13     $\mathbf{t}' \leftarrow \mathbf{t}^*(t_c+1)$  // resume the old safe path
14     $q' \leftarrow \text{First}(\mathbf{t}')$ ; MoveTo( $q'$ )
15     $t_c \leftarrow t_c + 1$ 
16 end while
17 end procedure
```

Figure 5. The on-line planning algorithm during motion execution for maintaining visibility under user intervention

tion $q(t_r)$ in $\mathbf{t}^*(t_c)$. In each execution step, we search for such a returning sub-path for the attempted new configuration $q'(t_n)$ ($t_n = t_c + 1$) with the given user input u . If the search succeeds, the $q'(t_n)$ can be safely executed and the returned subpath $\mathbf{t}_r(t_n)$ is saved for future references. Otherwise, the command u is cancelled and the observer will resume the subpath $\mathbf{t}_r(t_c)$ saved for the current time step.

In the ExecuteSafePath procedure, two subroutines need further explanations. The StartTime() subroutine in line 8 determines the time step where the backward search should start. It is the first time index, t_s ($t_s > t_c$), such that $q'(t_n)$ and $q(t_s)$ can possibly be connected without violating v_{max} . The planning process, performed in the FindPath() subroutine in line 9, starts from the configuration $q(t_s)$ and search backward in time in a breadth-first manner until the goal ($q'(t_n)$ in this case), is reached (success) or all configurations down to the time t_n have been exhausted (failure). Due to the v_{max} constraint, the backward reachable region from $q(t_s)$ is a cone-shape region. Since some of the configurations in this cone might have been visited in previous steps, the number of configurations that need to be visited is rather small. In other words, this region is visited incrementally and computation cost is distributed to each execution step. Therefore, such an incremental search strategy is appropriate for planning tracking motion with on-line user modification.

4.2. Fast visibility checks

On-line planning is integrated into the execution loop of the observer's motion with the help of efficient visibility

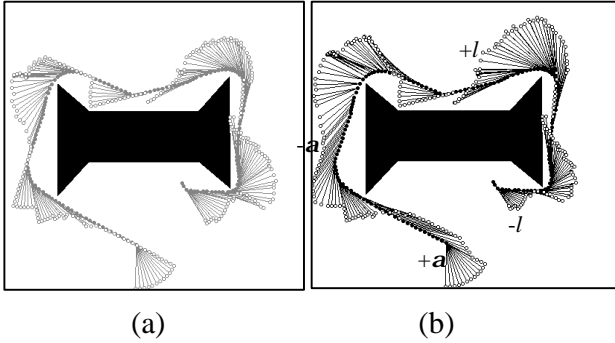


Figure 6. An example of on-line modification of the observer's tracking motion. (a) the planned path (b) the on-line modified path

checks. During the search, visibility with the target determine the legality of an observer's configuration in the CT'' space. Like in most motion planners, collision detection is the most time-consuming routine in the planning computation. A primitive routine for our visibility planning is detecting the potential collisions of a line segment with the obstacles in the workspace. For example, with the simplest view model, maintaining visibility could just mean keeping a line segment of variable lengths in the freespace. When the complexity of the environment increases, the collision detection routine will also become more expensive.

To speed up visibility checks, we compute a 3D distance map, denoted by DM , in a preprocessing step for all possible configurations of an observer. The distance information contained in DM is used to quickly determine if a line segment collides with obstacles in run-time. DM is a 3D uniform grid, in which the coordinate of each cell corresponds to an observer configuration (x^o, y^o, q^o) in the Cartesian space. The value contained in each cell represents the distance to the first obstacle boundary when we shoot a ray from the corresponding configuration. When doing collision checks for a line segment, we first determine its configuration from one of its end point and look up the cell value for this configuration in DM . If the value is less than the length of the line segment, a collision is implied; otherwise, the line segment is collision-free.

Although DM is computed in a preprocessing step for a given environment, the total computation should not take too long. In [15], a well-known algorithm was proposed to compute C-space obstacles in time linear to the complexity of the obstacles. We adopt this algorithm to efficiently compute DM . In our case, the robot in this C-obstacle computation is a line segment. With this algorithm, one can compute the 3D C-obstacles for such a robot in fractions of a second on a regular PC. A useful observation is that when the length of the line segment changes, the vertices of the C-obstacle polygons are only changed by a constant offset. We compute DM by first initializing it with some large

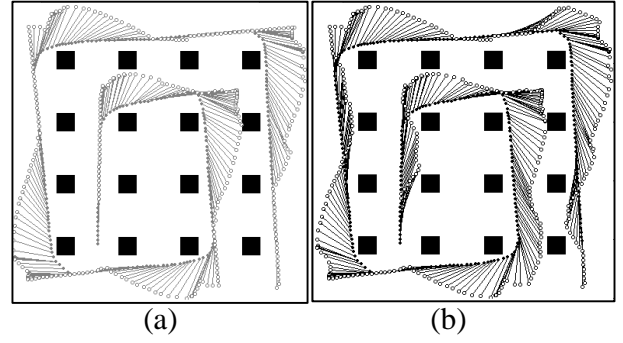


Figure 7. Another example of on-line modification of the observer's tracking motion. (a) the planned path (b) the on-line modified path

value. Then we discretize the length of the line segment in its legal range and compute the C-obstacle polygons for each discrete length in a descending order. The interior of the C-obstacles in DM for a given length is filled with the length value. Since the C-obstacle polygons are shrinking gradually as the length decreases, different distance values show up correctly after the iteration on length is finished.

5. Experiments

5.1. Implementation

We have implemented the planning algorithm described in the previous sections in the Java language. At the current stage, the planner contains two parts: a path planning module that computes sequences of non-holonomic motions for the target, and a motion tracking module that computes tracking motions for the observer. The system runs on regular desktop PC's or UNIX workstations and provides a graphical user interface for a user to specify the target's goals and personal tracking preferences such as the weights for various cost functions. During the execution of the motions, a user can use a mouse to input modification to the observer motion. The vector dragged with a mouse is decomposed into the horizontal and vertical components, representing the desired offsets for the l and a parameters of the observer's configuration, respectively. In each execution loop, this desired offsets are converted into motion commands applied to the current configuration of the observer.

We represent CT'' by a three-dimensional uniform grid (a 3D bitmap) as proposed in [12][14]. This bitmap is used to mark the visited configurations in the Best-First search for planning the observer's tracking motion. The time resolution of the CT'' space is determined according to the maximal velocity constraint of the observer. During the search, moving a configuration to its neighboring configuration in the next time slice will not violate the velocity constraint. The same bitmap, after being reinitialized, is also used to mark visited configurations in the on-line planning

phase.

5.2. Experimental results

The planning time for generating a tracking motion mainly depends on the volume of the regions visited during the Best-First search process. In the worst case where no feasible paths exist, the total number of cells in the free configuration-time space can still be visited in a few seconds. In most cases, the running times (measured on a Celeron 400 MHz PC) are less than a second with the improved visibility check routine. For the examples shown in Figure 2, the workspace resolution is a uniform grid of 128x128 and each rotational increment is 3 degrees. The minimal, neutral, and maximal view distances (l_{min} , l_0 , l_{max}) are 12, 20, and 60 units, respectively. The tracking angle (α) spans a range of 220 degrees (± 110 degrees) with the neutral position being right behind the target. The number of time steps in CT'' is 302, and the planning times for both cases are all 0.16 seconds (compared to 1.43 seconds, when the improved routine for visibility checks is not used).

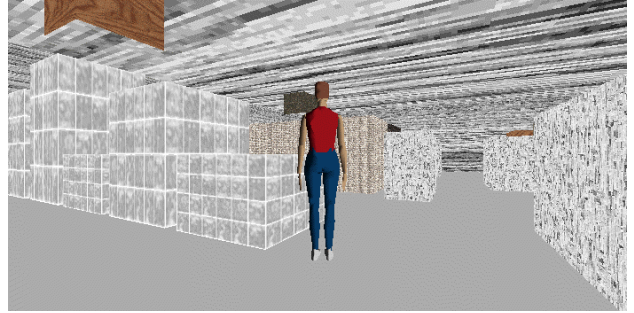
Figures 6 and Figure 7 show two examples of on-line modification to the observer's motions. Part (a) in each figure shows the planned path while part (b) shows the trace of the actual executed path under the influence of user interaction. In Figure 6, note that the user modified the planned path during run-time execution by consecutively increasing α , decreasing α , increasing l , and then decreasing l . For the example in Figure 7, out of the 346 execution steps, only 75 modifications are feasible. The remaining illegal ones include the situations where the desired observer configuration results in collision or the on-line search fails. The average number of configurations visited in each on-line search is only 23, and the maximal planning time in an execution step is around 60ms (about 230 configurations were visited), which is short enough for on-line uses.

6. Conclusion and Extensions

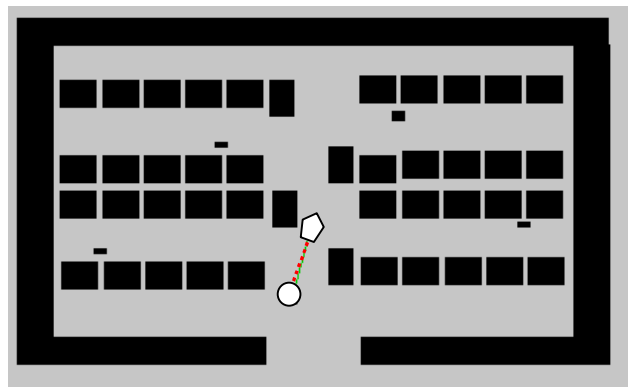
6.1. Conclusion

Visibility planning has great potential applications in both robotics and computer graphics. Due to the limited time for planning, allowing a user to modify the planned motion for an observer while maintaining visibility with the target in an on-line manner is a new challenging problem. In this paper, we have presented a novel approach to this problem, in which we use an incremental backward search algorithm to distribute planning computation along the execution of the observer's motion. An efficient visibility check routine has also been developed to speed up on-line planning. Our experiment shows that the planning time for each execution loop is rather satisfactory for on-line purposes.

6.2. Extensions



(a)



(b)

Figure 8. Graphical user interface of the virtual factory. (a) in a VRML browser, and (b) in a 2D-layout map (pentagon: virtual guide, circle: virtual camera).

In this paper we mainly consider the case where the motion of the target is known or predictable while allowing the motion of the observer to be modified on-line. However, in many applications, the target's motion is not necessarily known in advance. We can predict the target's motion and plan the observer's motion accordingly, but the generated path may become invalid as soon as the target does not move as predicted. Although we can re-plan the path whenever it becomes invalid, the efficiency of our planner is still not good enough to be evoked in every execution cycle. (The planning time in the worst case could be a few seconds when the size of the target's path is long.) We are developing an on-line planner with a search window (the number of future time steps) that is adjustable according to the available CPU time in each execution cycle. However, in this case, visibility can no longer be guaranteed since the planner only considers a few time steps ahead. Completeness and planning time become a tradeoff in this situation. Our preliminary experimental results show that as long as the target does not move too fast and the motion can be reasonably predicted, a small size (e.g. 5 time steps) of planning window will suffice to maintain visibility with the target.

We are in the process of integrating the on-line planner into a virtual presence system that motivated this work. Graphics rendering and user input will be connected to a 3D user interface as shown in Figure 8. In the current user interface, modifications made to the planned path are proportional to the magnitude of user input along any dimensions. However, it might be more desirable to incorporate some spring model into the interface such that the more the offset is in a dimension, the harder one can adjust the configuration along that direction. In addition, extending the planner to handle 3D obstacles is also a challenging work that will be considered in the future.

Acknowledgements

This work was partially supported by grants from National Science Council under NSC 89-2218-E-004-002 and NSC 89-2815-C-004-002-E.

References

- [1] J. Briggs and B. R. Donald. "Robust Geometric Algorithms for Sensor Planning," J.-P. Laumond and M. Overmars, editors, *Proceedings of 2nd Workshop on Algorithmic Foundations of Robotics*. A.K. Peters, Wellesley, pp. 197-212, MA, 1996.
- [2] S. M. Drucker and D. Zeltzer, "Intelligent Camera Control in a Virtual Environment," *Graphics Interface '94*, pp. 190-199, 1994.
- [3] M. Gleicher and A. Witkin, "Through-the-Lens Camera Control," In E.E. Catmull, editor, *Computer Graphics (SIGGRAPH'92 Proceedings)*, volume 26, pp. 331-340, 1992.
- [4] H. H. Gonzalez-Banos, L. Guibas, J.-C. Latombe, S.M. LaValle, D. Lin, R. Motwani, and C. Tomasi, "Motion Planning with Visibility Constraints: Building Autonomous Observers," *Proceedings of the Eighth International Symposium of Robotics Research*, Hayama, Japan, Oct. 1997.
- [5] L. J. Guibas, J.-C. Latombe, S. M. LeValle, D. Lin, and R. Motwani, "Visibility-Based Pursuit-Evasion in a Polygonal Environment," *Proceedings of the 5th Workshop on Algorithms and Data Structures*, Springer Verlag, pp. 17-30, 1997.
- [6] L-W He, M. F. Cohen and D. H. Salesin, "The Virtual Cinematographer: a Paradigm for Automatic Real-Time Camera Control and Directing," *Proceedings of ACM SIGGRAPH'96*, pp. 217-224, 1996.
- [7] S. Hutchinson, G.D. Hager, and P.I. Corke, "A Tutorial on Visual Servo Control," *IEEE Transaction on Robotics and Automation*, 12(5):651-670, Oct. 1996.
- [8] K. Kant and S. W. Zucker, "Toward Efficient Trajectory Planning: The Path-Velocity Decomposition," *International Journal of Robotics Research*, 5(3):72-89, 1986.
- [9] J.-C. Latombe, "*Robot Motion Planning*," Kluwer Academic Publisher, Boston, MA, 1991.
- [10] S. M. LaValle, H. H. Gonzalez-Banos, C. Becker, J.-C. Latombe, "Motion Strategies for Maintaining Visibility of a Moving Target," *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, 1997.
- [11] S. Lavallee, J. Troccaz, L. Gaborit, A.L. Benabid P. Cinquin, and D. Hoffman, "Image-Guided Operating Robot: A Clinical Application in Stereotactic Neurosurgery," in R.H. Taylor, S. Lavallee, G.C. Gurdea, and R. Mosges, editors, *Computer Integrated Surgery*, pp. 343-351, MIT Press, Cambridge, MA, 1996.
- [12] T.-Y. Li and J.-C. Latombe, "Online Manipulation Planning for Two Robot Arms in a Dynamic Environment," *International Journal of Robotics Research*, 16(2):144-167, Apr. 1997.
- [13] T.Y. Li, J.M. Lien, S.Y. Chiu, and T.H. Yu, "Automatically Generating Virtual Guided Tours," in *Proceedings of the Computer Animation '99 Conference*, Geneva, Switzerland, pp99-106, May 1999.
- [14] T.-Y. Li and T.-H. Yu, "Planning Object Tracking Motions," in *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, May 1999.
- [15] T. Lozano-Perez, "Spatial Planning: A Configuration Space Approach," *IEEE Transaction on Computers*, 32(3):108-120, 1983.
- [16] J. O'Rourke, "*Art Gallery Theorems and Algorithms*," Oxford University Press, NY, 1987.
- [17] 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 Transaction on Robotics and Automation*, 9(1):14-35, February 1993.
- [18] Teller, and P. Hanrahan, "Global Visibility Algorithms for Illumination Computations," *Proceedings of SIGGRAPH'97*, pp. 239-246, 1997.
- [19] Teller, and C. Sequin, "Visibility Preprocessing For Interactive Walkthroughs," *ACM Computer Graphics (Proceedings of SIGGRAPH'91)*, 25(4):61-69, 1991.