

# Visibility Planning for Active Video Collection

Ser-Nam Lim Larry S. Davis  
CS Dept., University of Maryland, College Park  
{sernam, lsd}@cs.umd.edu

## Abstract

Many surveillance tasks can only be performed effectively if sufficiently long videos containing one or more unobstructed views of the targeted objects are available. Temporal intervals suitable for collecting these videos, refer to as visibility intervals, are determined as the set complements of temporal intervals of occlusions. These occlusion intervals span intersections between temporal functions representing straight-line object trajectories in polar coordinates. To efficiently compute these intersection points, we propose an optimal segment intersection algorithm, as opposed to an obvious brute force algorithm with a computational complexity of  $O(N^2)$ . While seemingly restrictive, the requirement for straight-line object trajectories can be overcome with an object representation that combines the estimates of its physical size and time-varying positional uncertainty, and a simple algorithm that splits nonlinear trajectory of the object's contour into straight-lines. Although cameras can then be scheduled and actively controlled based on the constructed visibility intervals, finding an optimal schedule is a typical NP-hard problem. For this purpose, we propose a greedy algorithm and provide an analysis of its approximation factor. Simulation results show the optimal segment intersection algorithm outperforms the brute force algorithm, while experimental results from a real-time system demonstrate the applicabilities of our algorithms.

## 1. Introduction

We describe asymptotically efficient algorithms for controlling a collection of calibrated cameras to acquire video sequences of unobstructed objects (people, vehicles), that are suitable for performing specific surveillance task, as they move into and through a surveillance site. Examples of typical surveillance tasks are:

1. Collect  $k$  seconds of unobstructed video from as close to a side angle as possible for any person who enters the surveillance site. This task might be defined to support gait recognition, or the acquisition of an appearance model that could be used to subsequently identify the person when seen by a different camera.
2. Collect unobstructed video of any person while that

person is within  $k$  meters of region A. This might be used to determine if a person deposits an object into or takes an object out of region A.

Each camera can be independently panned, tilted and zoomed, and has a field of regard which is the subset of the surveillance site that it can image by controlling its settings. A field of view of a camera is the image obtained at a specific Pan/Tilt/Zoom (PTZ) setting and is generally much smaller than its field of regard. The cameras are actively controlled, based on an efficient scheduling algorithm, to capture video that satisfy as many of the defined surveillance tasks as possible, possibly subject to additional constraints on priority of the tasks.

Our approach involves tracking the objects in the surveillance site using one or more wide field of view cameras, for a short period of time, and then predicting their motions over a "small" future time interval. During this interval, we must predict time-varying visibility of the objects, solve the task scheduling problem, re-position cameras and acquire videos to support the scheduled tasks.

The information that would be needed by any such scheduling algorithm would include the future time intervals, which we call visibility intervals, during which each object being tracked through the surveillance site would be (probabilistically) visible from each of the surveillance cameras - i.e., would be in its field of regard and not occluded by another moving object or a fixed object in the site. A visibility interval is a 5-tuple:

$$(c, o, p, \alpha, [t^-, t^+]), \quad (1)$$

where  $c$  represents a camera,  $o$  is the index of the object to which the task is to be applied,  $p$  is the required duration of the task, including latencies in re-positioning cameras,  $\alpha$  is a predicate relating the direction of movement of the object and the optical axis of the camera used to accomplish the task (for example to specify a view that satisfies the requirements for a side view or a front view), and  $[t^-, t^+]$  is a time interval during which the object is visible in the field of regard of camera  $c$ , while satisfying  $\alpha$ . Additionally, the prediction of these visibility intervals must take into account the time-varying positional uncertainty of the moving people and vehicles, which generally increases as time progresses and which can be accounted for with an object representation that combines the estimates of its physical size and such uncertainties.

There is an obvious brute force solution to the problem of constructing such visibility intervals, which would have computational complexity  $O(N^2)$ , where  $N$  is the sum of the number of views of moving and fixed objects in the site. We refer to this problem as the visibility planning problem. For a small number of moving objects, this brute force algorithm suffices, but as the number of views of moving objects increases - due to either a more cluttered site or the use of a larger number of cameras with overlapping fields of regard, it could lead to a system bottleneck. We describe, then, a more efficient and scalable  $O(N \log N + I)$  algorithm based on an optimal segment intersection approach to solve the visibility planning problem,  $I$  being the number of reported occlusions. While camera scheduling could then be conducted solely on the basis of the constructed visibility intervals, finding an optimal solution is an NP-hard problem. Consequently, computationally feasible solutions can only be obtained with approximation algorithms. A greedy algorithm is proposed for this purpose, and its performance is analyzed.

The field of visibility-based sensor planning was surveyed in [1]; that survey covered sensing strategies for object feature detection, model-based object recognition and localization and scene reconstruction. Most prior research has considered static visibility planning problems - placements of cameras in static environments to maximize visibility. [2] introduced a locus-based approach to determine the placement of viewpoints, subjecting to resolution, focus, field of view and visibility constraint. [3] introduced local separating planes to define visibility volumes, in which occlusion-free viewpoints can be placed. The same idea was also described in [4]. Then, to satisfy the field of view constraint, they introduced the field of view cone, which is similar to the locus-based approach given in [2]. Planning in the presence of moving objects was considered in [5, 6, 7], a series of papers describing a dynamic sensor planning system. These researchers were concerned with objects moving in the scene, thus generating what they called a swept volume in temporal space [5]. Then, using a temporal interval search, they divide temporal intervals into halves while searching for a viewpoint that is not occluded in time by these sweep volumes. [8] uses observations of the motion of objects in the surveillance site as probabilistic priors for placing cameras that ensure (probabilistically) that cameras are positioned to have unobstructed views of moving objects.

## 2. Object Model

Determining visibility intervals for any given (object, camera) pair involves computing time intervals during which that object is in the same line of sight as some other object, but is further from the camera, causing it to be occluded. The complements of these intervals, which we refer to as occlusion intervals, are the visibility intervals. In addition to depending on the trajectory of the object acquired through visual tracking, such occlusion intervals would also depend on the object's shape and size, for which we use a spherical representation. The size of the sphere combines

our estimates of physical object size along with the time-varying uncertainty in the location of the object, estimated from tracking.

### 2.1. Motion Model

In the world coordinate system (with the axes' notations as  $X$ ,  $Y$  and  $Z$  respectively), we orthographically project the center of a given camera and the sphere representing each object at a given time, as points and circles respectively, onto the  $XY$ ,  $YZ$  and  $XZ$  planes. On each plane, a projected circle has two tangent points that define its extent w.r.t the projected camera center. The motion model is then defined as the time-varying angular extents of the pairs of tangent points belonging to such a triplet of circles representing the object. These projections serve as a simple representation of an otherwise complex 4D ( $XYZ$  and time) motion model. Fig. 1 shows such a projected circle on the  $XY$  plane, with radius  $r$ , of an object  $o$  w.r.t to the camera center  $c$ . Here,  $\theta$  is the angular displacement of the circle center from  $c$ , and the angular displacement of the upper and lower tangents can be expressed as  $\theta + \alpha$  and  $\theta - \alpha$  respectively, where  $\alpha = \arcsin \frac{r}{d}$ . Accordingly, the motion model of object  $o$ ,  $f_{c,o}(t)$ , parameterized by time  $t$  becomes:

$$f_{c,o}(t) = \bigcup_{\Pi=XY,XZ,YZ} \left\{ \theta(t, \Pi) \pm \arcsin \frac{r(t, \Pi)}{d(t, \Pi)} \right\}, \quad (2)$$

where  $d(t, \Pi)$  and  $\theta(t, \Pi)$  are the distance and the angular displacement, respectively, of the circle center from  $c$ , and  $r(t, \Pi)$  is the radius of the circle on the plane  $\Pi$ .

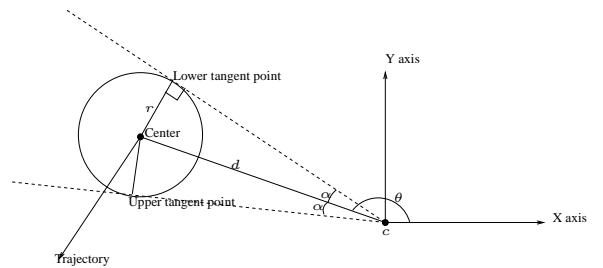


Figure 1. An example of projecting a sphere as circle onto the  $XY$  plane. Here,  $\alpha = \arcsin \frac{r}{d}$ .

### 2.2. Predicting Future Object Location

The constructed motion models are used to predict the future positions of the tangent points and circle centers representing an object on the  $XY$ ,  $XZ$  and  $YZ$  planes. In doing so, the positional uncertainty of the object has to be accounted for, which we address as follow. Let the center of a projected circle on one of the planes be  $c_{obj}$ . The system maintains a set,  $S_{hist}$ , that contains previous positions of  $c_{obj}$  up to the current time. Different subsets of  $S_{hist}$  are used for estimating the direction and speed of  $c_{obj}$ . To

determine the direction, a regression algorithm is used to fit a straight line to the past locations of  $c_{obj}$  in each subset, while an estimate of the speed is simply derived as the mean of the subset. Then, we can form a new set,  $S_{pred}$ , consisting of the predicted velocities of  $c_{obj}$  as:

$$S_{pred} = \{x_0, x_1, \dots, x_n\}, \quad (3)$$

where each  $x_{i=0, \dots, n}$  is a 2-vector of speed and direction. Each  $x_i$  is assigned a weight  $w_i$ , with more recent observations being assigned larger weights, and with all weights normalized so that  $\sum_{i=0}^n w_i = 1$ . Choosing a Gaussian kernel and assuming that both speed and direction are independent of each other, the probability of observing a velocity  $v$  can then be non-parametrically estimated as:

$$Pr(v) = \sum_{i=0}^n w_i \prod_{j=1}^2 \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{1}{2} \frac{(v_j - x_{i,j})^2}{\sigma_j^2}}, \quad (4)$$

where  $j$  represents the speed and direction component, and  $\sigma_j^2$  is the corresponding bandwidth. The confidence interval,  $[-a, a]$ , that gives a desired level of confidence,  $P$ , can be found by solving for:

$$P = \int_{-a}^a Pr(v) dv. \quad (5)$$

$[-a, a]$  provides multiple hypotheses of the predicted positions of  $c_{obj}$  starting from the current position, with each of these positions representing the center of a circle with radius equal to the object size. A Minimum Enclosing Circle (MEC) can be determined to enclose these smaller circles, using a linear-time algorithm given in [9] pp. 86-90. Such a MEC, when used to represent the object, appropriately models its positional uncertainty (at a desired confidence level) and physical extent. Moreover, it typically increases in size as the paths taken by different hypotheses increasingly deviate from each other. We provide an illustration in Fig. 2.

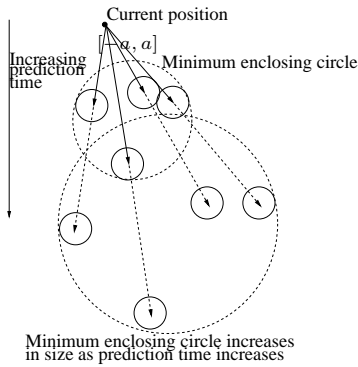


Figure 2. Multiple hypotheses are formed in  $[-a, a]$ , which are shown as the smaller circles at two different time instances in the prediction period. For the later time instance, the MEC, representing the object, grows in size as shown.

### 3. Visibility Analysis

Each hypothesis used in predicting the future location of an object is associated with a constant speed and a fixed direction. Together they form the predicted motion model, which can be visualized as a progression of MECs in time. Such a series of MECs displays the following properties that are useful for expressing the predicted motion model in an analytic representation:

**Theorem 1** *The set of hypotheses that touch the circumference of each MEC of the predicted motion model remains the same in time.*

**Proof** We consider the MEC of the hypotheses' centers; the radius of such a MEC can then be simply increased by the physical size of the object. A valid MEC possesses two attributes: (a) the centers of at least two hypotheses lie on the circumference, and (b) it does not contain an arc greater than half the circumference, on which no center lies. Consider, then, an MEC,  $C_{init}$ , with radius  $r$  at a given time step. Referring to Fig. 3 and property (a) and (b), we consider two hypotheses with centers lying on the circumference of  $C_{init}$ , and another hypothesis,  $h_{in}$ , of which the center lies inside  $C_{init}$ , and the straight line distance to  $C_{init}$  is  $r'$ . By similar triangles, it is easy to see that at the next time step, there exists a circle,  $C_{new}$  of radius  $2r$ , so that the pair of hypotheses that was originally on the circumference of  $C_{init}$  remains on that of  $C_{new}$ , and the distance of the new position of  $h_{in}$  from the center of  $C_{new}$  is  $2r'$ . Since  $r' < r$ , the new position of  $h_{in}$  would still lie within  $C_{new}$ . Thus, both property (a) and (b) are preserved, and  $C_{new}$  can serve as the new MEC.  $\square$

**Corollary 1** *The center of the MEC moves at a constant speed and in a fixed direction.*

**Proof** Referring to Fig. 3, let the distance from the center of  $C_{init}$  to the current position be  $d$ . Then by similar triangles, the center of  $C_{new}$  is at a distance  $2d$  from the current position in the second time step. Moreover, the center of  $C_{init}$  and  $C_{new}$ , and the current position lie on the same straight line.  $\square$

**Corollary 2** *The radius of the MEC grows by a constant factor. Proof omitted since it follows easily from Theorem 1.*

The above theorem and corollaries show that both the time-varying locations of the centers and radii of the MECs can be expressed in simple analytic form, thus allowing the predicted motion model to be represented as given in Eqn. 2. Using the predicted motion model, we convert the trajectories of the tangent points of the MEC w.r.t a given camera center into analytic straight-line representations, using Algorithm 1.

The computational advantage of using straight line is the ease of conversion to a much simpler analytic representation than Eqn. 2, which greatly simplifies the construction of occlusion intervals in Sec. 3.1. The conversion is performed in a camera-centered polar coordinate system, where the

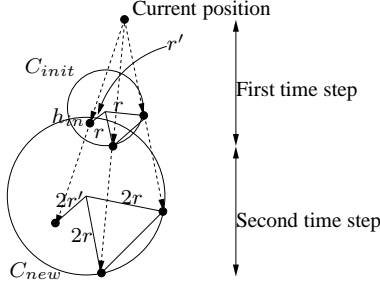


Figure 3. The new positions of the two centers lying on the circumference of  $C_{init}$  remain on that of  $C_{new}$ , while that of  $h_{in}$  remains inside  $C_{new}$ . By similar triangles, it is easy to see that the center of the MEC moves at a constant speed and in a fixed direction, while the radius grows at a constant factor. Note that each hypothesis is moving at a constant speed and thus its distance from the current position doubles in the second time step.

---

### Algorithm 1 SplitTangent( $t_0, t_1, \Pi$ )

---

- 1:  $\{\Pi$  is the  $XY, XZ$  or  $YZ$  plane, on which the trajectory of the tangent point is split into straight line(s)}.
  - 2: Let  $p_{t_0}$  be the position of the tangent point on  $\Pi$  at  $t_0$ , computed from the predicted motion model.
  - 3: Let  $p_{t_1}$  be the position of the tangent point on  $\Pi$  at  $t_1$ , computed from the predicted motion model.
  - 4: Interpolate for the midpoint,  $m$ , of  $p_{t_0}$  and  $p_{t_1}$  on  $\Pi$ , i.e.,  $m = \frac{p_{t_1} + p_{t_0}}{2}$ .
  - 5: Compute from the predicted motion model, the actual midpoint,  $m'$ , on  $\Pi$  of the tangent point at time  $\frac{t_0 + t_1}{2}$ .
  - 6: **if** the difference between  $m$  and  $m'$  is small **then**
  - 7:   Assume the trajectory of the tangent point is linear from time  $t_0$  to  $t_1$ , and return this trajectory.
  - 8: **else**
  - 9:   SplitTangent( $t_0, \frac{t_0 + t_1}{2}$ ).
  - 10:   SplitTangent( $\frac{t_0 + t_1}{2}, t_1$ ).
  - 11:   Return the trajectories found in the above two SplitTangent() calls.
  - 12: **end if**
- 

straight-line trajectory of a tangent point is first mapped to a coordinate system in which the given camera center is the origin, and the angle that the tangent point makes with the camera center is used to represent its trajectory. Accordingly, at time  $t \in [t_0, t_1]$ , where time instance  $t_0$  and  $t_1$  mark the beginning and end of the predicted trajectory respectively,  $f_{c,\tau}(t)$  returns the angle,  $\theta$ , that the tangent point  $\tau$  makes with camera center  $c$ . Utilizing a parametric straight-line representation, the general form of  $f_{c,\tau}(t)$  can be shown to be:

$$f_{c,\tau}(t) = \arctan2((t - t_0)y_0 + (t_1 - t)y_1, (t - t_0)x_0 + (t_1 - t)x_1), \quad (6)$$

where  $(x_0, y_0)$  and  $(x_1, y_1)$  are the positions corresponding to  $t_0$  and  $t_1$  respectively, and  $\arctan2(y, x)$  is the four-quadrant inverse tangent function over the range  $-\pi$  to  $\pi$ . Here,  $\theta = f_{c,\tau}(t)$  for  $t_0 \leq t \leq t_1$  defines a  $t$ -monotone

curve segment on the  $\theta$ - $t$  plane. Special care has to be taken for degenerate cases where the segment is not continuous. First, if the tangent point passes through the camera center,  $f_{c,\tau}(t)$  is changed by  $\pm\pi$ . Second, it is possible for  $\theta$  to wrap around between  $-\pi$  and  $\pi$ , which for example can happen when the tangent point passes through the negative portion of the  $X$ -axis on the  $XY$  plane, as illustrated in Fig. 4. Both degenerate cases can be handled by splitting the curve into segments.

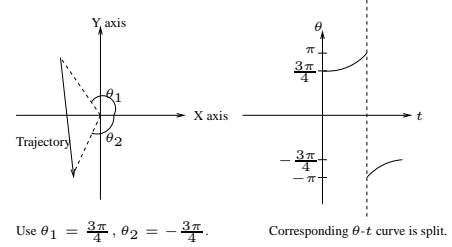


Figure 4. An example of handling wrap around segments on the  $XY$  plane.

### 3.1. Occlusion Intervals

The construction of occlusion intervals begins with finding all intersection points between the set of curve segments. This is performed in the  $\theta$ - $t$  planes corresponding to the  $XY, XZ$  and  $YZ$  planes respectively. Each intersection point in the respective  $\theta$ - $t$  plane represents an occlusion situation involving a pair of objects, of which the corresponding tangent points have the same angular displacement from the given camera center, causing the farther object from the camera to be occluded. Occlusion intervals are then constructed as time intervals spanning these intersection time instances, which can be found by solving for the intersections between a given pair of trajectories, represented as given in Eqn. 6. These intersections can be shown to be the solutions of a quadratic function of time  $t$  (i.e., in the form  $at^2 + bt + c = 0$ ), thus implying that two trajectories can lie on the same line of sight of a given camera at most twice. More importantly, it is much simpler to solve such a quadratic function, as opposed to solving for the intersections between a given pair of trajectories represented in the form of Eqn. 2, which leads to a polynomial in  $t$  of degree six.

Determining these intersections in a brute force manner incurs  $O(N^2)$  running time, which is acceptable when  $N$  is small. For large  $N$ , we propose the following optimal segment intersection algorithm. The set of curve segments,  $L$ , on the  $\theta$ - $t$  plane that spans the temporal interval  $[t_0, t_1]$ , is sorted according to the values of  $\theta$  at which they intersect the vertical line  $t = t_0$ . The resulting sorted set,  $L_{sorted}$ , is then recursively divided into two sets -  $Q$  containing curve segments that do not intersect each other and  $L'$  containing the rest, using Algorithm 2. The proof that  $Q$  contains only segments that do not intersect each other can be verified by the following theorem:

**Theorem 2** Let  $s_{i=N \dots 1}$  be the new set of segments added to  $Q$  (refer to Algorithm 2) at each recursion, sorted in descending order by the values of  $\theta$  at which  $s_i$  intersects  $t = t_0$ . If  $s_i$  does not intersect  $s_j$ , for  $j > i$ , then  $\forall \ell > j$ ,  $s_i$  does not intersect  $s_\ell$ . Conversely, if  $s_i$  does not intersect  $s_j$ , for  $j < i$ , then  $\forall \ell < j$ ,  $s_i$  does not intersect  $s_\ell$ .

**Proof** For  $j > i$ , if  $s_i$  does not intersect  $s_j$ , then it must be true that  $\forall t, f_i(t) < f_j(t)$ , where  $f$  is a function that returns the  $\theta$  value of a curve segment at time  $t$ . Since  $s_{j+1}$  does not intersect  $s_j$  (a segment is added to  $Q$  only if it does not intersect the previously added segment), then  $\forall t, f_j(t) < f_{j+1}(t)$  - i.e.,  $\forall t, f_i(t) < f_{j+1}(t)$ . It follows easily that,  $\forall \ell > j$ ,  $s_i$  does not intersect  $s_\ell$ . The converse can be similarly proven.  $\square$

At the end of every recursion, curve segments in set  $Q$  and  $L'$  are checked for intersections with each other. An additional set,  $Q'$ , contains the index of the conflicting segment in  $Q$  whenever a segment is added to  $L'$ . Additionally, depending on the amount of splitting caused by degenerate cases (Fig. 4), and transforming the tangent point trajectories into straight lines (Algorithm 1), the intersections might have to be determined for multiple time intervals, so as to ensure that a curve segment spans the time interval it is being processed in. For this purpose, the endpoints of all curve segments are sorted so that any given curve segment is guaranteed to span one of the time intervals formed by consecutive time instances in the sorted list.

The complexity of the algorithm is  $O(N \log N + I)$ , where  $I$  is the number of intersection points, and  $N$  the number of curve segments - the sorting stage takes  $O(N \log N)$ , populating  $Q$ ,  $L'$  and  $Q'$  takes  $O(N)$ , and the intersection-finding stage takes  $O(I)$ . The algorithm is output-sensitive, since its running time depends on the number of intersections, making it particularly useful when the number of intersections is small. The guarantee that the intersection-finding stage in Algorithm 2 is an  $O(I)$  operation can be easily verified. The intersection-finding stage checks for intersections of each element of  $L'$  with segments beginning from the index of the corresponding conflicting segment in  $Q$ , as given by  $Q'$ . The iterations are performed in both "directions", one decrementing and the other incrementing from the index of the conflicting segment, stopping when the segments do not intersect. The stopping condition is due to Theorem 2, and thus ensures that the total number of checks conducted are  $O(I)$ .

### 3.2. Visibility Intervals

After determining the occlusions intervals, we construct the visibility intervals as their set complements. Multiple occlusion intervals resulting from different objects occluding the same object during different temporal intervals are dealt with by combining their set complements. Such a procedure is performed for the projections on the  $XY$ ,  $XZ$  and  $YZ$  planes respectively, and the overlapping regions between the visibility intervals on the respective planes, after discarding those with durations smaller than the required processing time of the tasks, are the final visibility intervals.

---

### Algorithm 2 FindIntersections( $t_0, t_1, L_{sorted}$ )

---

```

1: {Let  $L_{sorted} = \{s_N, \dots, s_1\}$ .
2:  $L' = \emptyset$ .
3:  $Q = \emptyset$ .
4:  $Q' = \emptyset$ .
5: for  $i = N, \dots, 1$  do
6:   if the segment  $s_i$  doesn't intersect the last segment of  $Q$  then
7:     Add  $s_i$  to the end of  $Q$ .
8:   else
9:     Add  $s_i$  to the end of  $L'$ .
10:    Add the index of the conflicting segment in  $Q$  to  $Q'$ .
11:   end if
12: end for
13: if  $L' \neq \emptyset$  then
14:   {Intersection-finding stage}.
15:   {Let  $L' = \{s'_k, \dots, s'_1\}$ , and  $Q' = \{ind_k, \dots, ind_1\}$ }
16:   for  $j = k, \dots, 1$  do
17:     for  $\ell = ind_j, ind_j - 1, \dots, 1$  do
18:       if  $s'_j$  intersects  $s_\ell$  then
19:         Compute the intersection and report it.
20:       else
21:         Break the loop {Ensures  $O(I)$  for finding intersections}.
22:       end if
23:     end for
24:     for  $\ell = ind_j + 1, \dots, N$  do
25:       if  $s'_j$  intersects  $s_\ell$  then
26:         Compute the intersection and report it.
27:       else
28:         Break the loop.
29:       end if
30:     end for
31:   end for
32:   FindIntersections( $t_0, t_1, L'$ ).
33: end if

```

---

There could also be more intersection points between two objects than the endpoints of valid occlusion intervals though, as shown in Fig. 5. An intersection time,  $t_{i,j}^*$ , between object  $i$  and  $j$ , that is not the endpoint of any occlusion interval, however, possesses the following distinguishing property:

$$[f_{c,i}(t_{i,j}^* \pm \Delta t), f'_{c,i}(t_{i,j}^* \pm \Delta t)] \cap [f_{c,j}(t_{i,j}^* \pm \Delta t), f'_{c,j}(t_{i,j}^* \pm \Delta t)] \neq \emptyset, \quad (7)$$

where  $f_{c,i}$  and  $f'_{c,i}$  are the trajectories (in polar coordinates) of the respective pair of tangent points of object  $i$ . Note that each pair of these trajectories form an upper and lower bound on the angles that the extent of object  $i$  makes with the camera center  $c$ .  $f_{c,j}$  and  $f'_{c,j}$  are similarly the trajectories of the pair of tangent points belonging to object  $j$ , and  $\Delta t$  is a small time step.

## 4. Camera Scheduling

Given the set of visibility intervals in the form of Eqn. 1 that have been constructed, the scheduling problem is then to decide: (1) which visibility intervals should be executed, and (2) given the set of visibility intervals chosen for exe-

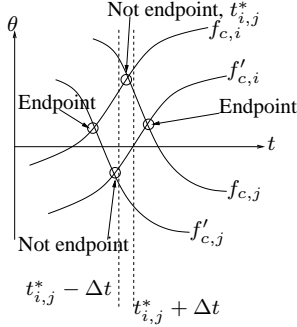


Figure 5.  $t_{i,j}^*$  is a valid intersection point between object  $i$  and  $j$ , but not a valid endpoint of an occlusion interval. The interval formed by  $f_{c,i}$  and  $f'_{c,i}$  at a small time interval  $\Delta t$  away from  $t_{i,j}^*$  intersects the interval formed by  $f_{c,j}$  and  $f'_{c,j}$  as given in Eqn. 7.

cution, what the order of execution should be, so as to maximize the coverage of tasks. Additionally, a schedule can only be executed if it is temporally feasible, determined as follow. We first define the slack  $\delta$  of a visibility interval as:

$$\delta = [t_{\delta}^-, t_{\delta}^+] = [t^-, t^+ - p], \quad (8)$$

where  $t^-, t^+$  and  $p$  are as given in Eqn. 1. Intuitively, the slack is just a temporal interval during which a task can be started. Then, a schedule containing a sequence of visibility intervals, each with slack  $\delta_i = [t_{\delta_i}^-, t_{\delta_i}^+]$ , and required duration of the task  $p_i$ , where  $i = 1 \dots n$  represents the order of execution, is only feasible if:

$$t_{\delta_1}^- + p_1 \leq t_{\delta_2}^+ \wedge t_{\delta_2}^- + p_2 \leq t_{\delta_3}^+ \wedge \dots \wedge t_{\delta_{n-1}}^- + p_{n-1} \leq t_{\delta_n}^+. \quad (9)$$

The scheduling problem is NP-hard, and computationally feasible solutions can only be obtained with approximation algorithms, for which we propose a greedy algorithm. The greedy algorithm iteratively picks the next visibility interval that has an uncovered task from a list of constructed visibility intervals sorted by their slacks' start times for each camera. Such a greedy algorithm is clearly sub-optimal. However, the following performance bound holds:

**Theorem 3** *Given  $k$  cameras, the approximation factor for multi-camera scheduling using a sub-optimal algorithm is  $1 + k\lambda$ , where the definition of  $\lambda$  is given in the proof.*

**Proof** Let  $G = \bigcup_{i=1 \dots k} G_i$ , where  $G_i$  is the set of visibility intervals assigned to camera  $i$  by the sub-optimal algorithm, and  $OPT = \bigcup_{i=1 \dots k} OPT_i$ , where  $OPT_i$  is the set of visibility intervals assigned to camera  $i$  in the optimal schedule. We further define (1)  $H_1 = \bigcup_{i=1 \dots k} H_{1,i}$ , where  $H_{1,i}$  is the set of visibility intervals for camera  $i$ , that has been chosen by the optimal schedule but not the sub-optimal algorithm and each of these tasks is not covered by the sub-optimal algorithm in any of the cameras, (2)  $H_2 = \bigcup_{i=1 \dots k} H_{2,i}$ , where  $H_{2,i}$  is the set of visibility intervals for camera  $i$ , that has been chosen by the optimal schedule but not the sub-optimal algorithm, and each of these tasks

is also covered by the sub-optimal algorithm, and finally (3)  $OG = OPT \cap G$ . Clearly,  $OPT = H_1 \cup H_2 \cup OG$ . Also,  $|H_{1,i}| = n_i$ , where  $n_i$  is the number of visibility intervals in  $H_{1,i}$ , since each visibility interval contains one task and no task is scheduled more than once in the optimal schedule. This leads to  $|H_1| = n_1 + n_2 + \dots + n_k \leq kn$ , where  $n = \max(n_i)$ . We can then define  $\lambda = \frac{n}{|G|}$ , making  $|H_1| \leq k\lambda|G|$ . Since  $|H_2 \cup OG| \leq |G|$ ,  $|OPT| \leq (1 + k\lambda)|G|$ .  $\square$

Theorem 3 shows that besides depending on the number of tasks it does not cover ( $H_1$ ), the performance of a sub-optimal algorithm would also depend on the "distribution" ( $\lambda$ ) of these uncovered tasks over the cameras. Due to this observation, the greedy algorithm also chooses visibility intervals belonging to a camera with the least number of chosen visibility intervals at each iteration.

## 5. Results

**Simulations** We conducted simulations comparing the performance of the brute force segment intersection algorithm and the optimal segment intersection algorithm. In the simulations, we use a scene of size  $50m \times 50m$ , with one camera located in the middle of the left border. We assume the camera's field of regard covers the whole scene. A fixed radius is initialized for the physical extent while the positional uncertainty is modeled by increasing that radius over the prediction period so that the confidence interval remains at 90%, using the algorithm in Sec. 2.2. For realistic simulations, observed trajectories of real objects were used as inputs to the simulations. The speed of using the optimal segment intersection algorithm and the brute force algorithm is compared in Fig. 6(a)-(c) for prediction time of 2, 5 and 10 seconds respectively. We can see that for a typical prediction time of 2-5 seconds, the breakeven point is at approximately 40 to 50 circles. Since each object is represented by a triplet of circles (Eqn. 2), the optimal segment intersection algorithm would outperform the brute force algorithm when there are approximately  $\frac{15}{V}$  objects,  $V$  being the number of views. We also show in Fig. 6(d) that the number of intersection points is much fewer than  $O(N^2)$ , even when the prediction time was as long as 30 seconds, showing that using an output-sensitive algorithm is more favorable than a brute force algorithm.

**Looking for Visible Objects** To test the capabilities of the system in predicting when objects are visible in a field of regard, we conducted experiments on a highly cluttered video sequence consisting of three different views of four moving persons in a scene. In each view, images of the persons that were predicted to be visible in the current frame were enclosed with green bounding boxes. A tracker that is very effective under severe occlusions, developed in [10], was used. The results are shown in Fig. 7(a).

**PTZ Camera System** We have also conducted real-time experiments with a prototype system consisting of four PTZ cameras, synchronized with a four-channel Matrox card. The PTZ setting of a camera assigned to a visibility interval is adjusted so that the center of the image of the targeted

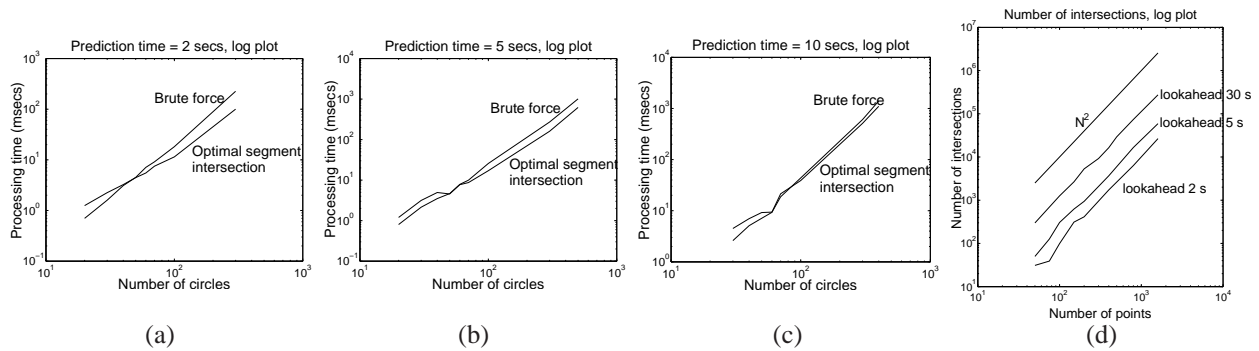


Figure 6. (a)(b)(c) The number of moving circles, at which the optimal segment intersection algorithm outperforms the brute force algorithm is showed for prediction time of 2, 5 and 10 seconds. The breakeven point for a typical prediction time of 2 secs is approximately 40. We show in (d) that the number of intersections between moving points is much fewer than  $O(N^2)$ , making an output-sensitive algorithm much more favorable.

object (or, region of the object, e.g. face, that is to be captured) coincides approximately with the camera's principal point, where the zoom setting can then be adjusted to obtain the maximum resolution. The targeted object is kept within the field of view by projecting an approximate 3D size estimate of the object, recovered from ground plane and camera calibration information, onto the camera's image frame and checking that the image of the object lies within the image boundaries. Fig. 7(b) shows two PTZ cameras controlled by a static detection camera to capture video sequences of two moving persons, so that their faces are visible. For this purpose, the constructed motion models are used to determine when they are unobstructed and moving towards the camera. In (c), two PTZ cameras were controlled to capture unobstructed full-body shots of three moving persons. Using the greedy scheduling algorithm, the first row of images shows both PTZ cameras assigned to two of the people, one of which became obstructed in the second row of images, causing the PTZ camera which became available to be assigned to the visibility interval associated with the third person. Accompanying videos are provided for the results shown in Fig. 7.

## 6. Conclusions

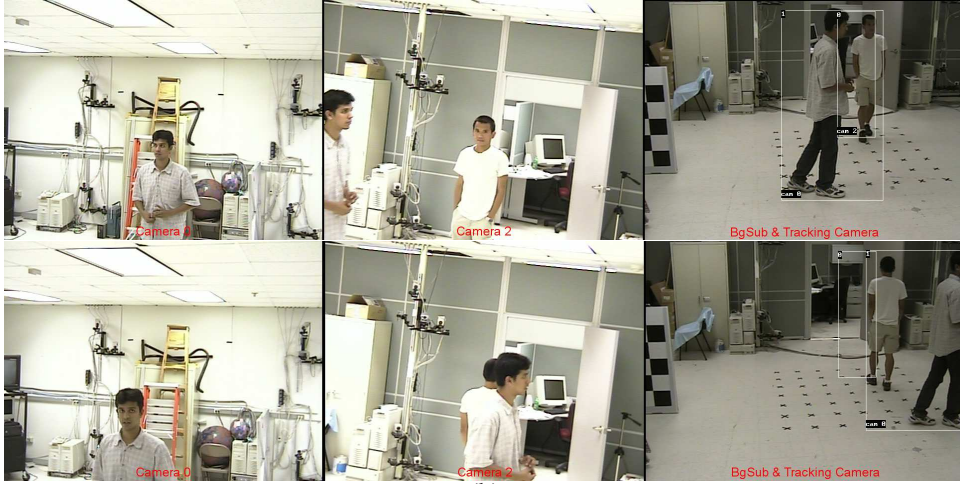
We have described a multi-camera system that constructs task-specific visibility intervals, during which a targeted object is unobstructed. We use an object representation that combines the physical size and positional uncertainty of the object. An analytic form of the motion model is used in an optimal segment intersection algorithm to construct the visibility intervals. Following the construction of these visibility intervals, a collection of cameras are scheduled for video collection. A greedy algorithm has been proposed and an analysis of its performance is given. Such a system should be useful in surveillance, where extensive camera planning and scheduling is necessary.

## References

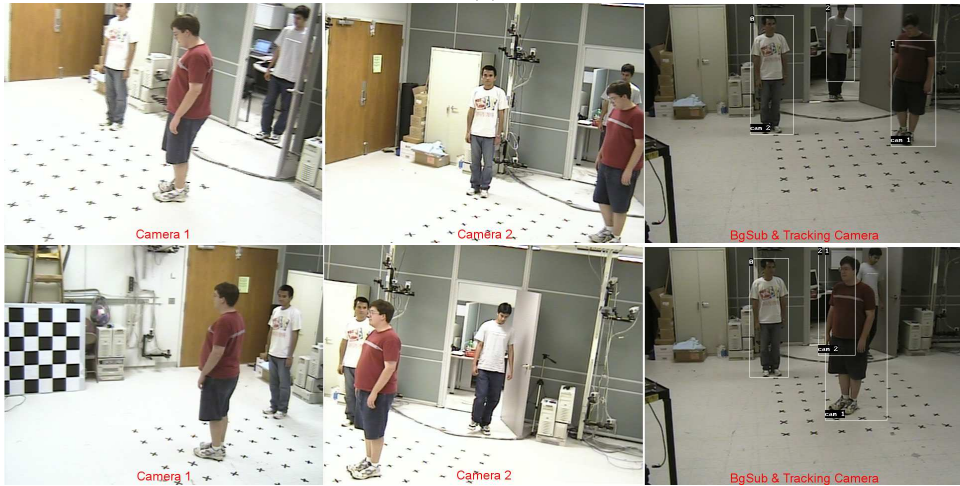
- [1] K.A. Tarabanis, P.K. Allen, and R.Y. Tsai, "A survey of sensor planning in computer vision," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 1, pp. 86–104, 1995. 2
- [2] Cregg K. Cowan and Peter D. Kovesi, "Automatic sensor placement from vision task requirement," *IEEE Transactions on Pattern Analysis and machine intelligence*, vol. 10, no. 3, pp. 407–416, 1988. 2
- [3] I. Stamos and P. Allen, "Interactive sensor planning," in *Computer Vision and Pattern Recognition Conference*, Jun 1998, pp. 489–494. 2
- [4] K. Tarabanis, R.Y. Tsai, and A. Kaul, "Computing occlusion-free viewpoints," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 3, pp. 279–292, 1996. 2
- [5] Steven Abrams, Peter K. Allen, and Konstantinos A. Tarabanis, "Dynamic sensor planning.," in *ICRA (2)*, 1993, pp. 605–610. 2
- [6] Steven Abrams, Peter K. Allen, and Konstantinos Tarabanis, "Computing camera viewpoints in an active robot work cell," *International Journal of Robotics Research*, vol. 18, no. 2, February 1999. 2
- [7] K.A. Tarabanis, R.Y. Tsai, and P.K. Allen, "The mvp sensor planning system for robotic vision tasks," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 1, pp. 72–85, February 1995. 2
- [8] Anurag Mittal and Larry S. Davis, "Visibility analysis and sensor planning in dynamic environments," in *European Conference on Computer Vision*, May 2004. 2
- [9] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry*, Springer, 1997. 3



(a)



(b)



(c)

Figure 7. (a) Views from three different cameras, extracted from running our visibility planning algorithm on a highly cluttered scene, are shown from left to right in each set, each of which was extracted from the video at different time instances. Green bounding boxes were drawn automatically around objects that have been previously predicted to be visible in the current frame. (b) In the first row, the motion models of two persons in the scene were used to determine when they are front-facing to the assigned camera (two movable cameras are used here), so that their faces are visible in the capture videos. This is clearly illustrated in the leftmost and middle image, since each person is front-facing to only one of the movable cameras, which was then assigned to the task accordingly. In the second row, one person was occluded and was not scheduled for capture. (c) In the first row, two available PTZ cameras were assigned to two of the three moving persons, one of which became obstructed in the view of its assigned camera, as shown in the middle image of the second row. The camera that becomes available as a result was then assigned to the visibility interval associated with the third person, who was still not captured by the system. For (b) and (c), note that the camera label in each bounding box (rightmost image) indicates the camera that was assigned to capture the person.

[10] Anurag Mittal and Larry S. Davis, “M2tracker: A multi-view approach to segmenting and tracking people in a cluttered scene using region-based stereo,” *7th*

*European Conference on Computer Vision, Copenhagen, Denmark, Jun 2002. 6*