Multimedia Systems manuscript No.
(will be inserted by the editor)

Ser-Nam Lim · Larry S. Davis · Anurag Mittal

Constructing Task Visibility Intervals for Video Surveillance

Abstract Vision systems are increasingly being deployed to perform complex surveillance tasks. While improved algorithms are being developed to perform these tasks, it is also important that data suitable for these algorithms be acquired - a non-trivial task in a dynamic and crowded scene viewed by multiple PTZ cameras. In this paper, we describe a real-time multi-camera system that collects images and videos of moving objects in such scenes, subject to task constraints. The system constructs "task visibility intervals" that contain information about what can be sensed in future time intervals. Constructing these intervals requires prediction of future object motion and consideration of several factors such as object occlusion and camera control parameters. Such intervals can also be combined to form multi-task intervals, during which a single camera can collect videos suitable for multiple tasks simultaneously. Experimental results are provided to illustrate the system capabilities in constructing such task visibility intervals, followed by scheduling them using a greedy algorithm.

Keywords Surveillance · Sensor Planning · Sensor Fusion · Active Camera

1 Introduction

We describe a sensor planning system for controlling, in real time, a collection of surveillance cameras to acquire video sequences of moving objects (people, vehicles), subject to visibility, resolution and positional constraints. Our approach, in general, 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, schedule the tasks at hand, re-position cameras and acquire videos to support the scheduled tasks. The demand for such video acquisition is motivated by the following surveillance scenario.

We are given a collection of calibrated surveillance cameras. They must be controlled to acquire surveillance video over a large surveillance site, which can most simply be modeled as a large patch of ground plane, possibly annotated with the locations of specific regions of interest (e.g., regions near the entrances to buildings, receptacles such as trash cans, or regions defined dynamically as vehicles enter and stop in the site).

Each camera has a field of regard, which is the subset of the surveillance site that it can image by controlling its viewing angles (Pan, Tilt and Zoom - PTZ - settings). A field of view of a camera is the image obtained at specific PTZ settings and is generally much smaller than its field of regard.

As people and vehicles move into and through the surveillance site, the cameras are to be controlled to acquire sets of videos that satisfy temporal and positional constraints that define generic surveillance tasks. 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. The video must be collected at some minimal resolution. 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.

One could imagine other surveillance tasks that would be defined to support face recognition, loading and unloading of vehicles, etc. Additionally, there are tasks related to system state maintenance - for example, tasks to image a person or vehicle to obtain current position data to update a track predictor such as a Kalman filter ([9]); or tasks to intermittently monitor regions in which people and vehicles can enter the surveillance site. We would like to efficiently

F. Author · S. Author
Department of Computer Science
University of Maryland, College Park, Maryland, USA
E-mail: {sernam,lsd}@cs.umd.edu

T. Author
Computer Science and Engineering Department
Indian Institute of Technology Madras, Chennai, India
E-mail: amittal@cse.iitm.ernet.in

schedule as many of these surveillance tasks as possible, possibly subject to additional constraints on priority of the tasks.

The problem of sensor planning and scheduling has been studied extensively. [18] presented a survey covering sensing strategies for object feature detection, model-based object recognition, localization and scene reconstruction. One of the earliest works is [5] which introduced a locus-based approach to decide on the placement of viewpoints, subjected to static resolution, focus, field of view and visibility constraints. They also described an extension of the sensing strategy to laser-scanner range sensor. [17] introduced the idea of local separating planes which are used to define visibility volumes, in which occlusion-free viewpoints can be placed. Then, to satisfy the field of view constraint, they introduced the idea of the field of view cone, which is similar to the locus-based approach given in [5]. These papers did not consider object motion. [1,2,19] discusses a dynamic sensor planning system, called the MVP system. They were concerned with objects moving in the scene, generating a swept volume in temporal space [2]. 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. This is then integrated with other constraints such as focus and field of view in [19]. The culmination is found in [1], where the algorithms are applied to an active robot work cell. [14] determines optimal sensor placements offline, by considering information that provides probabilistic priors of object motion - observations made about the motion of objects in the surveillance site are probabilistically used as inputs to placing sensors offline. This ensures (probabilistically) that cameras are placed in positions that will have unobstructed views of moving objects. Finally, studies on sensing strategies for the purpose of 3D reconstruction can be found in [10–12].

In comparison, our main contribution lies in the real-time construction of time intervals during which visibility constraints are satisfied on a per-camera basis. The construction of the time intervals are based on predicted object trajectories so that they can be used to schedule active cameras for video collection "ahead of time". Several papers have discussed these camera scheduling algorithms, such as [4] which evaluated scheduling policies like First Come First Serve (FCFS) and Earliest Deadline First (EDF), or, [16] which evaluated a weighted round-robin scheduling scheme with a static FCFS policy using a Virtual World Simulator. For efficiency, one could also consider the parallel scheduling algorithm described in [15]. Our focus, however, is on the algorithms for constructing these predicted time intervals. While our experiments demonstrate the applicability of using the constructed time intervals for greedily scheduling the cameras, other scheduling algorithms could be used. The basic idea of scheduling cameras based on predicted object trajectories was introduced previously in [13], which was mainly concerned with handling occlusions between moving objects in the scene. Later, [6] described a method to determine feasible PTZ settings based on predicted object trajectories. In contrast, our algorithm is an holistic approach that considers both predicted occlusions between moving objects and feasible camera settings during video collection. Specifically, our scheduling approach is based on the efficient construction of what we call Task Visibility Intervals (TVI's). A TVI is a 4-tuple:

$$(c, (T, o), [r, d], Valid_{\psi,\phi,f}(t)). \tag{1}$$

Here, c represents a camera, (T, o) is a (task, object) pair - T is the index or specification of a task to be accomplished and o is the index of the object to which the task is to be applied, and $[r, d]$ is a time interval within which (some temporal segment of) the task can be accomplished using camera c. r is the earliest release time of the task while d is the deadline by which the task has to be completed. Then, for any time instance $t \in [r, d]$, $Valid_{\psi,\phi,f}(t)$ is the set of pan settings (ψ), tilt settings (ϕ) and focal lengths (f) that camera c can employ to capture object o at time t.

We focus our attention on tasks that are satisfied by video segments in which an object is seen unobstructed for some task-specific minimal period of time, and is viewed at some task-specific minimal resolution during that time period. The task specifications, themselves, are 3-tuples:

$$(p, \alpha, \beta) \tag{2}$$

where

1. p is the required duration of the task, *including worst-case latencies involved in re-positioning cameras*,
2. α is a predicate relating the direction the object is moving relative to 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
3. β is the minimal ground resolution needed to accomplish the task.

In general, not all tasks would be applicable to all objects - one can imagine tasks for viewing faces of people, license plates of vehicles, etc. For simplicity, we assume that all tasks are to be accomplished for all objects in the surveillance site. Practically, one would also need some mechanism to verify, a posteriori, that the tasks have been successfully completed (i.e., that in fact we have obtained unobstructed video of some given object) so that it can be determined if the task has to be rescheduled (which, of course, will not always be possible). Ideally, the measurement stage of a suitable tracker can be used for such verification purpose.

2 System Overview

Our camera control system cycles through three stages of analysis (Fig. 1) :

1. A sensing stage, in which moving objects are tracked through the surveillance site using wide field of view cameras. Based on image analysis and calibration information, the physical size (height and width) of each object is estimated. For computational efficiency, in 2(a)

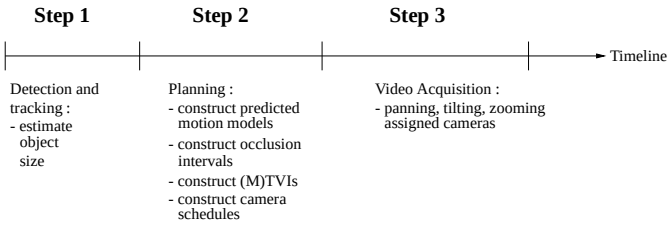


Fig. 1 Timeline depicting the steps involve in collecting task-specific video sequences.

and (b), the height and width are used to construct circumscribing circles to the object's orthographic projections on the projection planes (referred to below as "shadows") of the world coordinate system for path prediction and visibility analysis. These shadows are subsequently used for constructing an ellipsoidal representation of the object in 2(c) for determining task-specific feasible camera settings.

2. A planning stage, composed of five sub-stages:
   (a) A prediction stage, in which the tracks are extrapolated into the future. The predicted tracks are straight lines. Additionally, a variance measure is estimated for the track and incorporated into the shadows of the object volume. So, the final predicted motion model for each moving object consists of the individual circular shadow moving along a straight line; the radius of the shadow increases linearly over time with a constant proportional to the error in "fitting" a straight line to the track of the object in the sensing stage.
   (b) A visibility analysis stage in which we determine, for each camera and moving object, the intervals of time - called visibility intervals - during which that moving object will be contained within the camera's field of regard, and not be occluded by any other moving object. This analysis is done on the projection planes where we analyze the movements of the shadows of the moving objects. The trajectories of the shadows on the projection planes are represented by piecewise linear approximations to the trajectories of the tangent points of the shadows. Over their piecewise linear segments, the trajectories of the extremal angles of the shadows with respect to the projected camera center have a simple analytic representation. We then use asymptotically efficient algorithms to find crossings of the extremal angles. This allows us to directly determine the intervals during which an object is occluded by some other object; the complements of these occlusion intervals are the visibility intervals.
   (c) A task visibility stage now combines task specific information - resolution, direction and duration - with the visibility intervals to identify time-varying camera PTZ settings that would satisfy a given task during some portion of a visibility interval. This results in so-called Task Visibility Intervals (TVI's). Generally, there could be many cameras that could be used to satisfy any given task.
   (d) A TVI compositing stage, which efficiently finds small combinations of TVI's that can be simultaneously scheduled on a single camera. We call these intervals Multiple Task Visibility Intervals (MTVI's), and determining them involves finding non-empty intersections of camera settings over suitably long time intervals for subsets of tasks and specific cameras.
   (e) A scheduling stage using a greedy algorithm.
3. A collection stage in which the cameras are first positioned and then collect the video segments for the tasks for which they have been scheduled.

Here, the sensing stage is performed using the background subtraction algorithm described in [20], and the CONDENSATION tracking algorithm from [7]. Background subtraction is performed at every frame to detect foreground blobs (which may be the images of multiple moving objects), with the assumption that objects are initially sufficiently separated from each other to be detected individually. The set of objects are then tracked, and the observed locations are used as the prior to compute the likely object positions in the next frame. The CONDENSATION algorithm allows us to generally track individual object through short periods of occlusions.

3 Motion Model

Determining visibility intervals for any given (object, camera) pair involves *predicting* future 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, the prediction of occlusion intervals would also depend on the object's shape and size. The size of the object combines our estimates of its physical size along with the time-varying uncertainty of its location, predicted from tracking.

In the world coordinate system (with axes as X, Y and Z respectively), we orthographically project the center of a given camera and the silhouette of each object at a given time, as point and circle respectively, onto the XY, YZ and XZ planes. The sizes of the circles are determined by the object's width and height, as estimated from its silhouette and pre-determined homographies from the camera's image plane to the XY, YZ and XZ planes. On each plane, a projected circle has two tangent points that define its extent with respect to the projected camera center. The motion model is then defined as the time-varying angular extents of the pairs of tangent points belonging to the triplet of circles representing the object. These projections serve as a simple representation of an otherwise complex 4D (XYZ and time) motion model. Fig. 2 shows a projected circle on the XY plane, with radius r, of an object o with respect to to the camera center c. Here, θ is the angular displacement of the circle center from

c, and the angular displacement of the upper and lower tangents can be expressed as $\xi_{upper} = \theta - \alpha$ and $\xi_{lower} = \theta + \alpha$ respectively, where $\alpha = \arcsin \frac{r}{d}$. Accordingly, the motion model of object o, $f_{c,o}(t)$, parameterized by time t becomes:

$$f_{c,o}(t) = \bigcup_{\Pi = XY, XZ, YZ} \{\theta(t, \Pi) \pm \arcsin \frac{r(t, \Pi)}{d(t, \Pi)}\}, \quad (3)$$

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 Π.

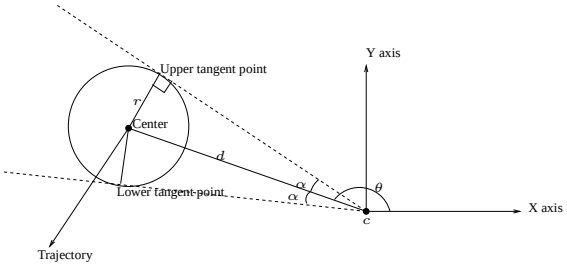


Fig. 2 Example of the object's shadow on the XY plane. Here, $\alpha = \arcsin \frac{r}{d}$.

4 Prediction Stage

Tracking information observed in the sensing stage (Step 1, Fig. 1) is used to predict the future positions of the triplet of circles (previous section) representing each object. Similar to filter-based (e.g., Kalman filter [9], extended Kalman filter [8], or particle filter [7]) trackers where known a priori distribution is used to predict a current state given the previous state, the distribution of the velocities given by the observed tracks (a priori) of each object is used to predict its future positions while providing an uncertainty measure. In addition, we particularly consider ease of computation and a geometrical interpretation of the positional uncertainties given by the a priori distribution. For these purposes, we employ a method that constructs straight-line prediction paths. The positional uncertainty, which allows variation from the straight-line path, is modeled by growing the radius of the circle linearly over time as it moves along the straight line.

Let the center of a projected circle on one of the planes be c_{obj}. Let S_{hist} be the successive positions of c_{obj} observed during the tracking interval. Subsets of S_{hist} formed from consecutive elements are used to predict the direction and speed of c_{obj}, with adjacent subsets sharing common elements. So, for example, the first to k^{th} element would belong to the first subset, the $(\eta + 1)^{th}$ to $(k + \eta)^{th}$ element to the second and so on, where $\eta < k$ and $(k - \eta)$ is the number of common elements in consecutive subsets. To determine the direction, a straight line is fit to the locations of c_{obj} in each subset, while an estimate of the speed is derived as the displacement between the first and last element of the subset divided by the corresponding time lag. Then, we form a new set, S_{pred}, consisting of the predicted velocities of c_{obj} as:

$$S_{pred} = \{x_0, x_1, ..., x_n\}, \quad (4)$$

where each $x_{i=0,...,n}$ is a 2-vector of speed and direction, and n is the number of subsets formed from S_{hist}. Each x_i is assigned a weight w_i, with more recent observations being assigned larger weights, and with all weights normalized so that $\sum_{i=0}^{n} w_i = 1$. If we assume that both speed and direction are independent of each other, the probability of observing a velocity v can then be estimated as:

$$Pr(v) = \sum_{i=0}^{n} w_i \prod_{j=1}^{2} \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{1}{2}\frac{(v_j - x_{i,j})^2}{\sigma_j^2}}, \quad (5)$$

where j represents the speed and direction component, and σ_j^2 is the corresponding bandwidth. The confidence interval, $[v_{min}, v_{max}]$, that provides a desired level of confidence, P, can be found by solving for:

$$P = \int_{v_{min}}^{v_{max}} Pr(v) dv. \quad (6)$$

$[v_{min}, v_{max}]$ is used to compute the region in which c_{obj} lies in future time instances. A Minimum Enclosing Circle (MEC) is constructed to enclose the predicted region into which the object is moving, inflated by the size of the circular shadow of the object, using the linear-time algorithm given in [3] pp. 86-90. It is easy and efficient to determine the MEC because the predicted region in which c_{obj} lies at a particular time instance is delimited by the arcs of two concentric circles, with the four endpoints of the arcs computed from the minimum and maximum speed in the "minimum" direction, and the minimum and maximum speed in the "maximum" direction, as given by $[v_{min}, v_{max}]$. This allows the MEC to be determined by just considering these four points. This is illustrated in Fig. 3.

The MEC models the positional uncertainty and physical extent of each object. Thus, if the object moves approximately along a straight line with approximately constant speed, then the subsets in S_{pred} will have similar velocities, and the computed $[v_{min}, v_{max}]$ will have a small range, giving rise to a small MEC. This is opposed to objects moving along complex trajectories (e.g., in circles, which can occur in scenes with curved pathways), in which case the MEC typically increases in size more quickly as the paths given by $[v_{min}, v_{max}]$ increasingly deviate from each other.

The predicted motion model of an object can thus be visualized as a progression of a triplet of MEC's in time. Two particularly useful observations, utilized for the construction of visibility intervals later, can be made about the series of MEC's. Firstly, each MEC moves along a straight line, and

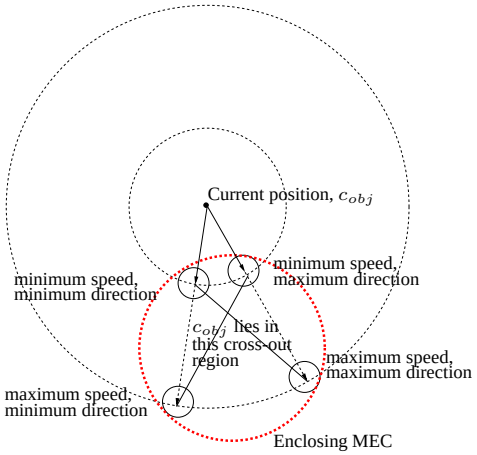


Fig. 3 In the next time instance, the region where the predicted positions of c_{obj} lie is delimited by the arcs of two concentric circles as shown, with the four delimiting corners of the region computed by the minimum and maximum speed, and the minimum and maximum direction, given by v_{min} and v_{max}. A MEC (drawn in red) can then be constructed to enclose the predicted region into which the object is moving.

secondly its radius grows linearly with time. Both properties can be easily verified by construction.

To illustrate the performance of the motion predictor, we consider a video sequence of people walking naturally in a parking lot. We select an individual walking on a curved path for tracking, and predicted his motion model on the ground plane as a series of MEC's, using observed tracks of 100 frames. The MEC at each frame is then compared with the actual position of the person, obtained by tracking him throughout the sequence. The predictions were sufficiently accurate for the required amount of time (≥ 60 frames), even though the observed individual was moving along a curved path. This is shown in Fig. 4.

5 Visibility Analysis Stage

The goal of the visibility analysis stage is to construct piecewise analytic representation of the extremal angles of the time-varying MEC for analysis by an efficient segment intersection algorithm. Time instances at which the extremal angles of the MEC's of different objects coincide delimit occlusion intervals, during which the objects are occluding one another.

If the trajectories of the tangent points of the MEC's were straight lines over time, $t \in [t_0, t_1]$, then the trajectories, $g^-_{c,i}$ and $g^+_{c,i}$, of the lower and upper extremal angles respectively, with respect to camera c and object i, would be:

$$\arctan2((t-t_0)y_0 + (t_1-t)y_1, (t-t_0)x_0 + (t_1-t)x_1), \quad (7)$$

where (x_0, y_0) and (x_1, y_1) are the positions of the tangent point at t_0 and t_1 respectively, and $\arctan2(y, x)$ is the four-quadrant inverse tangent function over the range $-\pi$ to π. Such a representation greatly simplifies the computation of occlusion intervals - when Eqn. 7 for two objects are equated, they form a simple quadratic function of time t, which is easily solved for the time instances that delimit periods of crossing between the two objects.

However, the trajectories of the tangent points are, generally, nonlinear. So, we construct piecewise linear approximations to these trajectories, using Algorithm 1 and then employ Eqn. 7 to construct the desired angular trajectories of the pieces. In Algorithm 1, the predicted motion model refers to the time-sampled values of $\theta(t, \Pi)$, $r(t, \Pi)$ and $d(t, \Pi)$ in Eqn. 3, and are easily derived due to the observations in the previous section (i.e., that the MEC's move along straight lines and grow linearly over time). The time-sampled positions of the corresponding tangent points can then be derived accordingly, so that, for example, in Fig. 2, the X and Y coordinates of the upper tangent point are given as $\sqrt{d(t,\Pi)^2 - r(t,\Pi)^2}\sin(\theta(t,\Pi) - \alpha)$ and $\sqrt{d(t,\Pi)^2 - r(t,\Pi)^2}\cos(\theta(t,\Pi) - \alpha)$ respectively.

There could also be more solutions between two objects than the endpoints of valid occlusion intervals, as shown in Fig. 5(a). A solution, $t^*_{i,j}$, between object i and j, that is not the endpoint of any occlusion interval, however, possesses the following distinguishing property:

$$[g^-_{c,i}(t^*_{i,j} \pm \Delta t), g^+_{c,i}(t^*_{i,j} \pm \Delta t)] \bigcap [g^-_{c,j}(t^*_{i,j} \pm \Delta t), g^+_{c,j}(t^*_{i,j} \pm \Delta t)] \neq \emptyset, \quad (8)$$

where Δt is a small time step.

Special care has to be taken for degenerate cases where the trajectories of the extremal angles are not continuous. First, if the tangent point passes through the camera center, the subtending angle is changed by $\pm\pi$. Second, it is possible for the subtending angle to wrap around between $-\pi$ and π, which happens when the tangent point passes through the negative portion of the X-axis on the XY plane, as illustrated in Fig. 5(b). Both degenerate cases can be handled by splitting the curve of Eqn. 7 into segments.

5.1 Determining Occlusion Intervals Efficiently

Occlusion intervals could now be determined using a brute force approach that considers all pairs of object extremal angle trajectories. Such a brute force approach incurs $O(N^2)$ running time, where N is the number of curve segments. For large N, we propose the following optimal segment intersection algorithm.

The set of curve segments, L, on the extremal angle-t plane that spans the temporal interval $[t_0, t_1]$, is sorted according to the values 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 as follows:

(a) Path taken by the tracked person

(b) Predicted MEC's using 100 frames of observed tracks.

Fig. 4 In (b), red circles represent the MEC's predicted for the following frames, while the green circle represents the predicted location in the current frame. Comparing these circles with the positions given by the tracker (blue bounding box), the prediction were sufficiently accurate, as shown in (b), for the required number of frames, even though the person was walking along a curved path as shown in (a).

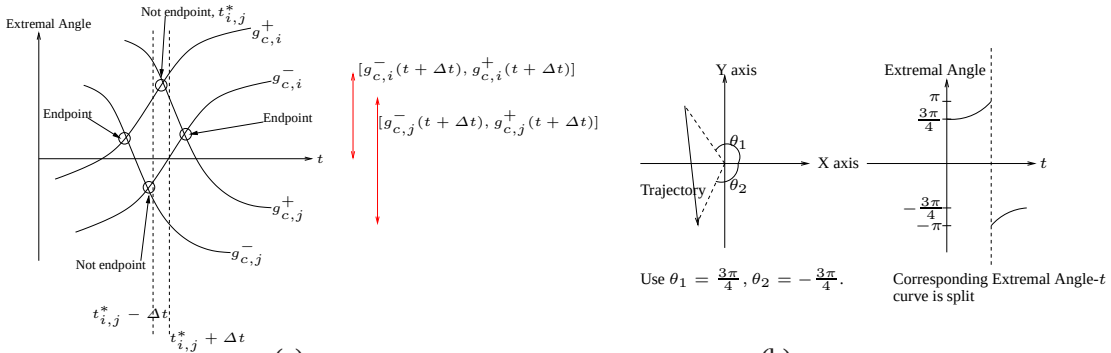


Fig. 5 (a) $t^*_{i,j}$ is a valid intersection point between object i and j, but not a valid endpoint of an occlusion interval. Shown as red arrows, the interval formed by $g^+_{c,i}$ and $g^-_{c,i}$ at a small time interval Δt away from $t^*_{i,j}$ intersects the interval formed by $g^+_{c,j}$ and $g^-_{c,j}$ as given in Eqn. 8. (b) An example of handling wrap around segments on the XY plane.

Algorithm 1 SplitTangent(t_0,t_1,Π)

1: {Π 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 Π at t_0, computed from the predicted motion model.
3: Let p_{t_1} be the position of the tangent point on Π at t_1, computed from the predicted motion model.
4: Interpolate for the midpoint, m, of p_{t_0} and p_{t_1} on Π, i.e., $m = \frac{p_{t_1}+p_{t_0}}{2}$.
5: Compute from the predicted motion model, the actual midpoint, m', on Π 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**

Proposition 1 *Let $s_{i=N\ldots1}$ be the new set of segments added to Q (refer to Algorithm 2) at each recursion step, sorted in descending order by the values 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_ℓ. Similarly, if s_i does not intersect s_j, for $j < i$, then $\forall \ell < j$, s_i does not intersect s_ℓ.*

Proof For $j > i$, if s_i does not intersect s_j, then it must be true that $\forall t, g_{c,i}(t) < g_{c,j}(t)$ (Eqn. 7). 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 $\forall t, g_{c,j}(t) < g_{c,j+1}(t)$ - i.e., $\forall t, g_{c,i}(t) < g_{c,j+1}(t)$. It follows easily that, $\forall \ell > j$, s_i does not intersect s_ℓ. The converse can be similarly proven. □

At the end of every step of the recursion, curve segments in Q and L' are checked for intersections with each other. An additional set, Q', contains the index of the intersecting segment in Q whenever a segment is added to L'. Additionally, the algorithm requires that all curve segments have common start and end time, which would be violated due to the splitting caused by degenerate cases (Fig. 5(b)), and the piecewise approximations to the tangent point trajectories. So, we break time into sub-intervals bounded by the endpoints of the curve segments, to ensure that a curve segment crosses the entire time interval in which it is processed. The num-

ber of sub-intervals is usually small, typically in the range between 5 to 8.

The complexity of the algorithm is $O(N \log N + I)$, where I is the number of intersection points - 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 intersecting 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 intersecting segment, stopping when the segments do not intersect. The stopping condition is due to Proposition 1, and thus ensures that the total number of checks conducted is $O(I)$.

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×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 of each object 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. 4. 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 times 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 MEC's. Since each object is represented by a triplet of MEC's (Eqn. 3), the optimal segment intersection algorithm outperforms the brute force algorithm when there are approximately $\frac{15}{V}$ objects, V being the number of cameras, since our visibility analysis is conducted for each camera. We also show in Fig. 6(d) that the number of intersection points is much fewer than 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.

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. The process is performed on the XY, XZ and YZ planes, and the overlapping regions between the visibility intervals on the respective planes, after discarding those with durations smaller than the required processing time of any task, are the final visibility intervals.

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

```
{Let L_sorted = {s_N, ..., s_1}}.
L' = ∅.
Q = ∅.
Q' = ∅.
for i = N, ..., 1 do
  if the segment s_i doesn't intersect the last segment of Q then
    Add s_i to the end of Q.
  else
    Add s_i to the end of L'.
    Add the index of the intersecting segment in Q to Q'.
  end if
end for
if L' ≠ ∅ then
  {Intersection-finding stage}.
  {Let L' = {s'_k, ..., s'_1}, and Q' = {ind_k, ..., ind_1}}
  for j = k, ..., 1 do
    for ℓ = ind_j, ind_j − 1, ..., 1 do
      if s'_j intersects s_ℓ then
        Compute the intersection and report it.
      else
        Break the loop {Ensures O(I) for finding intersections}.
      end if
    end for
    for ℓ = ind_j + 1, ..., N do
      if s'_j intersects s_ℓ then
        Compute the intersection and report it.
      else
        Break the loop.
      end if
    end for
  end for
  FindIntersections(t_0, t_1, L').
end if
```

6 Task Visibility Stage

6.1 3D Representation

The constructed visibility intervals can now be combined with task specific information - resolution, direction and duration - to identify time-varying camera PTZ settings that would satisfy a given task during some portion of a visibility interval, giving us a set of TVI's for each camera. For this purpose, we consider a 3D ellipsoidal object representation that can be written in the form of a quadric expression as:

$$X^T Q X = 0, \tag{9}$$

where Q is a symmetric 4×4 coefficient matrix for the quadric and X is a point on Q in homogeneous coordinates. Q is determined from the sizes of the MEC's on the projection planes at each time step, and the values of Q over time, $Q(t)$, now makes up the predicted motion model.

6.2 Obtaining TVI's

The predicted 3D motion model of each object can be used to compute feasible sensor settings which camera c can employ to capture the object over time while satisfying task

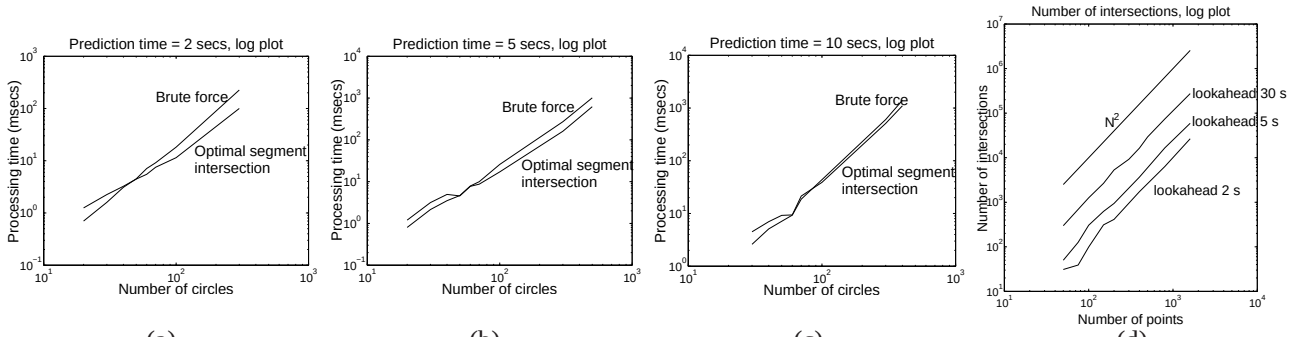


Fig. 6 (a)(b)(c) The number of moving MEC's, at which the optimal segment intersection algorithm outperforms the brute force algorithm is showed for prediction time of 2, 5 and 10 seconds. The breakeven point for a typical prediction time of 2 secs is approximately 40. We show in (d) that the number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

requirements. Each camera used in our system rotates about an axis passing (approximately) through the corresponding optical center, and is zoomable so that the focal length can be adjusted. As a result, the projection matrix P of a camera c can be written as:

$$P(R) = K[R|I], \tag{10}$$

where

- R is the rotation matrix in the world coordinate system and I is the identity matrix. P is parameterized by R as c re-positions itself by rotating in the midst of executing some capture, and
- $K = \begin{bmatrix} f_c m_x & s & x_0 \\ 0 & f_c m_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$ is the camera intrinsic matrix. Here, f_c is the focal length, (m_x, m_y) are the image scalings in the x and y directions, s is the skew factor and (x_0, y_0) is the principle point.

Then, the image of the object ellipsoid is a conic $\zeta(t) = [\zeta_1, \zeta_2, \zeta_3]$ such that:

$$\zeta^*(t) = P(R) * Q^*(t) * P^T(R), \tag{11}$$

where

- $\zeta^*(t) = \zeta^{-1}(t)$ is the dual of $\zeta(t)$ assuming full rank, and
- $Q^*(t)$ is the adjoint of $Q(t)$.

Given that $Q(t)$ represents an ellipsoid, ζ_1, ζ_2 and ζ_3 can be respectively written as $[a^2, 0, 0]^T$, $[0, b^2, 0]^T$ and $[0, 0, a^2 * b^2]^T$, where a and b are the image width and height of $\zeta(t)$.

The minimum of a and b then allows us to determine the range of focal lengths (possibly none) for which the resolution requirement of the task would be satisfied. We employ the following procedure to determine ranges of feasible camera settings for each camera c and (task, object) pair (T, o):

1. Iterate t from the start to the end of the visibility interval.
2. Iterate (ψ_c, ϕ_c) from the minimum to maximum pan and tilt settings of the camera.
3. Determine the projection matrix, $P(R)$ (Eqn. 10), where R is determined by ψ_c and ϕ_c.
4. Let $f_c = f_c^-$, where f_c^- is the shortest focal length that satisfies the minimum resolution β_{min} required by the task.
5. Perform a field of view test by checking whether the image conic (Eqn.11) lies outside the image boundaries (either partially or completely). If so, go to step 7.
6. Increment f_c and repeat step 5.
7. If $f_c \neq f_c^-$, let $f_c^+ = f_c$ since f_c now gives the maximum possible resolution while keeping the object in the field of view.
8. Update the TVI $(c, (T, o), [r, d], Valid_{\psi,\phi,f}(t))$ (Eqn. 1).

Two things that are important to note are, first, that the predicted motion model is used to compute the direction the object is moving relative to the camera pose; so the above procedure is conducted only for cameras for which the object is moving in a direction that satisfies task requirements. For example, if the task is to collect facial images, then the object must be moving towards the camera. Secondly, for computational efficiency, we use reasonably large discrete steps in t, ψ_c and ϕ_c. An interpolation algorithm is then used to construct each pair of lines representing the minimum and maximum *valid* pan settings, (ψ_c^-, ψ_c^+), on the pan-time plane, the minimum and maximum valid tilt settings, (ϕ_c^-, ϕ_c^+), on the tilt-time plane, and (f_c^-, f_c^+) on the focal-time plane, as determined by the above procedure. These projections serve as a simple representation of an otherwise complex 4D volume in PTZ and time. Illustrations are shown in Fig. 7. In (a) and (b), 3D surfaces in ψ_c, ϕ_c and f_c at $t = 0$ are shown. Both surfaces for f_c^- and f_c^+ are shown in each plot. (a) is without field of view constraint (Step 5 in the above algorithm) while (b) includes that constraint.

7 TVI Compositing Stage

The TVI's constructed above satisfy object visibility, task-specific resolution and field of view constraint for a single

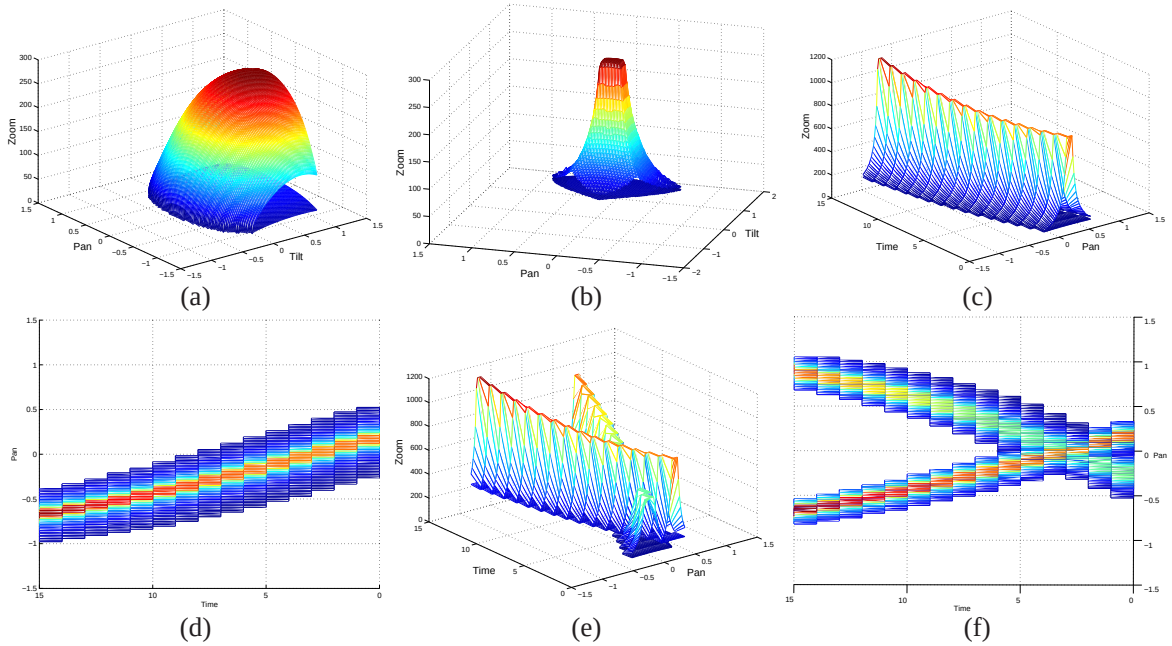


Fig. 7 Determining feasible sensor settings. (a) Without field of view test. (b) With field of view test. (c) Temporal behavior of the relations between the object motion and sensor settings. The tilt value is kept at zero in this plot. Readers should take care not to miss that there are two surfaces in these three plots: one for the maximum feasible focal length, and one for the minimum (somewhat flat surface beneath the maximum focal length surface). (d) The projection of plot (c) on the pan-time plane. (e) Two tasks shown here can be satisfied with the same sensor settings where they intersect. (f) A 2D view of (e). The start and end of the temporal interval can be obtained as the time instances where they intersect.

task. In other words, a collection of camera settings for every time step in a TVI has been computed, so that at each time step, the system can choose a zoom setting from the range of focal length allowed at a particular pan and tilt. For a given camera, subsets of TVI's can possibly be combined so that multiple tasks could be satisfied simultaneously in a single scheduled capture. The resulting intervals are called Multiple Task Visibility Intervals (MTVI's). Formally, a set of n TVI's, each represented in the form:

$$(c, (T_i, o_i), [r_i, d_i], Valid_{\psi_i,\phi_i,f_i}(t)),$$

for TVI i [Eqn. 1], can be combined into a valid MTVI represented as:

$$(c, \bigcup_{i=1\ldots n} (T_i, o_i), \bigcap_{i=1\ldots n} [r_i, d_i], \bigcap_{i=1\ldots n} Valid_{\psi_i,\phi_i,f_i}(t)), \tag{12}$$

when:

$$\bigcap_{i=1\ldots n} [r_i, d_i] \neq \emptyset,$$

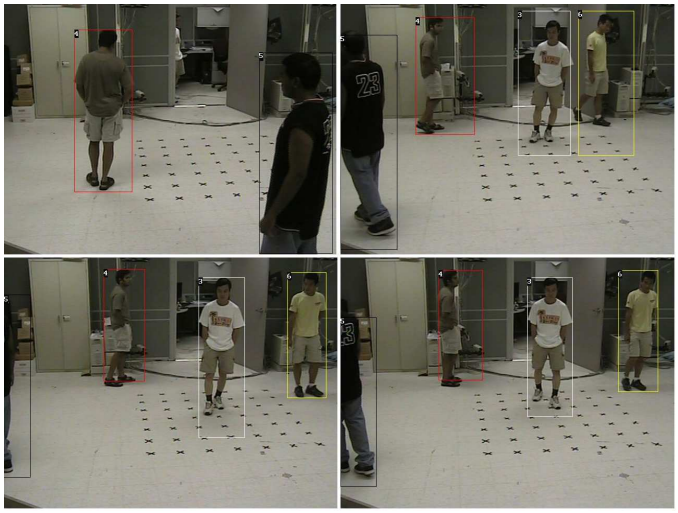
$$\bigcap_{i=1\ldots n} [r_i, d_i] \geq p_{max},$$

where p_{max} is the largest processing time among the tasks and for $t \in \bigcap_{i=1\ldots n}[r_i, d_i]$,

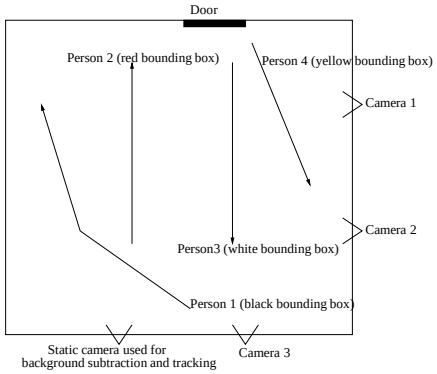
$$\bigcap_{i=1\ldots n} Valid_{\psi_i,\phi_i,f_i}(t) \neq \emptyset. \tag{13}$$

The combination of TVI's into MTVI's is illustrated in Fig. 7(c). For visualization, we kept the tilt fixed. The figure illustrates how the allowable range of focal length varies with the pan setting over time. A corresponding 2D view is shown in (d) in the pan-time plane. In (e) and (f), the plot for this task is intersected with that of another task. The resulting volumetric intersection is delimited by a temporal interval, and a region of common camera settings. Again, we utilize a simple representation of such volumes to find these common camera settings. This involves projecting them onto the 2D planes (i.e., pan-time, tilt-time and focal length-time), where the intersections can be computed efficiently.

To combine TVI's efficiently, the same optimal segment intersection algorithm described in Sec. 5.1 is used to locate the intersection points on the pan-time, tilt-time and focal length-time planes. This is effective for constructing MTVI's based on two tasks. Unfortunately, finding all feasible MTVI's for any number of tasks is computationally expensive. The system, heuristically, constructs MTVI's using the following iterative procedure, terminated after a few iterations. The set, S_2, composed of the two task MTVI's is first constructed using the optimal segment intersection algorithm. Subsequent iterations then combine elements in



(a) Sample frames used for constructing the motion model of each object are shown here. Detected objects are tracked in a CONDENSATION framework, and the observed tracks are shown in (b).The tracks are constructed over 20 frames and are used subsequently for building the predicted motion models.



(b) The observed tracks.

Fig. 8 Constructing motion models of four persons.

the set from the previous iteration, doing so only for elements containing common tasks, identifying valid combinations as those that satisfy Eqn. 13. After constructing these (M)TVI's, the system is then ready to collect video sequences that satisfy the tasks on hand, by scheduling a given collection of active cameras based on the MTVI's and TVI's (Fig. 1).

8 Results

For experimentation purposes, we developed a prototype real-time system consisting of four PTZ cameras, synchronized by a four-channel Matrox card. Experiments were performed by keeping one of the cameras static, and using it for background subtraction and tracking. The system recovers an approximate 3D size estimate of each detected object from homography-based camera calibration, and uses them to construct the (M)TVI's, which are then assigned to cameras using the greedy scheduling algorithm. Such a greedy algorithm assigns the (M)TVI that provides the maximum coverage of the tasks to an available camera at each iteration, and terminates when no additional (M)TVI's with uncovered tasks can be found, or when all available cameras have been assigned. The latencies of step 2 and 3 in Fig. 1 have to be dealt with properly. Specifically, time is "wasted" as the system plans (step 2) and the cameras assigned for capture are re-positioned in real-time (step 3) based on the PTZ settings associated with the corresponding (M)TVI's. The system deals with these latencies by adding the time required for planning and camera movement to the required processing time of the task. The latencies are, in fact, dominated by the time it takes the camera motors to stabilize after moving, so is largely independent of the angles through which the cameras are turned.

The first set of results are shown in Figs. 8 and 9, and they illustrate the system timeline. Here, there is only one task, which involves capturing unobstructed full-body video segments of all the objects at some minimal resolution. Fig. 8 illustrates how the system constructs motion models of the detected objects. Tracks of the objects observed over 20 frames (of which four frames are shown in Fig. 8(a)) are shown in a plan view in Fig. 8(b). These tracks are used to construct the predicted motion models, which are then utilized in constructing the (M)TVI's. These (M)TVI's are assigned to the three active cameras for capture based on the greedy scheduling algorithm. In the example shown in Fig. 9, (a), (b), (c) and (d) show sample frames of the captured videos. Referring to Fig. 8(b), person 3 was captured with camera 1 in Fig. 9(a), person 2 was captured with camera 2 in Fig. 9(b), and person 1 was captured with camera 3 in Fig. 9(c). The remaining person 4 was captured with camera 3, but at a different time period after camera 3 was freed up. Additionally, although the system was set to capture 60 frames of unobstructed video of each object, the processing time was specified as 80 frames so that a time period of 20 frames each is provided for camera re-positioning.

Fig. 10 then demonstrates the use of (M)TVI's for collecting facial images. Two PTZ cameras are controlled by a static detection camera to capture video sequences of two

moving persons so that their faces are visible. The predicted motion models are used to determine when people are unobstructed and moving towards a camera.

Fig. 11 and Fig. 12 illustrate the effect of changing resolution requirement on the construction of MTVI's. Four remote-controllable 12x14 inches robots moved through the scene. Fig. 11 has a lower resolution requirement than Fig. 12. The robots were controlled to move in approximately the same trajectories in both figures. While only two active cameras are needed to capture the four robots in Fig. 11, three were needed in Fig. 12 as we increase the resolution requirement.

Finally, Fig. 13 illustrates the effectiveness of the tracker to track through occlusions, allowing the prediction to be sufficiently accurate for acquiring unobstructed and well-magnified video segments of the people.

Fig. 9 Based on the predicted motion models constructed from the observed tracks given in Fig. 8, we show here sample frames of the captured video clips (sequentially from left to right) in each row. There are three active cameras available.

Fig. 10 Row 1: The motion models of two people in the scene were used to determine when they are front-facing to the assigned camera (two active cameras are used here), for face capture. This is illustrated in rows 2 and 3, where each person is front-facing to only one of the movable cameras, which was then assigned to the task accordingly. Here, the right image shows the scheduler annotating the bounding boxes with the ID of the assigned camera. The TVI of person 0 in this example is delimited by the predicted crossing with person 1. Row 2: Frames showing camera 0 capturing person 1's face. Row 3: Frames showing camera 2 capturing person 0's face.

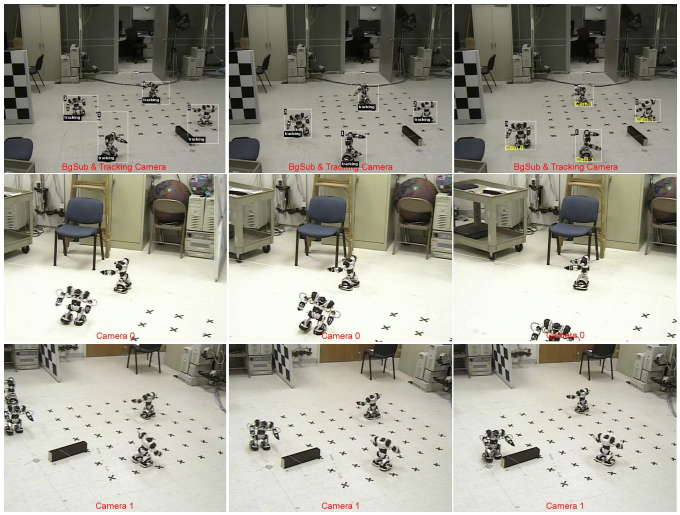


Fig. 11 Row 1: The robots are tracked (left and middle image) and assigned cameras by the scheduler (annotated in the right image). Row 2: Camera 0 captures robot 3 based on its TVI. Row 3: Due to the lower resolution than Fig. 12, a three task MTVI is sufficient for capturing robot 0, 1 and 2 simultaneously.

9 Conclusions

We have described a multi-camera system that constructs (M)TVI's as the basis for deciding suitable time periods to capture moving objects in the scene. These (M)TVI's are constructed for every camera and can be further used for more complex multi-camera planning and scheduling purposes. By constructing these (M)TVI's, the system can ensure (probabilistically, based on the predicted motion model) that targeted objects in acquired videos are unobstructed, in

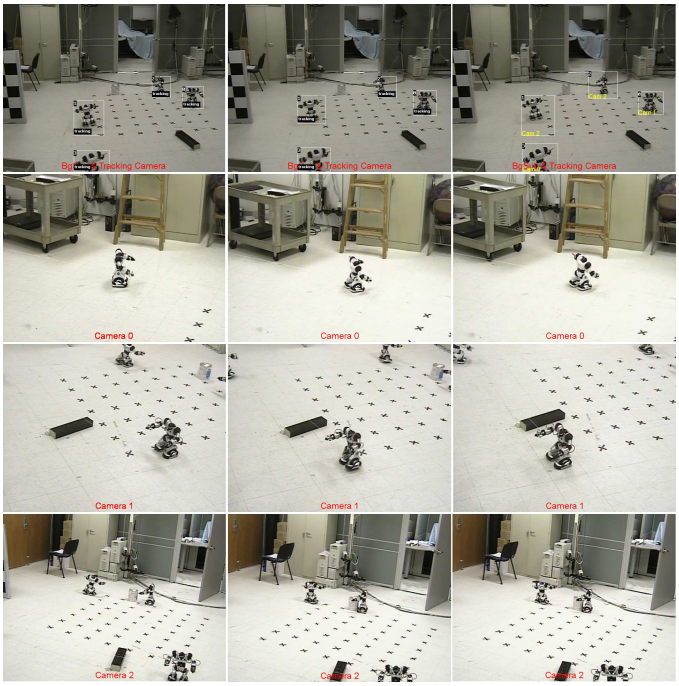


Fig. 12 The resolution requirement was increased relative to Fig. 11, and three cameras are now needed. Row 1: Tracking and scheduling. Row 2: Camera 0 captures robot 3. Row 3: Camera 1 captures robot 0. With the higher resolution requirement, robot 0 now needs to be captured alone, instead of simultaneously with robot 1 and 2. Row 4: Camera 2 captures robot 1 and 2 with a two task MTVI.

Fig. 13 Row 1: The three persons first appeared sufficiently separated to be detected individually (left image, row 1). Given only two active cameras, person 0 and 2 was scheduled first. Because the system was able to track through occlusions, shown in row 1, so that the image of person 0 was prevented from merging with those of person 1 and 2 as they were captured, the predicted motion model of person 0 was accurate enough for camera 2 to capture unobstructed and well-magnified frames of person 0 after it finished capturing person 2, shown in row 4. Row 2: Capturing person 1 with camera 0. Row 3: Capturing person 2 with camera 2. Row 4: Capturing person 0 with camera 2 after person 2.

the field of view, and meet task-specific resolution requirements. We have demonstrated the capabilities of the system, which should be useful in surveillance, where extensive camera planning and scheduling is necessary.

References

1. Abrams, S., Allen, P.K., Tarabanis, K.: Computing camera viewpoints in an active robot work cell. International Journal of Robotics Research **18**(2) (1999)
2. Abrams, S., Allen, P.K., Tarabanis, K.A.: Dynamic sensor planning. In: ICRA (2), pp. 605–610 (1993)
3. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry. Springer (1997)
4. Costello, C., Diehl, C., Banerjee, A., Fisher, H.: Scheduling an active camera to observe people. In: 2nd ACM International Workshop on Video Surveillance and Sensor Networks. New York City (2004)
5. Cowan, C.K., Kovesi, P.D.: Automatic sensor placement from vision task requirement. IEEE Transactions on Pattern Analysis and machine intelligence **10**(3), 407–416 (1988)
6. Erdem, U.M., Sclaroff, S.: Look there! predicting where to look for motion in an active camera network. In: International Conference on Advanced Video and Signal based Surveillance (2005)
7. Isard, M., Blake, A.: Condensation - conditional density propagation for visual tracking. International journal of computer vision **29**(1), 5–28 (1998)
8. Julier, S.J., Uhlmann, J.K.: A new extension of the kalman filter to nonlinear systems. In: 11th Int. Symp. on Aerospace/Defence Sensing, Simulation and Controls (1997)
9. Kalman, Rudolph, Emil: A new approach to linear filtering and prediction problems. Transactions of the ASME - Journal of Basic Engineering **82**(Series D), 35–45 (1960)
10. Kutulakos, K., Dyer, C.R.: Global surface reconstruction by purposive control of observer motion. In: IEEE Conference on Computer Vision and Pattern Recognition, Seattle, Washington, USA (1994)
11. Kutulakos, K., Dyer, C.R.: Occluding contour detection using affine invariants and purposive viewpoint control. In: IEEE Conference on Computer Vision and Pattern Recognition, Seattle, Washington, USA (1994)
12. Kutulakos, K., Dyer, C.R.: Recovering shape by purposive viewpoint adjustment. International Journal of Computer Vision **12**(2), 113–136 (1994)
13. Lim, S.N., Davis, L.S., Wan, Y.J.: Visibility planning: Predicting continuous period of unobstructed views. In: Technical Report, CS-TR 4577, Department of Computer Science, University of Maryland, College Park (2004)
14. Mittal, A., Davis, L.S.: Visibility analysis and sensor planning in dynamic environments. In: European Conference on Computer Vision (May 2004)
15. Miura, J., Shirai, Y.: Parallel scheduling of planning and action for realizing an efficient and reactive robotic system. In: 7th International Conference on Control, Automation, Robotics and Vision. Singapore (2002)
16. Qureshi, F.Z., Terzopoulos, D.: Surveillance camera scheduling: A virtual vision approach. In: 3nd ACM International Workshop on Video Surveillance and Sensor Networks. Singapore (2005)
17. Stamos, I., Allen, P.: Interactive sensor planning. In: Computer Vision and Pattern Recognition Conference, pp. 489–494 (Jun 1998)
18. Tarabanis, K., Allen, P., Tsai, R.: A survey of sensor planning in computer vision. IEEE Transactions on Robotics and Automation **11**(1), 86–104 (1995)
19. Tarabanis, K., Tsai, R., Allen, P.: The mvp sensor planning system for robotic vision tasks. IEEE Transactions on Robotics and Automation **11**(1), 72–85 (1995)
20. W.E.L.Grimson, C.Stauffer: Adaptive background mixture models for real-time tracking. In: IEEE Conference on Computer Vision and Pattern Recognition (1999)