Task-Driven Video Collection

Ser-Nam Lim* Anurag Mittal^ Larry Davis*

*CS Dept., University of Maryland, College Park ^Siemens Corporate Research, Princeton

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 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. Using a plane-sweep algorithm, these atomic intervals can be combined to form multi-task intervals, during which a single camera can collect videos suitable for multiple tasks simultaneously. Although cameras can then be scheduled based on the constructed intervals, finding an optimal schedule is a typical NP-hard problem. Due to this, and the lack of exact future information in a dynamic environment, we propose several methods for fast camera scheduling that yield solutions within a small constant factor of optimal. Experimental results illustrate system capabilities for both real and more complicated simulated scenarios.

1 Introduction

We describe a sensor planning system for which we are given a collection of calibrated surveillance cameras. Each camera has a field of regard, which is the subset of the surveillance site that it can image by controlling its viewing angles (e.g., pan and tilt settings for PTZ cameras). A field of view of a camera is the image obtained at a specific camera setting and is generally much smaller than its field of regard. The cameras must be controlled, as people and vehicles move into and through the surveillance site, to acquire videos that satisfy temporal and positional constraints that define generic surveillance tasks. Examples of typical surveillance tasks are:

- 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 ground resolution. This task might be defined to support gait recognition.
- Collect unobstructed video of any person within a given region. This might be used to secure a particular sensitive point, such as a switch.
- Observe a particular marked person continuously from at least one view, if possible. This marking may result, for instance, from an event such as a package drop.
- If two people approach one another, observing both of them simultaneously.

We would like to efficiently 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. [1] presented a survey covering sensing strategies for object feature detection, model-based object recognition, localization and scene reconstruction. One of the earliest works is [2] which introduced a locus-based approach to decide on the placement of viewpoints, subjecting to static resolution, focus, field of view and visibility constraints. They also described an extension of the sensing strategy to laser-scanner range sensor. [3] 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 [2]. These papers did not consider object motion. [4–6] 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 [4]. 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 [6]. The culmination is found in [5], where the algorithms are applied to an active robot work cell. [7] determines optimal sensor placements offline, by considering online 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 sensors are placed in positions that will have unobstructed view of moving objects. Finally, studies on sensing strategies for the purpose of 3D reconstruction can be found in [8–10].

Our scheduling approach is based on the efficient construction of what we call Task Visibility Intervals (TVIs). A TVI is a 6-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 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 some time instance $t \in [r, d]$, $Valid_{\psi,\phi,f}(t)$ is the set of different combinations of azimuth angles (ψ), elevation angles (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 tasks themselves are 3-tuples:

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

where p is the required duration of the task, including latencies involved in re-positioning cameras, α 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 β is the minimal resolution, represented with respect to the ground plane, needed to accomplish the task. Given a (task, object) pair, the trajectory of the object acquired through visual tracking along with a simple object shape model that combines size and positional uncertainty is used to compute, for each camera, c, in the camera suite, temporal intervals of visibility along with the ranges of sensor settings that can be employed to accomplish the task while keeping the object within the camera's field of view.

While camera scheduling could be conducted solely on the basis of the atomic TVIs, we can schedule more (task, object) pairs if we could efficiently identify subsets of such pairs that can be simultaneously satisfied within the same time interval. We call such time intervals Multiple Tasks Visibility Intervals (MTVIs), and we will show how a plane-sweep algorithm can be used to determine these MTVIs. Each camera can then be scheduled to execute a sequence of (M)TVIs, with the objective of maximizing task coverage. Due to the intrinsic computational complexity of scheduling, we present several approximation algorithms that yield solutions that are within a small factor of optimal. Furthermore, due to the uncertainties associated with future object motion, 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). The tracking module used for providing real-time information about the objects in the scene can be used for such verification purposes.

2 Constructing TVIs

2.1 Visibility Intervals

The first step in the construction of atomic TVIs is to determine visibility intervals for any given (object, camera) pair, which are temporal intervals during which the object remains unobstructed and within the camera's field of regard. This 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 trajectories of the objects, such occlusion intervals would also depend on the objects' physical extents. Thus, to determine these occlusion intervals, a suitable representation of object shape and size is needed, for which we use an ellipsoidal representation. The size of the ellipsoid combines our estimate of physical object size along with the time varying positional uncertainty in the location of the object, as estimated by the tracker. The computational advantage of the ellipsoidal representation is the ease with which the contour of the image of the object can be computed using the polar plane [11]. Given camera center c_p, the polar plane π w.r.t c_p is defined by the set of tangent points corresponding to the cone of rays through c_p that are in contact with the ellipsoid's contour. One can see in Fig. 1(a) that π intersects the quadric Q representing the ellipsoid in a conic C. From the equation representing C we can compute the minimum and maximum azimuth and elevation angles subtended by the conic. The motion model is then defined as the time-varying function of these vectors of azimuth and elevation angles, and can be obtained by sampling in time. One can visualize such a vector as a rectangle at each time step in $\psi - \phi$ space as shown in Fig. 1(b).

The time-varying angular extents computed from the motion models of different objects can be intersected to determine occlusion intervals. An example is shown in Fig. 1(c). To efficiently compute these occlusion intervals, we proceed as follows: First,

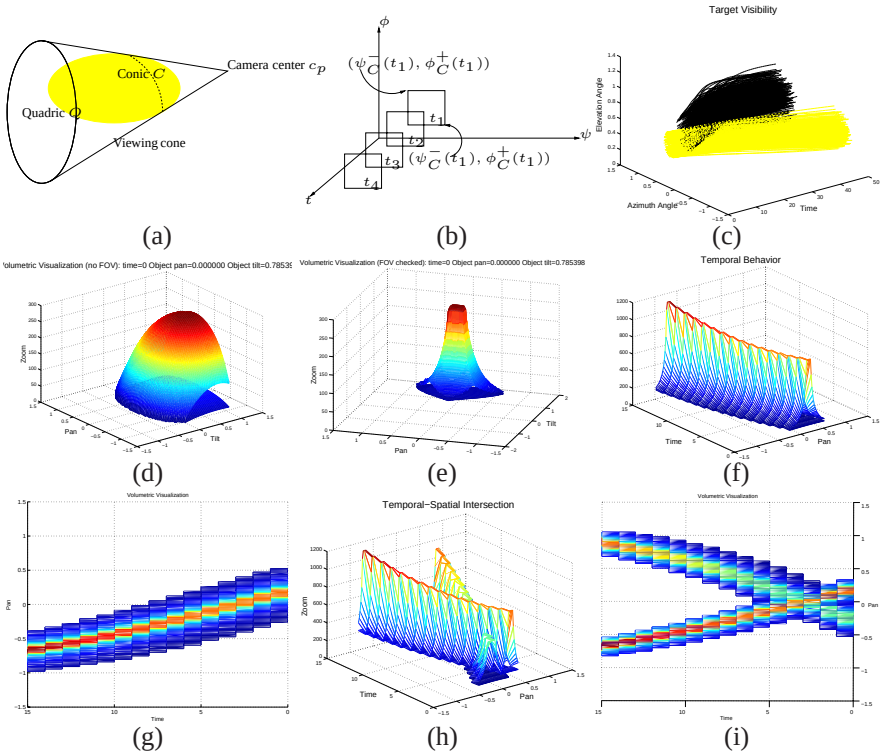

Fig. 1. (a) The contour of an object can be determined from the intersection of the polar plane w.r.t c_p with Q. Such an intersection is a conic C. (b) At each time step, the angular extents, $(\psi_C^-, \psi_C^+, \phi_C^-, \phi_C^+)$, representing the minimum and maximum azimuth and elevation angles of conic C respectively, can be visualized as a rectangle. The motion model M can be viewed as the footprints of these angular extents through time. (c) Tracked points on the contours of two real objects moving in the scene are used in a particle filter framework to construct their angular extents in time. These made up two volumes, which clearly can be closely approximated with an ellipsoidal object representation as shown and where they intersect is an occlusion interval. (d) Without field of view test. (e) With field of view test. (f) Temporal behavior of the relations between the object motion and sensor settings. The tilt value is kept at zero in this plot. (g) The two extremes of plot (f) in the ψ-t plane. (h) Two objects shown here can be captured with the same sensor settings where they intersect. (i) A 2D view of (h). The start and end of the temporal interval can be obtained as the time instances where they intersect.

for each time, t, we sort the azimuth angles by minimum azimuth angle. A simple sweeping algorithm can then identify all pairs of azimuth intervals that overlap. We construct an $n \times n$ matrix, O_t, where n is the number of objects being tracked, and mark entry (i, j) as -1 if the azimuth interval for object i overlaps the azimuth interval for object j at time t and if i is closer to the camera at time t than j. Otherwise, we mark entry (j, i). We next sort the elevation angles for all objects, again according to minimal elevation angle, and employ a sweep algorithm to find all overlapping intervals. If the elevations angle intervals for objects i and j overlap, we multiply the $(i, j)^{th}$ entry of

O_t by -1 if i is closer than j, otherwise we multiply the $(j,i)^{th}$ entry by -1; when this is completed, the $(i,j)^{th}$ entry of O_t is 1 iff object i occludes object j at time t. Finally, we project the matrix O_t on the j axis using an OR projection. Let OC_t be the result of this projection operation. $OC_t(j) = 1$ iff object j is occluded by some other object at time t. This is done for all times t; we then simply stack the OC_t vectors one atop the other, and scan the columns for visibility intervals.

2.2 Sensor Settings

Given these visibility intervals, the atomic TVIs for a given camera can be constructed by pruning such visibility intervals to satisfy task-specific resolution requirements and maintain the object within a field of view. Each camera is assumed to (1) rotate about an axis passing approximately through the corresponding camera center, and (2) be zoomable so that its focal length can be adjusted. The projection matrix P of a camera c can then be written as:

$$P(R) = K[R|T]. \tag{3}$$

Given that R is the rotation matrix in the world coordinate system, Eqn. 3 parameterizes P by R as c re-positions itself by rotating in the midst of executing some viewing schedule. T is then the translation matrix in the world coordinate system, given by the fixed-mount position of the camera, and K is the camera intrinsic matrix.

We again consider the ellipsoidal representation of an object, the image of which is a conic. The dimensions of such a conic can be used to determine whether the resolution requirement of a task is satisfied. Additionally, we can now 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 a visibility interval, and iterate (ψ_c, ϕ_c) respectively from the minimum to maximum azimuth (ψ_c^-, ψ_c^+) and elevation angles (ϕ_c^-, ϕ_c^+) (the minimum and maximum viewing angles would depend on the field of regard of the camera).
2. Determine $P(R)$ [Eqn. 3], where R is given by ψ_c and ϕ_c.
3. Let $f_c = f_c^-$, where f_c^- is the focal length that satisfies the minimum resolution β_{min} required by the task. f_c^- can be determined by adjusting the focal length in K so that the size of the image conic is β_{min}.
4. If the image conic lies outside the image boundaries (i.e., outside the field of view), go to step 6.
5. Increment f_c and repeat step 4.
6. 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.
7. Update the TVI $(c, (T, o), [r, d], Valid_{\psi,\phi,f}(t))$ [Eqn. 1], using the valid ranges of sensor settings that have been determined.

A practical consideration when utilizing this procedure is to use reasonably large discrete steps in t, ψ_c and ϕ_c. A regression algorithm can then be used to construct each pair of lines representing the range of feasible azimuth angles, elevation angles and focal lengths on the azimuth-time plane, elevation-time plane and focal length-time plane respectively. These projections serve as a simple representation of an otherwise complex 4D volume in azimuth angle, elevation angle, focal length and time. Illustrations are shown in Fig. 1(d) and (e), where 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. (d) is without the field of view constraint while (e) includes that constraint - plot (d) thus covers wider viewing angles, which become invalid only when the object is behind the image plane. The peaks in both plots coincide with the object's orientation, in which case the image conic is just a circle and has the lowest resolution.

3 Compositing TVIs Using Plane-Sweep

After constructing the TVIs for a given camera, we identify subsets of these TVIs that share common regions of camera settings during overlapping time intervals. If the durations of such time intervals are additionally at least as long as the required acquisition times of the tasks involved, a condition we refer to as the Composition Feasibility (CF) condition, the corresponding TVIs can be combined, so that multiple tasks can be satisfied simultaneously in a single scheduled capture. The resulting intervals are called Multiple Tasks Visibility Intervals (MTVIs). The construction of all feasible MTVIs for a given camera is a computationally expensive operation, because one would then have to determine all feasible MTVIs that satisfy numbers of tasks ranging from 2 to the total number of tasks at hand. We introduce an efficient plane-sweep algorithm for this purpose. We begin by defining the "slack" δ of a (M)TVI as:

$$\delta = [t_\delta^-, t_\delta^+] = [r, d - p], \tag{4}$$

where again r, d and p are as given in Eqn. 1 and 2. Intuitively, the slack is just a temporal interval within which a task can be started. We further define the lapse $|\delta|$ as $t_\delta^+ - t_\delta^-$.

We can now construct what we call an Ordered Timeline (OT) along which the sweeping is performed, using the following procedure: First, compute the set S_2 of all feasible MTVIs containing two tasks (which can be done simply using an $O(n^2)$ approach). If p_1 and p_2 are the respective processing times of the two tasks in a MTVI in S_2, indexed by i, then let its slack δ_i be $[t_{\delta_i}^-, t_{\delta_i}^+] = [r, d - \max(p_1, p_2)]$, where r and d are the earliest release time and deadline of the MTVI. A timeline can now be constructed as $\bigcup_i \{t_{\delta_i}^-, t_{\delta_i}^+\}$, which is then sorted to get the OT. An example is shown in Fig. 2(a), in which four different pairwise MTVIs are used to construct an OT. Due to the "splitting" effect of the procedure, the utilization of slacks, instead of $[r, d]$ in Eqn. 1, when forming the OT ensures that any resulting MTVI after applying our plane-sweep algorithm does not have negative lapse in its slack - i.e., there is no valid time to start tasks in the MTVI. It is also apparent from Fig. 2(a) that the new set of slacks along the OT is made up of the following types. An SS interval is formed from the start times of two consecutive slacks, an SE interval from the start and end times of two consecutive slacks, an ES interval from the end and start times of two consecutive slacks, and finally an EE interval is formed from the end times of two consecutive slacks.

In addition, at any time step along the OT, it is possible that one or more previously encountered MTVIs remain "active". So, for example in Fig. 2(a), the MTVI with slack $[t_{\delta_1}^-, t_{\delta_1}^+]$ remains active until the end of the OT. In the following plane-sweep algorithm, we will maintain a set S_{active} of such active MTVIs. For ease of illustration, we will also refer to the two MTVIs that made up each interval along the OT as m_{first} and m_{second}, respectively, in order of their appearance. The plane-sweep algorithm can then proceed by advancing across the OT in the following manner:

1. If an SS slack is encountered, initialize a new MTVI m_{new} with tasks in m_{first} and slack equals to the SS slack. MTVIs in S_{active} that satisfy the CF condition with m_{new} are combined; otherwise they are added as separate MTVIs, again with slack equal to the SS slack.
2. If an SE slack is encountered, (1) add tasks from both m_{first} and m_{second} to m_{new} and assign to it the SE slack, if they meet the CF condition, or (2) keep m_{first} and m_{second} as two MTVIs $m_{new,1}$ and $m_{new,2}$, but assigning to both the SE slack. Process S_{active} in the same manner as step 1, but on either m_{new} (case 1) or both $m_{new,1}$ and $m_{new,2}$ (case 2).
3. If an ES slack is encountered, add MTVIs in S_{active}, assigning to them the ES slack.
4. If an EE slack is encountered, initialize m_{new} with tasks in m_{second} and slack equals to the EE slack. Process S_{active} in the same manner as step 1 on m_{new}.

After performing the plane-sweep algorithm for a given camera, each interval along the corresponding OT now consists of a set cover (not necessarily a minimum set cover) of tasks that could be satisfied in that interval. These set covers (comprised of the resulting MTVIs from the plane-sweep algorithm) are eventually used in the final scheduling algorithm, which will be described in the following section.

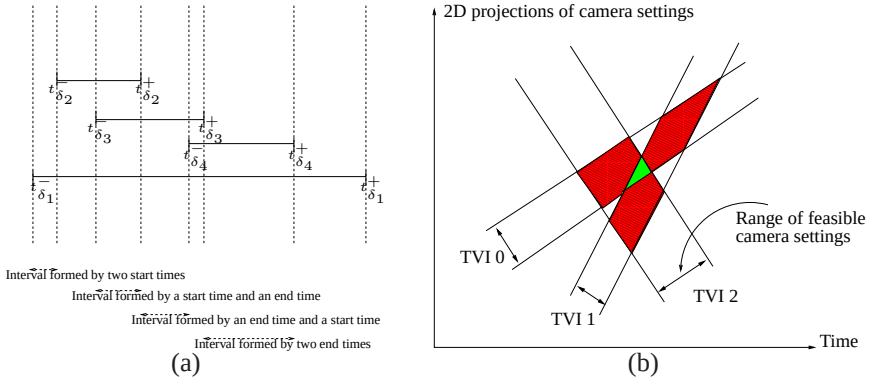

Fig. 2. (a) Forming OT from different pairwise MTVIs. There are four MTVIs given in this example with slacks: $[t^-_{\delta_1}, t^+_{\delta_1}]$, $[t^-_{\delta_2}, t^+_{\delta_2}]$, $[t^-_{\delta_3}, t^+_{\delta_3}]$ and $[t^-_{\delta_4}, t^+_{\delta_4}]$. The resulting OT consists of the slacks: $[t^-_{\delta_1}, t^-_{\delta_2}]$, $[t^-_{\delta_2}, t^-_{\delta_3}]$, $[t^-_{\delta_3}, t^+_{\delta_2}]$, $[t^+_{\delta_2}, t^-_{\delta_4}]$, $[t^-_{\delta_4}, t^+_{\delta_3}]$, $[t^+_{\delta_3}, t^+_{\delta_4}]$ and $[t^+_{\delta_4}, t^+_{\delta_1}]$. (b) The green region of feasible sensor settings is a valid MTVI with three tasks, iff its slack lies within the green region, and the CF condition is satisfied. Here, the slack's start time equals that of the slack of the pairwise MTVI formed by TVI 0 and 1 and the end time is the minimum among those of the slacks of the pairwise MTVI formed by TVI 0 and 1, 0 and 2, and 1 and 2.

Finally, we verify the correctness of the plane-sweep algorithm: that the OT sufficiently delineate all feasible MTVIs' slacks (i.e., that any (M)TVI's slack is delimited by a pair of time instances on the OT). This can be easily proved by the following theorem:

Theorem 1. *Let the slacks start and end times of any MTVI be* t^-_δ *and* t^+_δ*. Then* t^-_δ, $t^+_\delta \in$ *OT.*

Proof. Let the number of tasks in the MTVI be $n(\geq 2)$. We can form up to $\binom{n}{2}$ feasible pairwise MTVIs. The corresponding slacks $[t^-_{\delta_i}, t^+_{\delta_i}] \in$ OT, for $i = 1... \binom{n}{2}$. Then, by considering the forms of the common sensor setting volume encountered in this application, an example of which is shown in Fig. 2(b), $t^-_\delta = max(t^-_{\delta_i})$ and $t^+_\delta = min(t^+_{\delta_i})$. □

The combination of TVIs into MTVIs is illustrated in Fig. 1(f). The temporal behavior of an object for a task is shown by keeping the elevation angle as zero. The corresponding 2D view is shown in (g) in the azimuth-time plane. In (h) and (i), the plot for the same object is intersected with that of another task. The resulting volumetric intersection is delimited by a temporal interval, and a region of common camera settings that can be used to satisfy both tasks simultaneously. Again, we utilize a simple representation of such volumes to find these common camera settings by projecting them onto the respective 2D planes (i.e., azimuth-time, elevation-time and focal length-time), where the intersections can be computed efficiently.

4 Sensor Scheduling

Given the set of atomic TVIs and MTVIs that have been constructed for a given camera, the scheduling problem is then to decide: (1) which (M)TVIs should be executed, and (2) given the set of (M)TVIs chosen for execution, what the order of execution should be, so as to maximize the coverage of tasks. In general, scheduling problems such as this are NP-hard, making the search for an optimal solution computationally infeasible. In the following sections, we begin by studying the scheduling problem when only a single camera is used before extending to multiple cameras, using the assumption that any resulting schedule is non-preemptive.

4.1 Single-camera Scheduling

To find the optimal schedule for a single camera, we introduce the following theorems that make the single-camera scheduling problem tractable:

Theorem 2. *Consider a camera. Let* $\delta_{max} = \arg\max_{\delta_i}(|\delta_i|)$ *and* p_{min} *be the smallest processing time among all (M)TVIs. Then, if* $|\delta_{max}| < p_{min}$*, any feasible schedule for the camera is ordered by the slacks' start times.*

Proof. Consider that the slack $\delta_1 = [t^-_{\delta_1}, t^+_{\delta_1}]$ precedes $\delta_2 = [t^-_{\delta_2}, t^+_{\delta_2}]$ in a schedule and $t^-_{\delta_1} > t^-_{\delta_2}$. Let the processing time corresponding to δ_1 be p_1. Then $t^-_{\delta_1} + p_1 > t^-_{\delta_2} + p_1$. We know that if $t^-_{\delta_1} + p_1 > t^+_{\delta_2}$, then the schedule is infeasible. This happens if $t^+_{\delta_2} \leq t^-_{\delta_2} + p_1$ - i.e., $t^+_{\delta_2} - t^-_{\delta_2} \leq p_1$. Given that $|\delta_{max}| < p_{min}$, $t^+_{\delta_2} - t^-_{\delta_2} \leq p_1$ is true. □

Theorem 2 implies that if $|\delta_{max}| < p_{min}$ we can just look for feasible schedules that are ordered by the slacks' start times. In practise, such a condition is hardly restrictive at all, since it is easy to adjust the processing time of tasks to meet the condition. Based on this assumption, we can construct a Directed Acyclic Graph (DAG), where each (M)TVI is a node with an incoming edge from a common source node and outgoing edge to a common sink node, with the weights of the outgoing edges initialized to zero. An outgoing edge from one (M)TVI node to another exists iff the slack's start time of the first node is earlier than that of the second (Theorem 2), which can however be removed if it makes the schedule infeasible. Consider the following theorem and corollary:

Theorem 3. *A feasible schedule contains a sequence of* n *(M)TVIs each with slack* $\delta_i = [t^-_{\delta_i}, t^+_{\delta_i}]$*, where* $i = 1...n$ *represents the order of execution, such that* $t^+_{\delta_n} - t^-_{\delta_1} \geq (\sum_{i=1...n-1} p_i) - (\sum_{i=1...n-1} |\delta_i|)$*,* p_i *being the processing time of the* i^{th} *(M)TVI in the schedule.*

Proof. For the schedule to be feasible the following must be true: $t^-_{\delta_1} + p_1 \leq t^+_{\delta_2}$, $t^-_{\delta_2} + p_2 \leq t^+_{\delta_3}$, ... , $t^-_{\delta_{n-1}} + p_{n-1} \leq t^+_{\delta_n}$. Summing them up gives $t^-_{\delta_1} + t^-_{\delta_2} + ... + t^-_{\delta_{n-1}} + \sum_{i=1...n-1} p_i \leq t^+_{\delta_2} + t^+_{\delta_3} + ... + t^+_{\delta_n}$, which can then be simplified as $t^+_{\delta_n} - t^-_{\delta_1} \geq (\sum_{i=1...n-1} p_i) - (\sum_{i=1...n-1} |\delta_i|)$. It is also clear that the condition, $t^-_{\delta_1} + p_1 \leq t^+_{\delta_2}$, $t^-_{\delta_2} + p_2 \leq t^+_{\delta_3}$, ... , $t^-_{\delta_{n-1}} + p_{n-1} \leq t^+_{\delta_n}$, is sufficient for checking the feasibility of a schedule. □

Corollary 1. *Define a new operator* $\preceq$*, such that if* $\delta_1 (= [t^-_{\delta_1}, t^+_{\delta_1}]) \preceq \delta_2 (= [t^-_{\delta_2}, t^+_{\delta_2}])$*, then* $t^-_{\delta_1} + p_1 \leq t^+_{\delta_2}$*. Consider a schedule of (M)TVIs with slacks* $\delta_{i...n}$*. The condition:* $\delta_1 \preceq \delta_2$*,* $\delta_2 \preceq \delta_3$*, ...,* $\delta_{n-1} \preceq \delta_n$*, is necessary for the schedule to be feasible. Conversely, if a schedule is feasible, then* $\delta_1 \preceq \delta_2$*,* $\delta_2 \preceq \delta_3$*, ...,* $\delta_{n-1} \preceq \delta_n$*. Proof is omitted since it follows easily from Theorem 3.*

As a result, due to Corollary 1, an edge between two (M)TVI nodes can then be removed if they violate the $\preceq$ relationship since they can never be part of a feasible schedule.

Using such a DAG, a Dynamic Programming (DP) algorithm can be used to solve the single-camera scheduling problem. The algorithm assigns weights to edges between nodes in the DAG on the fly during a backtracking stage, illustrated by the following example with the aid of Fig. 3(a) and 4. Consider the following set of (M)TVIs that have been constructed for a given camera, represented by the tasks ($T_{1...6}$) they can satisfy and sorted in order of their slacks' start times:$\{node_1 = \{T_1, T_2\}, node_2 = \{T_2, T_3\}, node_3 = \{T_3, T_4\}, node_4 = \{T_5, T_6\}\}$, where the set of nodes in the DAG in Fig. 3(a) is given as $node_{i=1...4}$. Based on the constructed DAG, we can form a table for running DP as illustrated in Fig. 4. DP is run by first initializing paths of length 1 starting from each of the (M)TVI nodes to the sink, all with "merits" 0. At each subsequent path length, the next node $node_{next}$ chosen for a given node $node_{curr}$ in the current iteration is:

$$node_{next} = \underset{n \in S_{curr2next}}{\arg\max} |S_n \bigcup Tasks(node_{curr})|, \tag{5}$$

where $S_{curr2next}$ is the set of nodes that have valid paths starting from them in the previous iteration and for which $node_{curr}$ has an outgoing edge to. S_n is defined as the set of tasks covered by the path (in the previous iteration) starting from n, and $Tasks()$ gives the set of tasks covered by the (M)TVI associated with $node_{curr}$. So, for example, from $node_1$, paths of length 2 exist by moving on to either one of $node_{2...4}$, with the move to $node_2$, $node_3$ and $node_4$ covering $\{T_1, T_2, T_3\}$ (merits=3), $\{T_1, T_2, T_3, T_4\}$ (merits=4) and $\{T_1, T_2, T_5, T_6\}$ (merits=4) respectively. We choose the path of length 2 from $node_1$ to $node_3$. Iterations are terminated when there is only one path left that starts at the source node or a path starting at the source node covers all the tasks. In our example, the optimal path becomes $node_1 \rightarrow node_3 \rightarrow node_4$, terminated at paths of length 4 from the sink when all the tasks are covered. It can also be observed from the example, that the layout of the DP table is always "triangular" due to the assumption in Theorem 2.

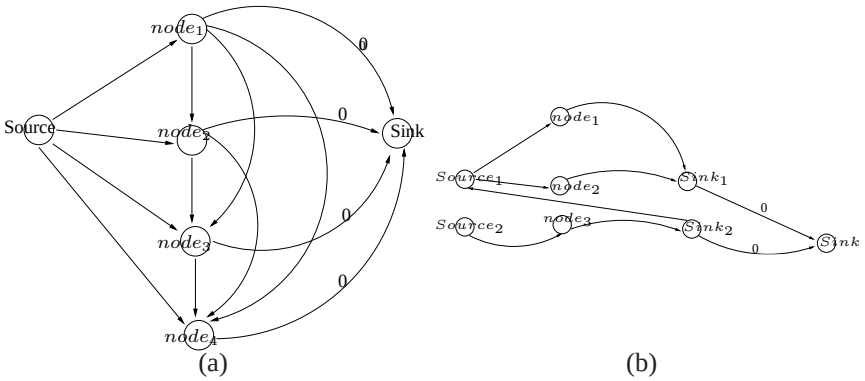

Fig. 3. (a) Single-camera scheduling: DAG formed from the set $\{node_1 = \{T_1, T_2\}, node_2 = \{T_2, T_3\}, node_3 = \{T_3, T_4\}, node_4 = \{T_5, T_6\}\}$. The weights between (M)TVI nodes are determined on the fly during DP. Assume that, in this example, the $\preceq$ relationship is satisfied for the edges between the (M)TVI nodes. (b) Multi-camera scheduling: DAG formed from the set $\{node_1 = \{T_1, T_2, T_3\}, node_2 = \{T_3, T_4\}\}$ for the first camera, and the set $\{node_3 = \{T_1, T_2, T_3\}\}$ for the second camera.

Distance from sink	Nodes Source	$node_1$	$node_2$	$node_3$	$node_4$
1	X	merit=0	merit=0	merit=0	merit=0
2	merit=2 $\rightarrow node_1$ $\{T_1, T_2\}$	merit=4 $\rightarrow node_3$ $\{T_1, T_2, T_3, T_4\}$	merit=4 $\rightarrow node_4$ $\{T_2, T_3, T_5, T_6\}$	merit=4 $\rightarrow node_4$ $\{T_3, T_4, T_5, T_6\}$	X
3	merit=4 $\rightarrow node_1$ $\{T_1, T_2, T_3, T_4\}$	merit=6 $\rightarrow node_3$ $\{T_1, T_2, T_3, T_4, T_5, T_6\}$	merit=5 $\rightarrow node_3$ $\{T_2, T_3, T_4, T_5, T_6\}$	X	X
4	merit=6 $\rightarrow node_1$ $\{T_1, T_2, T_3, T_4, T_5, T_6\}$	merit=6 $\rightarrow node_2$ $\{T_1, T_2, T_3, T_4, T_5, T_6\}$	X	X	X

Fig. 4. Dynamic programming table for DAG in Fig. 3(a). An "X" indicates that no path of the specific length starts at that node.

4.2 Multi-camera Scheduling

While single-camera scheduling using DP is optimal and has polynomial running time, the multi-camera scheduling problem is unfortunately NP-hard. Consequently, computationally feasible solutions can only be obtained with approximation algorithms. We compare a simple greedy algorithm with a branch and bound-like algorithm.

Greedy Algorithm The greedy algorithm iteratively picks the (M)TVI that covers the maximum number of uncovered tasks, subject to schedule feasibility as given by Theorem 3. Under such a greedy scheme, the following is true:

Theorem 4. *Given k cameras, the approximation factor for multi-camera scheduling using the greedy algorithm is $2 + k\lambda\mu$, where the definitions of λ and μ are given in the proof.*

Proof. Let $G = \bigcup_{i=1\ldots k} G_i$, where G_i is the set of (M)TVIs scheduled on camera i by the greedy algorithm, and $OPT = \bigcup_{i=1\ldots k} OPT_i$, where OPT_i is the set of (M)TVIs assigned to camera i in the optimal schedule. We further define (1) $H_1 = \bigcup_{i=1\ldots k} H_{1,i}$, where $H_{1,i}$ is the set of (M)TVIs for camera i, that have been chosen by the optimal schedule but not the greedy algorithm and each of these (M)TVIs contains tasks that are not covered by the greedy algorithm in any of the cameras, (2) $H_2 = \bigcup_{i=1\ldots k} H_{=2,i}$, where $H_{2,i}$ is the set of (M)TVIs for camera i, that have been chosen by the optimal schedule but not the greedy algorithm and each of these (M)TVIs contains tasks that are also covered by the greedy algorithm, and finally (3) $OG = OPT \bigcap G$. Clearly, $OPT = H_1 \bigcup H_2 \bigcup OG$. Then, for $h_{j=1\ldots n_i} \in H_{1,i}$ where n_i is the number of (M)TVIs in $H_{1,i}$, $\exists g_{j=1\ldots n_i} \in G_i$ such that h_j and g_j cannot be scheduled together based on the requirement given in Theorem 3, else h_j should have been included by G. If $Tasks(h_j) \bigcap Tasks(g_j) = \emptyset$, then h_j contains only tasks that are not covered by G. In this case, $|h_j| \leq |g_j|$, else G would have chosen h_j instead of g_i. Note that the cardinality is defined as the number of unique tasks covered. In the same manner, even if $Tasks(h_j) \bigcap Tasks(g_j) \neq \emptyset$, h_j could have replaced g_j unless $|h_j| \leq |g_j|$. Consequently, $|H_{1,i}| = |h_1 \bigcup h_2 \bigcup \ldots \bigcup h_{n_i}| \leq |h_1| + |h_2| + \ldots + |h_{n_i}| \leq |g_1| + |g_2| + \ldots + |g_{n_i}|$. Let $\beta_j = \frac{|g_j|}{|G_i|}$ and $\lambda_i = \max(\beta_j * n_i)$. This gives $|H_{1,i}| \leq \beta_1|G_i| + \ldots + \beta_{n_i}|G_i| \leq \lambda_i|G_i|$. Similarly, we know $|H_1| \leq \lambda_1|G_1| + \ldots + \lambda_k|G_k| \leq \lambda(|G_1| + \ldots + |G_k|)$, where $\lambda = \max(\lambda_i)$. Introducing a new term, $\gamma_i = \frac{|G_i|}{|G|}$ and letting $\mu = \max(\gamma_i)$, we get $|H_1| \leq k\lambda\mu|G|$. Since $|H_2| \leq |G|$ and $|OG| \leq |G|$, $|OPT| \leq (2 + k\lambda\mu)|G|$. □

Branch and Bound Algorithm The branch and bound approach runs DP in a similar manner as single-camera scheduling but on a DAG that consists of multiple source-sink pairs (one pair per camera), with the node of one camera's sink node linked to another

camera's source node. An example is shown in Fig. 3(b). Then, for a source node s, we define its "upper bounding set" S_s as:

$$S_s = \bigcup_{c \in S_{link}} S_c, \tag{6}$$

where S_{link} is the set of cameras for which paths starting from the corresponding sink nodes to s exist in the DAG, and S_c is the set of all tasks that are covered by some (M)TVIs belonging to camera c. Intuitively, such an approach aims to overcome the "shortsightedness" of the greedy algorithm by "looking forward" in addition to backtracking and using the tasks that can be covered by other cameras to influence the (M)TVI nodes chosen for a particular camera. Admittedly, better performance is possibly achievable if "better" upper bounding sets are used, as opposed to blindly using all the tasks that other cameras can cover without taking scheduling feasibility into consideration.

The algorithm can be illustrated with the example shown in Fig. 3(b), which shows two cameras, c_1 and c_2, and the following sets of (M)TVIs that have been constructed for them, again ordered by the slacks' start times and shown here by the tasks ($T_{1...4}$) they can satisfy. For c_1, the set is $\{node_1 = \{T_1, T_2, T_3\}, node_2 = \{T_3, T_4\}\}$ and for c_2, $\{node_3 = \{T_1, T_2, T_3\}\}$. The DAG that is constructed has two source-sink pairs, one for each camera - $(Source_1, Sink_1)$ belongs to c_1 and $(Source_2, Sink_2)$ to c_2. The camera sinks are connected to a final sink node as shown, with the weights of the edges initialized to zero. Weights between nodes in the constructed DAG are similarly determined on the fly like in the single-camera scheduling. Directed edges from $Sink_2$ to $Source_1$ connects c_1 to c_2. As illustrated in Fig. 5(a), the DP algorithm is run in almost the same manner as single-camera scheduling, except that at paths of length 3 from the final sink node, the link from $Source_1$ to $node_2$, is chosen because the upper bounding set indicates that choosing the link potentially covers a larger number of tasks (i.e., the upper bounding set of $Source_1$, $\{T_1, T_2, T_3\}$ combines with the tasks covered by $node_2$ to form $\{T_1, T_2, T_3, T_4\}$). This turns out to be a better choice as compared to the results shown in Fig. 5(b), where no such upper bounding set was used.

The branch and bound algorithm can be viewed as applying the single-camera DP algorithm, camera by camera in the order given in the corresponding DAG, with the schedule of one camera depending on its upper bounding set. In this sense, we can derive a potentially better approximation factor than the greedy algorithm as follow:

Theorem 5. *For k cameras, the approximation factor of multi-camera scheduling using the branch and bound algorithm is $\frac{(1+k\mu(1+u))^k}{(1+k\mu(1+u))^k-(k\mu(1+u))^k}$. μ and u are defined as follow. Let $G^* = \bigcup_{i=1...k} G_i^*$, where G_i^* is the set of (M)TVIs assigned to camera i by the branch and bound algorithm. Then, $\mu = \max(\frac{|G_i^*|}{|G^*|})$ and $u = \max(u_i)$, where u_i is the ratio of the cardinality of the upper bounding set of camera i to $|G_i^*|$.*

Proof. Let α be the approximation factor of the branch and bound algorithm. Then, assuming that schedules for $G_1^*, ..., G_{i-1}^*$ have been determined, $|G_i^*| \geq \frac{1}{\alpha}(|OPT| - \sum_{j=1}^{i-1} |G_j^*|)$. Adding $\sum_{j=1}^{i-1} |G_j^*|$ to both sides gives:

$$\sum_{j=1}^{i} |G_j^*| \geq \frac{OPT}{\alpha} + \frac{\alpha - 1}{\alpha} \sum_{j=1}^{i-1} |G_j^*|).$$

A proof by induction shows, after some manipulation:

$$\frac{\alpha^k}{\alpha^k - (\alpha - 1)^k} \sum_{j=1}^{k} |G_j^*| \geq |OPT|.$$

Distance from sink	Nodes $Source_1$	$Source_2$	$node_1$	$node_2$	$node_3$	$Sink_1$	$Sink_2$
1	X	X	X	X	X	$\rightarrow Sink$	$\rightarrow Sink$
2	X	X	$\rightarrow Sink_1$ $\{T_1, T_2, T_3\}$	$\rightarrow Sink_1$ $\{T_3, T_4\}$	$\rightarrow Sink_2$ $\{T_1, T_2, T_3\}$	X	X
3	$\rightarrow node_2$ $\{T_3, T_4\}$	$\rightarrow node_3$ $\{T_1, T_2, T_3\}$	X	X	X	X	X
4	X	X	X	X	X	X	$\rightarrow Source_1$ $\{T_3, T_4\}$
5	X	X	X	X	$\rightarrow Sink_2$ $\{T_1, T_2, T_3, T_4\}$	X	X
6	X	$\rightarrow node_3$ $\{T_1, T_2, T_3, T_4\}$	X	X	X	X	X

(a)

Distance from sink	Nodes $Source_1$	$Source_2$	$node_1$	$node_2$	$node_3$	$Sink_1$	$Sink_2$
1	X	X	X	X	X	$\rightarrow Sink$	$\rightarrow Sink$
2	X	X	$\rightarrow Sink_1$ $\{T_1, T_2, T_3\}$	$\rightarrow Sink_1$ $\{T_3, T_4\}$	$\rightarrow Sink_2$ $\{T_1, T_2, T_3\}$	X	X
3	$\rightarrow node_1$ $\{T_1, T_2, T_3\}$	$\rightarrow node_3$ $\{T_1, T_2, T_3\}$	X	X	X	X	X
4	X	X	X	X	X	X	$\rightarrow Source_1$ $\{T_1, T_2, T_3\}$
5	X	X	X	X	$\rightarrow Sink_2$ $\{T_1, T_2, T_3\}$	X	X
6	X	$\rightarrow node_3$ $\{T_1, T_2, T_3\}$	X	X	X	X	X

(b)

Fig. 5. Dynamic programming table for DAG in Fig. 3(b). Using the upper bounding set yields a solution that is optimal in (a) as opposed to (b). In (a), at distance of length 3 from the sink, the link chosen for $Source_1$ is to $node_2$ instead of $node_1$ since the union of the upper bounding set for $Source_1$ ($\{T_1, T_2, T_3\}$ as given by Eqn. 6) with $node_2$ potentially has a larger task coverage.

Let $H = \bigcup_{i=1...k} H_i$, H_i being the set of (M)TVIs chosen by the optimal schedule on camera i but not the branch and bound algorithm. The condition $|H_i| \leq |G_i^*| + u_i|G_i^*|$ is true; otherwise, H_i would have been added to G^* instead. Consequently, $|H| \leq (|G_1^*| + ... + |G_k^*|) + (u_1|G_1^*| + ... + u_k|G_k^*|) \leq k\mu|G^*| + ku\mu|G^*| \leq k\mu(1+u)|G^*|$. Since $OPT = OG \bigcup H$ (Theorem 4), we get $|OPT| \leq 1 + k\mu(1+u)|G^*|$. Thus, $\alpha = 1 + k\mu(1+u)$. $\square$

Both the single-camera and branch and bound multi-camera algorithm have a computational complexity of $O(N^3)$, N being the average number of (M)TVIs constructed for a given camera and used in the resulting DAG. The number of iterations (i.e., number of rows in our DP tables), depends on the number of cameras multiplied by N. This, together with a asymptotic cost of $O(N^2)$ checking possible backtracking paths at each iteration give $O(N^3)$. Clearly, this means that one advantage of employing the greedy multi-camera algorithm is its faster computational speed of $O(N^2)$.

5 Results

In Theorem 4 and 5, we have shown that both the performance of the branch and bound and greedy algorithm are sensitive to several factors. Empirically, we have conducted extensive simulations to compare their performance - 600 simulations were conducted each time based on 5, 10, 15 and 20 objects for 1 (i.e., single-camera DP), 2 and 3 cameras respectively, and the percentage of these simulations whereby the branch and bound algorithm outperforms the greedy one - i.e., schedules more tasks - (and vice versa) is recorded. The results are shown in Fig. 6(a), (b) and (c) for 1, 2 and 3 cameras respectively. While the branch and bound algorithm outperforms the greedy one in these simulations, the mean number of objects captured by the branch and bound algorithm is very close to that of the greedy algorithm, as shown in (d).

We have also conducted real-time experiments with a prototype multi-camera system, consisting of four PTZ cameras synchronized by a Matrox four-channel card. One camera is kept static, so that it can be used for background subtraction and tracking.

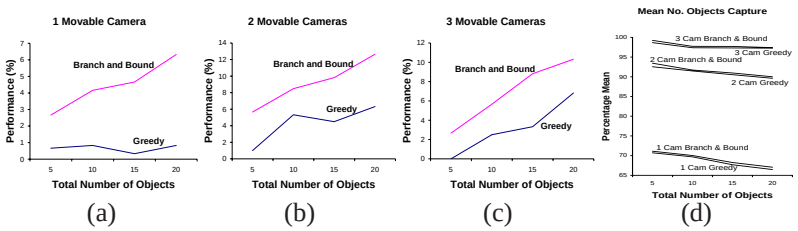

Fig. 6. (a)-(c) The simulations conducted here do not enforce the condition ($|\delta_{max}| < p_{min}$) in Theorem 2. The plots record the percentage of time when the branch and bound algorithm outperforms the greedy one (labeled as "Branch and Bound") and vice versa (labeled as "Greedy"). Ties are not shown. While the branch and bound algorithm clearly outperforms the greedy one in these plots, the mean number of objects, expressed as percentage of the total number of objects, captured by the branch and bound algorithm is very close to that of the greedy algorithm, as shown in (d).

The system recovers an approximate 3D size estimate of each detected object from ground plane and camera calibration, and uses them to construct (M)TVIs, which are then scheduled for capture. Due to the lack of exact future object location information, the system also updates the schedules as new observations are made by combining the predicted motion trajectories with the measured ones probabilistically. Fig. 7 first demonstrates the working of the system, from the acquisition of the motion models in (a) to the final video captures in (b) and (c), where the system's capabilities in dealing with occlusions and merging foreground blobs are also demonstrated. In Fig. 8(a) and (b), the effect of changing resolution requirement on the (M)TVIs constructed, using four remote-controllable 12x14 inches robots, are shown. As we increase the resolution requirement, while only two cameras are needed to capture the four robots in (a), three were needed in (b). The system's capabilities in scheduling specific tasks are further shown in (c) and (d), where tasks involving captures of faces and close interactions between objects are demonstrated. Comprehensive results are provided and explained in the accompanying videos.

6 Conclusions

We have described a multi-camera system that constructs (M)TVIs, during which targeted objects in acquired videos are unobstructed, in the field of view, and meet task-specific resolution requirements. Following the construction of these (M)TVIs, a collection of cameras are to be scheduled for video collection so as to maximize the coverage of tasks. We have presented several computationally feasible scheduling algorithms, both for single and multiple cameras. 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. 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.
3. I. Stamos and P. Allen, "Interactive sensor planning," in *Computer Vision and Pattern Recognition Conference*, Jun 1998, pp. 489–494.
4. Steven Abrams, Peter K. Allen, and Konstantinos A. Tarabanis, "Dynamic sensor planning.," in *ICRA (2)*, 1993, pp. 605–610.
5. 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.

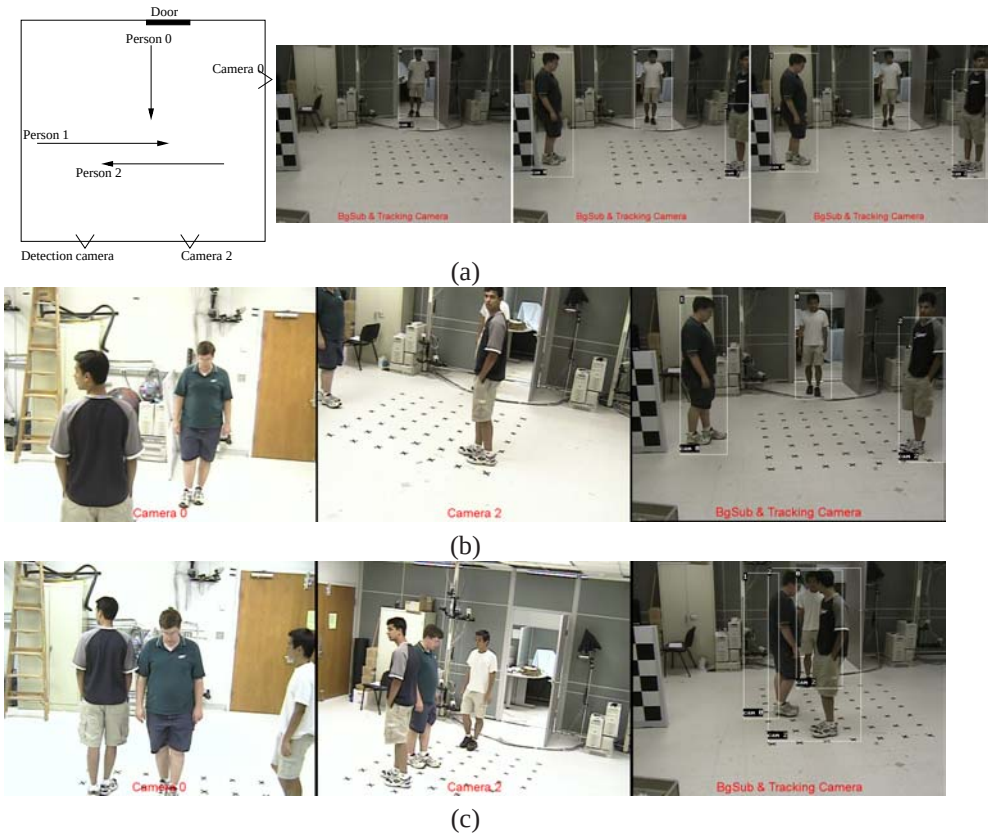

Fig. 7. Two movable cameras and one detection camera were used for the experiments in (a)-(c). (a) Left to right: A plan view of the predicted motion models, and sample frames used for constructing the motion model of each object. (b) Person 0 was occluded by the door from camera 0's view and by person 2 from camera 2's view. Thus, only person 1 and 2 were assigned to camera 0 (leftmost image) and 2 (middle image) respectively. The rightmost image is the detection camera. (c) Person 0 is finally visible and assigned to camera 2 together with person 2. The merging of foreground blobs was prevented by utilizing the predicted positions of the people. Note that the camera label in each bounding box indicates the camera that was assigned to capture the person.

6. 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.
7. Anurag Mittal and Larry S. Davis, "Visibility analysis and sensor planning in dynamic environments," in *European Conference on Computer Vision*, May 2004.
8. K.N. Kutulakos and C. R. Dyer, "Global surface reconstruction by purposive control of observer motion," in *IEEE Conference on Computer Vision and Pattern Recognition, Seattle, Washington, USA*, June 1994.
9. K.N. Kutulakos and C. R. Dyer, "Occluding contour detection using affine invariants and purposive viewpoint control," in *IEEE Conference on Computer Vision and Pattern Recognition, Seattle, Washington, USA*, June 1994.
10. K.N. Kutulakos and C. R. Dyer, "Recovering shape by purposive viewpoint adjustment," *International Journal of Computer Vision*, vol. 12, no. 2, pp. 113–136, 1994.
11. R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge University Press, ISBN: 0521623049, 2000.

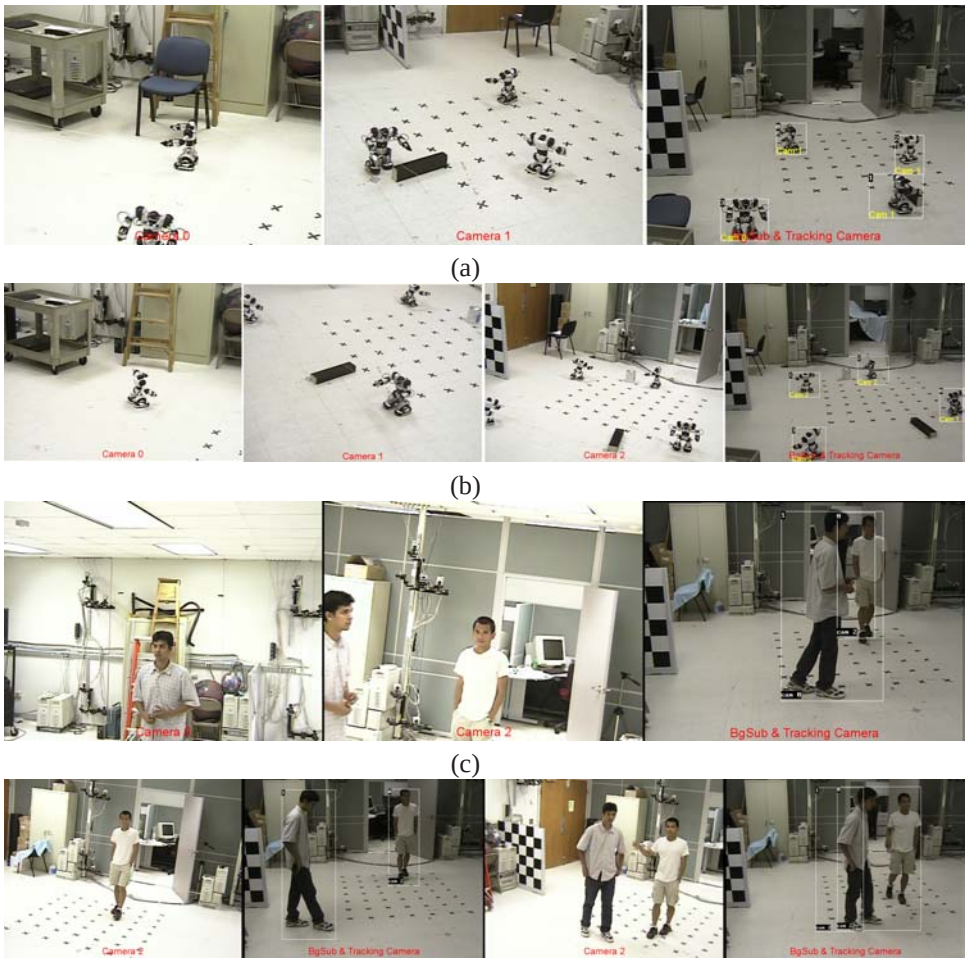

Fig. 8. (a) Two cameras are needed (one TVI for camera 0 shown in the leftmost image, and one MTVI with three tasks for camera 1, shown in the second image from the left) to capture the four robots. The rightmost image is the detection camera. (b) The resolution requirement was increased, and three cameras are now needed - one TVI for camera 0 and 1 and one MTVI with two tasks for camera 2, shown sequentially from left to right. In (c), the motion models of two persons in the scene were used to determine TVIs 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 (d), one movable camera was used and two persons are walking in the scene. The leftmost image shows the camera being assigned to one of the persons in the scene. However, the motion models of the two persons were used by the system to predict that there is a high likelihood of the two persons interacting - a task which was scheduled and captured accordingly, shown in the third image from the left.