Coordination of multiple cameras to track multiple people

Nam T. Nguyen Svetha Venkatesh Geoff West Hung H. Bui
School of Computing, Curtin University of Technology
GPO Box U1987 Perth, 6845 Western Australia
{nguyentn, svetha, geoff, buihh}@cs.curtin.edu.au

Abstract

In this paper, we present a distributed surveillance system that uses multiple cheap static cameras to track multiple people in indoor environments. The system has a set of Camera Processing Modules and a Central Module to coordinate the tracking tasks among the cameras. Since each object in the scene can be tracked by a number of cameras, the problem is how to choose the most appropriate camera for each object. We propose a novel algorithm to allocate objects to cameras using the object-to-camera distance while taking into account occlusion. The algorithm attempts to assign objects in the overlapping fields of view to the nearest camera which can see the object without occlusion. Experimental results show that the system can coordinate cameras to track people properly and can deal well with occlusion.

1 Introduction

Rapid advances in technology have made cheap sensors, and especially cameras, available. This has changed the goals of surveillance from building surveillance systems using only a single, powerful camera to building surveillance systems deploying multitudes of cheap cameras. In multiple camera tracking, we have to solve some problems in single camera tracking such as object segmentation, object occlusion, and so on. Several robust systems that use a single camera to track multiple objects are presented in [13, 4, 6]. We also have to deal with the new issues that arise when there are multiple cameras in the system. These include how to identify an object when it moves between the fields of view (FOVs) of the cameras and how to coordinate the cameras to track objects in the overlapping FOVs. Huang and Russell [5] provide a threshold-based approximation algorithm to identify objects observed by two spatially separated sensors. This work is scaled up to multiple sensors by Pasula *et al* [8]. Orwell *et al* present a method to recognize an object reappearing in the FOV of another camera by using the object color distributions [7]. Other work of note in this field includes matching human subjects between consecutive frames taken by multiple cameras [1] and coordinating both static cameras and mobile cameras to track people [12].

The current scenario we consider is many cheap cameras monitoring a large area, dividing up the area and objects among themselves. The problem is complex given the limited capabilities of camera on-board processing and the difficulties in coordinating multiple cameras. Assuming that a number of cameras can solve the same tasks and that many objects may be tracked at the same time, the issue is how to choose the most appropriate camera for each task. This can be regarded as a coordination problem in that the goal is to maximize the reliability of the tracking system given that there is limited processing capability available for each camera. The key question we seek to answer is: given overlapping FOVs of the cameras and multiple people moving around, can we find a good assignment algorithm to track the people reliably? The algorithm should deal well with situations where a person moves from the FOV of one camera to the FOV of another camera, or when a person is occluded by others.

This paper presents a distributed surveillance system. It operates in indoor environments and uses multiple cheap static cameras to track multiple people. Unlike the other tracking systems, in this system, a novel algorithm is introduced to allocate people to the cameras. With the help of this algorithm, the system can track people reliably in the condition of the limitation of camera on-board processing.

2 The distributed surveillance system

Our aim is to build a cheap surveillance system to track people moving in indoor environments. The system consists of multiple cameras, each connected to a computer on a local area network. We use static cameras in the system because of their low price. The cameras' FOVs overlap one another, therefore, an object may be viewed from several cameras at a time. Due to noise, objects need to be tracked

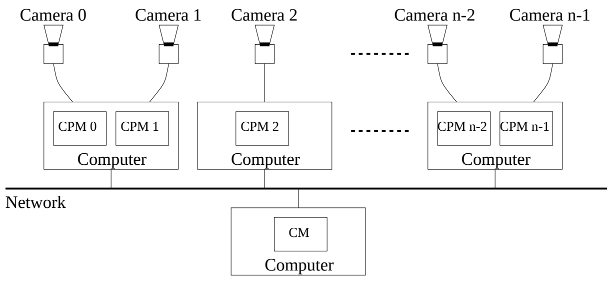


Figure 1. *A system of distributed network of cameras.*

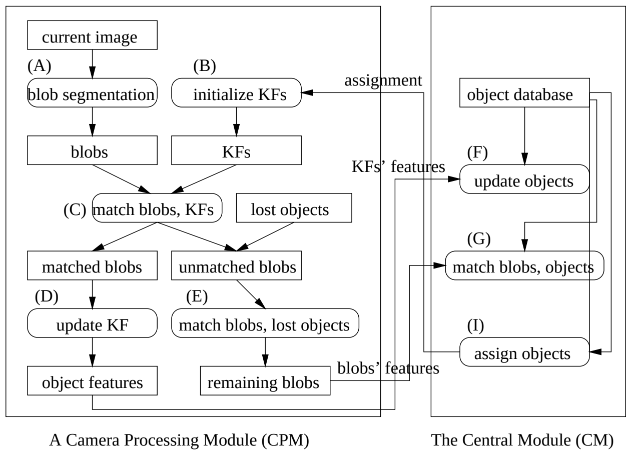


Figure 2. *Processing at a CPM and at the CM.*

by Kalman filters. There are two approaches to track an object using Kalman filters: (1) it is tracked by all cameras that can see the object, and (2) it is tracked by only one appropriate camera. The object is robustly tracked with the first approach, but more computational resources would be required for the system because it needs to maintain more Kalman filters. Moreover, the increase of the tracking reliability sometimes is not worth the additional computational costs. Because of the limitation of camera on-board processing, we choose the second approach for our system. With this approach, the system still can track objects reliably with the help of Kalman filters, but it requires an algorithm to assign each object to a suitable camera.

The distributed surveillance system is shown in Figure 1. With each camera, we have a corresponding Camera Processing Module (CPM) running on the computer that the camera is connected to. The system also has a Central Module (CM) that maintains a database of objects in the entire scene and coordinates the tracking tasks between the CPMs. Each CPM then needs to process the video stream to track the objects assigned to it by the CM.

Figure 2 shows the data processing and communication that takes place in a CPM and the CM. The left block shows the tasks performed at the CPM. The image captured by the camera is processed by the blob segmentation step to extract the motion blobs (the step A). The CPM also initializes a set of Kalman filters to track objects that are assigned to the camera (the step B). These Kalman filters are updated using the blobs' information. The right block shows the tasks performed at the CM. These include the maintenance of an object database and the assignment of the objects to the suitable cameras.

2.1 Blob segmentation

The first processing step at the CPM is to segment out motion blobs in the image sequences using background subtraction (step A in Figure 2). To do this, we need a background model. There are numerous methods to build background models [10, 3]. However, these methods are computationally expensive. Our system runs in indoor environments, so for simplicity and fast computation, the background image is first computed as the average of several images, and then updated using exponential forgetting.

The foreground pixels are found by comparing the current image with the background. We detect the boundaries of these foreground pixels by a chain-code algorithm [2]. The blobs of motion are computed based on these boundaries. Then, we calculate the size, position and average color of each blob. Due to the camera noise and the similarity of the object and background color, an object may be broken into several blobs. Therefore, two blobs which are near each other are merged into a single blob. We do this by merging recursively until all blobs are far from one another. After merging, blobs which are too small will be considered as noise and discarded.

2.2 Matching blobs to the Kalman filters

The blobs extracted from the blob segmentation step are the observations of the Kalman filters. However, we need to find the blob that corresponds to the observation of a specific Kalman filter. This task is performed by the blob-to-Kalman-filter matching step (step B in Figure 2).

The state of a Kalman filter K is represented as the vector $(K_x, K_y, K_w, K_h, K_r, K_g, K_b)$. (K_x, K_y) is the estimate of the position of the bottom edge of the object bounding box in the image. (K_w, K_h) is the estimate of the width and height of the object bounding box. $(K_r$, K_g,$K_b)$ is the estimate of the average red, green and blue color components of the object. All these variables are assumed to have Gaussian distributions.

We use the position, size and average color properties to find the match. The probabilistic distance between a blob B and a Kalman filter state K is defined as:

$$d_1(B,K) = P(B_x, B_y, B_r, B_g, B_b | K_x, K_y, K_r, K_g, K_b) \tag{1}$$

where (B_x, B_y) is the position of the bottom edge of the blob bounding box, (B_w, B_h) is the width and height of the blob bounding box, (B_r, B_g, B_b) is the average red, green and blue color components of the blob. The probabilistic distance $d_1(B, K)$ is computed from the Kalman filter state K. We also enforce two hard constraints to exclude invalid matches:

(1)$d_1(B, K) \leq pdist_{max}$

(2)$P(B_w, B_h|K_w, K_h) \leq psize_{max}$

where $pdist_{max}$ and $psize_{max}$ are the probabilistic distance threshold and the size threshold respectively. The problem is to find a set of matched (blob, Kalman filter state) pairs, so that the total match distance is minimal. This is an instance of the bipartite matching problem [11]. In our system, we use the non-iterative greedy algorithm to solve this problem [9]. It works as follows:

1. Choose a valid pair (B,K) for which the distance $d_1(K, B)$ is the minimum. Output the match (B,K).

2. Remove B from the list of blobs, remove K from the list of Kalman filter states. Return to step 1 if there are still valid (blob, Kalman filter state) pairs.

2.3 Matching blobs to the lost objects

Because of occlusion, a Kalman filter may have no observation. In that case, it continues to estimate the object properties for several frames before being removed. The corresponding object is considered to have the status of a "lost" object. We need to catch the lost objects again when the occlusion disappears. If the blob corresponding to a lost object exists, it will be in the set of unmatched blobs resulting from the matching step in section 2.2. Therefore, we need to match these blobs with the lost objects to find their correspondences. This matching is performed by step E in Figure 2. Because the position and size of an object in the image may change significantly after being lost, we can not match blobs to the lost objects using the same properties as in the previous section. Notice that, the color and the real-world size[1] of objects are nearly constant regardless of their locations in the scene. Therefore, we can define the probabilistic distance between a blob B and a lost object O as:

$$d_2(B, O) = P(B_r, B_g, B_b|O_r, O_g, O_b) \qquad (2)$$

where (O_r, O_g, O_b) is the average red, green and blue color components of the lost object O. The probabilistic distance $d_2(B, O)$ is computed from the last estimate of the Kalman filter corresponding to the lost object O. We also have two hard constraints to prohibit invalid matches:

(1) $d_2(B, O) \leq pcolor_{max}$

[1]The real-world size of an object is calculated from the image size through calibration.

	Camera 1	Camera 2	Camera 3
Object 1	tracks with KF, features (taken from the KF)	not sees clearly	sees clearly
Object 2	sees clearly	tracks with KF features (taken from the KF)	not sees clearly

Figure 3. *The status of the database at a particular time*

(2) $|B_{rw} - O_{rw}| + |B_{rh} - O_{rh}| \leq rwsize_{max}$

where (B_{rw}, B_{rh}) is the real-world width and height of blob B, (O_{rw}, O_{rh}) is the real-world width and height of object O before being lost, $pcolor_{max}$ and $rwsize_{max}$ are the color threshold and real-world size threshold respectively. We have the same bipartite matching problem as in the previous section. We also use the non-iterative greedy algorithm [9] to compute the (blob, lost object) matches.

2.4 The object database at the CM

At the CM, we maintain a database of all objects in the scene. It stores the object features, which are updated by taking the information from the corresponding Kalman filters (step F in Figure 2). Moreover, to support the object-to-camera assignment step (described in detail in Section 3), the database needs to contain information of which cameras seeing which objects "clearly". A camera sees an object clearly at a particular time if the system can find the observation (blob) of the object in the image captured by this camera at that time. Thus, a camera can not see an object clearly when it is occluded or it goes out of the FOV of the camera. The database knows whether a camera sees an object clearly or not by the matching step at the CM (step G in Figure 2). This step is described in detail in Section 2.5 below. Table 3 shows the status of the object database at a particular time.

2.5 Matching blobs to the objects at the CM

This step matches the blobs sent from a camera with the objects in the database (step G in Figure 2). The purpose of this matching is to let the system know that each camera can see which objects clearly. Notice that, with the objects tracked by the camera using Kalman filtering, we already know whether the camera can see them clearly or not after the matching steps C and E in each CPM. Therefore, we can exclude these objects and the corresponding blobs in the matching step at the CM. Because of this, we send to the CM only unmatched blobs resulted from step E (see Figure 2). Then, we match them with the objects that are not tracked by the camera. A matched (blob, object) pair found means that the camera can see this object clearly. The blobs are matched with the objects using real-world position, real-world size and color properties[2]. The distance between a blob B and an object O is defined as:

$$d_3(B,O) = ((B_{rx} - O_{rx})^2 + (B_{ry} - O_{ry})^2)^{1/2} \quad (3)$$

where (B_{rx}, B_{ry}) and (O_{rx}, O_{ry}) are the real-world locations of blob B and object O respectively. A match between blob B and object O must satisfy three constraints:

(1) $d_3(B,O) \leq \mathit{rwpos}_{max}$
(2) $|B_{rw} - O_{rw}| + |B_{rh} - O_{rh}| \leq \mathit{rwsize}_{max}$
(3) $|B_r - O_r| + |B_g - O_g| + |B_b - O_b| \leq \mathit{color}_{max}$

where (O_{rw}, O_{rh}) is the real-world width and height of object O, (O_r, O_g, O_b) is the average red, green and blue color components of object O, rwpos_{max}, rwsize_{max} and color_{max} are the thresholds. We have the same bipartite matching problem as in the previous sections and the same algorithm is used to find the blob-to-object correspondences.

3 Assigning objects to the cameras

In this part, we propose an algorithm to assign objects to the cameras using the object-to-camera distance while taking into account the object occlusion. We also introduce a function to measure the performance of the tracking system. This function is then used to evaluate our assignment algorithm.

3.1 Quality of service of a tracking system

In many cases, we need to compare the operation of a tracking system working with the different parameters and algorithms. To do that, we need a function representing quantitatively the performance of tracking systems. We term this function the Quality of Service (QOS) function. The QOS function is defined based on the sizes of the objects in the image and the object occlusion status as follows:

$$\begin{aligned} Q_{all} &= QOS(C_0, ..., C_{n-1}, O_0, ..., O_{m-1}) \\ &= \textstyle\sum_{i=0}^{n-1} \sum_{O \in \Omega_i} QOS(C_i, O) \end{aligned} \quad (4)$$

where Q_{all} is the QOS of the whole system, $QOS(C_i, O)$ is the QOS which camera C_i gives to object O, Ω_i is the set of objects currently tracked by camera C_i.

The QOS that a camera gives to an object equals the estimated size of the object obtained from the Kalman filter. Thus, the closer the object is to the camera the higher the QOS becomes. Should the object be occluded, the Kalman filtering estimate is then used for several frames, after that the object is considered as lost and the QOS drops to zero.

[2]Real-world position and size properties are calculated from image position and size properties through calibration at the CPMs.

3.2 Object assignment algorithm

For a reliable tracking system, we need an object assignment algorithm to obtain the highest QOS for the system. From equation 4, the QOS of the system is maximized when each object is tracked by the camera that can view the object with the largest size. Usually, in the case of no occlusion, the nearest camera will give the largest view of the object. If there is occlusion, an object needs to be tracked by one of the cameras which can see it clearly. Therefore, we have the assignment algorithm as follows: with each object, among the cameras that see this object clearly, choose the nearest camera to track the object. In the case that no camera sees the object clearly, the object continues to be assigned to the current camera.

The system gets the set of cameras which can see an object clearly from the database at the CM (see Section 2.4). The distance between a camera and an object can be calculated using their positions in real-world space. Therefore, the assignment algorithm described above can be easily implemented in the system.

To see how the algorithm works, assume that an object O is being tracked by a camera C. When object O moves out of the FOV of camera C, camera C can no longer see object O clearly. Then, the algorithm will switch the tracking of object O to one of the cameras that sees it clearly. Thus, the system can avoid losing the object in this case. In the case of object O being occluded, the system will attempt to switch the object to the camera that can view the object without occlusion. By this switching step, the system can take the advantage of multiple cameras to reduce the time the object is not observed by the system.

4 Experimental results

We implement our algorithm in a surveillance system to track multiple people moving in a room. The goal of the system is to find the trajectories of people in the scene.

In the system setup, we place three static cameras C_0, C_1 and C_2 in the corners of the room. The cameras are calibrated to get the correspondence between points on the floor (ground plane) and on the image plane. The 3-D positions of cameras C_0, C_1 and C_2 are (0,0,3), (0,6,3) and (7,1,3) respectively. The room has only one door. This door is monitored by camera C_1 at all times to detect the entering and leaving of objects to/from the room. Figure 4 shows the positions of the cameras and the door region viewed from above.

We let two people enter the room through the door, one after the other. They walk inside the room for about 40 seconds. Because we need to compare the performances of the system running with different algorithms, the scenario was recorded to video files from the cameras (frame rate of

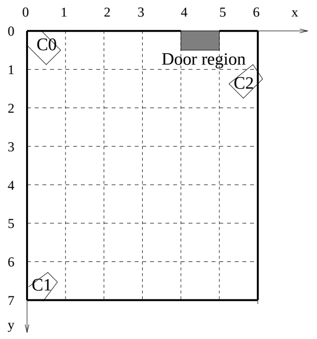


Figure 4. *The positions of cameras C_0, C_1, C_2 and the door region.*

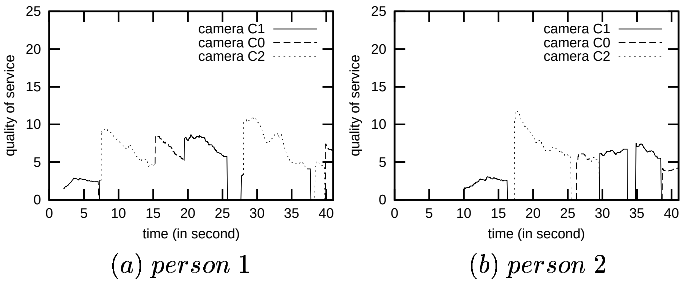


Figure 6. *The QOSs assigned to (a) person 1 and (b) person 2*

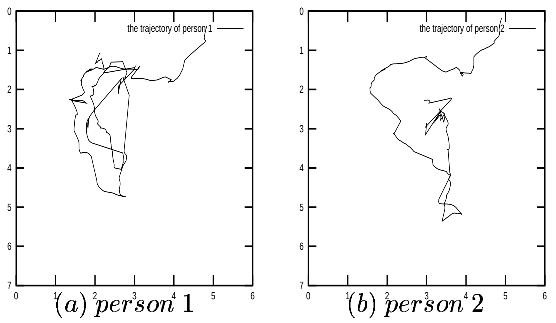


Figure 5. *The trajectories of (a) person 1 and (b) person 2.*

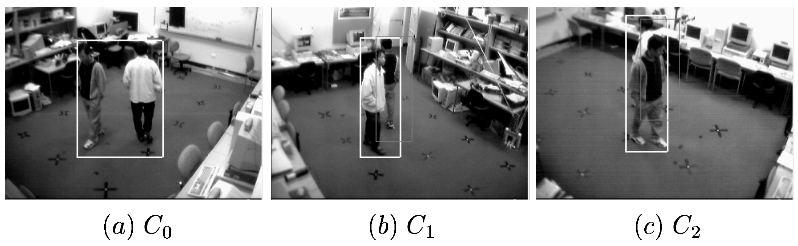


Figure 7. *Room viewed from cameras (a) C_0, (b) C_1 and (c) C_2 at 26.3 seconds.*

10 frames per second). These files are taken as the input for the system.

Figure 5 shows the trajectories of the two people returned by the system. The QOSs assigned to each person are shown in Figure 6. This figure shows us which person is being tracked by which camera and when a person is switched from one camera to another camera. For example, person 2 is switched from camera C_1 to C_2 at 17.3 seconds, from camera C_2 to C_0 at 26.3 seconds and so on. It also shows that the QOS of the system increases after each switching. It means that the switching helps the system track people more reliably.

To understand how the system performs the switching, we examine the scenario at 26.3 seconds (see Figure 7). At this time, person 1 (in dark clothes) is being tracked by C_1 and person 2 (in bright clothes) is being tracked by C_2 (gray boxes). Due to occlusion, person 1 and person 2 are lost. C_0 can not track the objects because they have been combined into one blob. The QOSs assigned to these people drop to zero at this time. Because both person 1 and person 2 are not seen clearly by all cameras, the system can not do any switching. After several frames, person 2 is seen clearly by camera C_0, so this person is switched from camera C_2 to camera C_0. After the switching, the QOS assigned to person 2 increases from zero to around 7.0 (see Figure 6(b)).

In our assignment algorithm, when an object is occluded, the system will switch this object to another camera which views it better. To illustrate the efficiency of this switching in dealing with object occlusion, we compare it with another assignment algorithm, which is similar to the first algorithm, except that it does no object switching in the case of occlusion. We term the first the switching algorithm and the latter the non-switching algorithm. In the non-switching algorithm, when an object is occluded, the system still assigns this object to the camera which is currently tracking it. It means that each camera has to deal with the occlusion without the help of other cameras in the system. We run the two algorithms with the same scenario above and make a comparison of their performance using the QOS function.

Figure 8 plots the QOS of the system running with the two algorithms. As expected, the switching algorithm is able to maintain a higher QOS.

Figures 9(a) and 9(b) plot the QOSs that the system assigns to each person when it runs with the two algorithms. As can be seen from Figure 9(b), at 16.5 seconds, the QOSs assigned to person 2 in both cases drop to zero due to the occlusion. At 17.5 seconds, the QOS for the switching algorithm is not zero any more, because the system switches person 2 to camera C_2, which can see him clearly (this can be seen in Figure 6). In contrast, the QOS for the non-switching algorithm remains zero until 19.5 seconds. The advantage of the QOS in the case of using the switching algorithm also can be seen from 26.3 to 27.2 seconds and

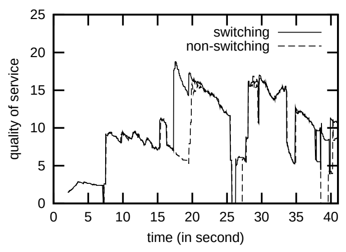
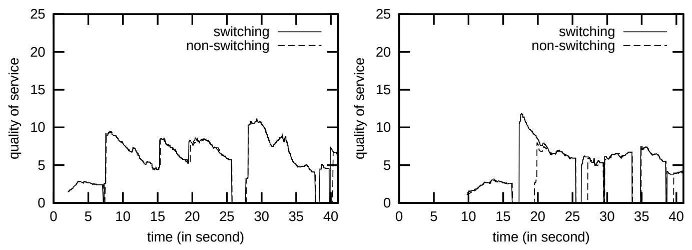


Figure 8. *The QOS of the system running with the switching algorithm and the non-switching algorithm.*



(a) $person$ 1 (b) $person$ 2

Figure 9. *The QOS assigned to (a) person 1 and (b) person 2 for the switching and the non-switching algorithms.*

from 38.7 to 39.6 seconds in Figure 9(b). These results show that the switching when the object occlusion occurs helps the system track the objects for longer periods and hence gets a higher QOS.

5 Conclusion and future work

In this paper, we have presented a distributed surveillance system to track multiple people using multiple cameras. In the system, we introduced an algorithm to coordinate the cameras to track people more reliably. The algorithm allocates objects to cameras using the object-to-camera distances while taking into account occlusion. We also have proposed a QOS function to measure the efficiency of the algorithm. Our experimental results show the robustness of our algorithm in dealing with the occlusion and its performance to maximize the QOS at all times.

Several future research directions can be considered. Firstly, better image processing algorithms are needed to estimate the features of the objects in the case of occlusion. This will reduce the time that objects are lost because they can be tracked better. There is also the issue of resource allocation when dealing with many objects. With a large number of cameras and objects involving in the system, the processing at the cameras should be balanced to guarantee the objects are tracked reliably. In this case, the QOS function needs to take into account other parameters such as the camera frame rate, the number of objects currently tracked by each camera, etc. Finally, we will scale up this system to work in a more complex spatial environment such as a complete buildings.

References

[1] Q. Cai and J. K. Aggrwal. Tracking human motion using multiple cameras. In *Proceedings of the 5th International Conference on Pattern Recognition*, pages 68–72, 1996.

[2] H. Freeman. Computer processing of line-drawing images. *Computing Surveys*, 6(1):57–97, 1974.

[3] N. Friedman and S. Russell. Image segmentation in video sequences: A probabilistic approach. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence*, 1997.

[4] I. Haritaoglu, D. Harwood, and L. Davis. w^4: Who? when? where? what? a real time system for detecting and tracking people. In *International Face and Gesture Recognition Conference*, 1998.

[5] T. Huang and S. Russell. Object identification in a Bayesian context. In *Proc. Fifteenth International Joint Conference on Artificial Intelligence*, Japan, 1997.

[6] S. S. Intille, J. W. Davis, and A. F. Bobick. Real-time closed-world tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, 1996.

[7] J. Orwell, P. Remagnino, and G. Jones. Multi-camera colour tracking. In *IEEE International Workshop on Visual Surveillance*, pages 14–21, 1999.

[8] H. Pasula, S. Russell, M. Ostland, and Y. Ritov. Tracking many objects with many sensors. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI99)*, 1999.

[9] K. Rangarajan and M. Shah. Establishing motion correspondence. In *CVGIP: Image Understanding*, pages 56–73, July 1991.

[10] C. Ridder, O. Munkelt, and H. Kirchner. Adaptive background estimation and foreground detection using kalman-filtering. In *Proceedings of International Conference on recent Advances in Mechatronics*, pages 193–199, 1995.

[11] C. H. P. K. Steiglitz. *Combinatorial Optimazation: Algorithms and Complexity*. Prentice Hall, New Jersey, 1982.

[12] S. Stillman, R. Tanawongsuwan, and I. Essa. A system for tracking and recognising multiple people with multiple cameras. In *Proc. 2nd Int. Conf. on Audio and Video based biometric person authentication*, March 1999.

[13] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-time tracking of the human body. In *In International Conference on Automatic face and Gesture Recognition*, pages 51–56, 1996.