This appears in: Workshop on Distributed Smart Cameras in conjunction with ACM SenSys'06

A Look-up Table Based Approach for Solving the Camera Selection Problem in Large Camera Networks

Johnny Park, Priya C. Bhat and Avinash C. Kak
School of Electrical and Computer Engineering
Purdue University, West Lafayette, Indiana 47907-1285
{jpark,pcbhat,kak}@purdue.edu

Abstract

Modern navigation, mapping and surveillance systems require numerous cameras mounted at random locations over a geographically large area. In order to efficiently extract any location-specific intelligence from such a large network of cameras, we propose to create distributed look-up tables that rank the cameras according to how well they can image a specific location. The look-up table covers all possible locations within the corresponding camera's *viewing frustum*[1] by optimally dividing the volume of the viewing frustum. The process of updating look-up tables is purely incremental, thus it can easily be used in a dynamic network. The foremost advantages of our approach are a significant reduction of run-time computation for selecting a set of *favorable* cameras using a simple lookup operation and a significant reduction of network traffic that would otherwise be required to carry out the process of camera selection. The proposed algorithm can be applied to various applications involving distributed cooperative processing in a large camera network.

1 Introduction

In recent years, the area of distributed sensor networks has received much attention. One of the important research issues in this field is the problem of selecting a subset of sensors in the network that is likely to carry out the current requested task most effectively. From the perspectives of reliability and accuracy, it would be desirable to use as many sensor measurements as possible. On the other hand, taking measurements at a large number of sensors and communicating these individual measurements between sensor nodes would result in a large consumption of energy, thus reducing the lifetime of the network. Even for sensor network systems with no energy constraints, the network bandwidth may constrain the number of measurements. This trade-off between the accuracy and the network efficiency necessitates the process of selecting a small number of sensors that are likely to provide the most accurate measurements.

The sensor selection problem is known to be NP-hard [5] and the situation becomes even more challenging in case of a camera-based sensor network for two main reasons: First, unlike the sensors that detect sound, temperature or radio signals, cameras are limited-aperture, directed sensors. This requires the consideration of viewing volume or frustum instead of range while determining the sensors that can detect the target. Secondly, in distributed camera networks, each node (consisting of an imaging device and a processing unit) should ideally store information about only the space it observes. The challenge is to execute a given task by finding and using cameras that are in the best position *globally*, through a process that takes place *locally*, in a *distributed manner* and in *real time*.

Our work achieves the above goals through look-up tables that are stored locally at each node and that are calculated through a distributed processing. The localized camera nodes calculate their viewing frustums and this information, along with the camera IDs, is flooded into the network. A receiving camera node compares the received frustum coordinates with those of its own and updates its own look-up table. The process of updating the look-up table is purely incremental, thus it can easily adjust to a dynamic network of cameras where addition, deletion and positional change of cameras can occur. We also suggest a novel method of volume quantization of a frustum that allows us to put an analytical bound on the memory requirement at each camera node. The look-up operation for choosing the cameras that are likely to provide the most accurate measurements is much simpler than a run-time computation. The speed gain is significant, since retrieving a value from memory is much faster than undergoing an expensive computation. This, along with considerable reduction in network traffic, are the salient features of our approach.

2 Related Work

Pottie and Kaiser [10] discussed the advantages, opportunities and constraints of distributed sensors as opposed to single large devices, and emphasize that *if the application and infrastructure permit, it pays to process data locally to reduce traffic volume* and make use of multi-hop routing to reduce network costs. Martinez and Bullo [8] presented an algorithm for optimal placement of range sensors for target tracking. The algorithm requires multi-sensor fusion to be performed at a central site. Isler et al. [5] used a restricted 2D camera geometry to solve the Focus of Attention (FOA) problem. The algorithm requires cameras to be placed along a straight line or along the circumference of a circle. Isler and

[1] By viewing frustum, we mean the region of space (rectangular pyramid extending from the optical center of the camera) where a target point would appear with an "acceptably" sharp focus in the image. (See Section 3.1 for further details)

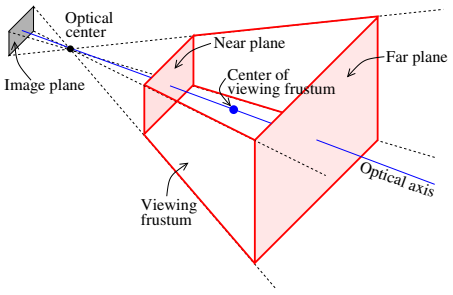

Figure 1. The pinhole model of a camera showing the near plane and the far plane that bound the volume that can be imaged by the camera. The viewing frustum is the 3D space enclosed by the near plane the and far plane as shown.

Bajcsy [4] applied the minimum enclosing parallelogram approach to develop a 2-approximation algorithm for selecting the best cameras. However, the implementation described in the paper requires central processing, and the alternate distributed implementation suggested by the authors results in look-up tables that would become prohibitively large as the network size increases. Further it is restricted to sensors whose measurement uncertainty could be represented by a convex polygon on a plane. Ng et al. [9] have made use of omni-directional cameras to develop a vision system that monitors a dynamically changing environment. The approach as such is not scalable to a large network. [2] presented an algorithm for multi-camera tracking in distributed video surveillance systems by calculating the entry edge of a field of view and the overlap in the field of view. The preliminary results indicate that the approach is computationally expensive and not directly usable in dynamic camera sensor networks.

The work presented in this paper distinguishes itself as we use the much more realistic pinhole camera model representing the 3-D *viewing frustum*, rather than a 2-D *viewing area* model that has been used in most of the previous works. To the best of our knowledge, ours is the first work to address the issue of look-up table generation for vision sensors without any central processing.

3 Background

3.1 Camera Viewing Frustum

Figure 1 shows the pinhole model of a camera illustrating the idea of the viewing frustum of a notional camera. In an ideal pinhole model, all points in front of the camera are visible as long as the projection ray from the point to the optical center intersects the image plane. In practice, however, all cameras have a limited depth of field, i.e., points that are too close to or too far away from the optical center may not appear in sharp focus in the image. The near and far planes, which are perpendicular to the optical axis, set boundaries in the viewing volume where a target point would appear with an "acceptably" sharp focus in the image. The resulting enclosed volume, depicted with red edges in Figure 1, is called the viewing frustum of the camera.

3.2 Concept of Favorable Cameras

Because some camera lens distortion is always present, as an object gets further away from the optical axis of a camera, the error in the image increases. Further, as the object gets close to either the near plane or the far plane that bound the frustum, the sharpness of the imaged points decreases. Therefore, it is reasonable to say that the closer the object is to the center of the viewing frustum, the more accurate the image measurements we can obtain.

Let $\mathbf{c}_i$ be the 3D coordinates of the center of the viewing frustum for camera C_i, and let $dist(\mathbf{c}_i, \mathbf{p})$ return the Euclidean distance between $\mathbf{c}_i$ and a 3D point $\mathbf{p}$. We call the camera C_i *favorable* for a particular $\mathbf{p}$, if $dist(\mathbf{c}_i, \mathbf{p})$ is very small. When comparing two cameras, say $C_1$and C_2, we say that C_1 is *more favorable* than C_2 for $\mathbf{p}$ if $dist(\mathbf{c}_1, \mathbf{p}) < dist(\mathbf{c}_2, \mathbf{p})$. This, of course, assumes that these two cameras have identical intrinsic and distortion parameters (i.e., the sizes and shapes of the two viewing frustums are the same). If the parameters of the two cameras are not identical, we can compare the normalized distances by dividing $dist(\mathbf{c}_i, \mathbf{p})$ with the total volume of the respective viewing frustums.

Note that our definition of favorable cameras is rather simple. One could use a much more sophisticated model, for example, by considering the target orientation along with its position. But we refrain from doing so because the basic idea here is to show how to overcome the difficulties of a distributed network for look-up table generation at each node of the network.

4 Use of Look-up Tables for Object Tracking

The look-up table approach can be applied to one of the most common applications in distributed smart cameras, namely, object tracking. Consider the scenario where we want to track an object with near constant certainty, typically decided by the target-camera geometry. We are given the initial position of the target and a small set of cameras that can view the target. The tracking process then can be carried out by selecting a set of cameras that would achieve the required tracking confidence, from the list of cameras in the look-up table entry that corresponds to the current target position. As the target moves, a new list of cameras can be retrieved from the look-up table and, if necessary, a new set of cameras can be selected.

5 Construction of Look-up Tables

We assume that all camera nodes have been localized i.e., each node is aware of its position and orientation with respect to a fixed reference coordinate frame. We refer to [3, 6, 7] for recently proposed localization algorithms for camera-based sensor networks. The goal is to efficiently generate look-up tables for every camera node in a distributed manner. *Every possible location within the viewing frustum will correspond to a particular entry in the look-up table where a sorted list of most favorable cameras for that location can be retrieved.*

5.1 Quantization of Viewing Frustum

In this section, we describe how to divide a viewing frustum into smaller unit volumes and create a one-to-one correspondence between a particular unit volume and its address in the

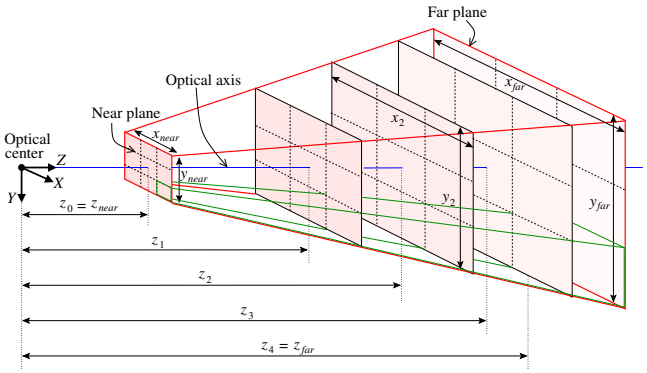

Figure 2. Frustum quantization into sub-frustums of equal volume. The near plane is divided into n_x equal parts along the X-axis, n_y equal parts along the Y-axis, and n_z *unequal* parts along the Z-axis, so that the volume of each of the $n_x n_y n_z$ sub-frustums is equal.

look-up table. The quantization should be such that given a 3D coordinates of a point, the corresponding look-up table address can be computed quickly and unambiguously.

One intuitive way to quantize a viewing frustum is to construct a rectangular box that encloses the viewing frustum and divide it into smaller unit boxes. Although this approach provides an easy way to create the correspondence between 3D coordinates and the look-up table address, much of the memory space is wasted since a large portion in the enclosing rectangular box does not belong to the viewing frustum. A simple solution that eliminates the waste of memory space would be to divide the viewing frustum into smaller frustums by equidistant planes perpendicular to the optical axis. However, the volume covered by each unit frustum will not be equal. In fact, the volume of a unit frustum close to the near plane and that close to the far plane will differ significantly.

In order to resolve the problems mentioned above, we propose a new frustum quantization strategy. Consider the image shown in Figure 2. We first divide the near plane into n_x equal parts along the X-axis and n_y equal parts along the Y-axis. In the figure, we have $n_x = n_y = 3$. The resulting $n_x n_y$ rectangles when projected onto the far plane will carve out frustums, each having $\frac{1}{n_x n_y}$ of the original volume. One such frustum is depicted with green edges. Now, we would like to divide each of the $n_x n_y$ frustums into n_z sub-frustums of equal volume with the help of planes perpendicular to the optical axis. Therefore, the distances from the optical center to these planes, $z_0, z_1, \ldots, z_{n_z}$, need to be calculated. Actually, we only need to calculate $z_1, z_2, \ldots, z_{n_z-1}$ since z_0 is the known distance from the optical center to the near plane (denoted as z_{near}) and z_{n_z} is the distance to the far plane (denoted as z_{far}). Let x_{near} and y_{near} be the lengths of the near plane in X and Y axes, respectively, and similarly x_{far} and y_{far} for the far plane. Also, let A_{near} and A_{far} be the areas of the near and far planes, respectively (e.g., $A_{near} = x_{near} y_{near}$). Then, the volume of the entire frustum, V, is given by

$$V = \frac{z_{far} - z_{near}}{3} \left(A_{near} + A_{far} + \sqrt{A_{near} A_{far}} \right) \qquad (1)$$

Now we would like to calculate z_k, the distance between the optical center to the k-th plane with the side lengths denoted as x_k and y_k. Using the property of similarity, we have

$$\frac{z_{near}}{x_{near}} = \frac{z_k}{x_k} = \frac{z_{far}}{x_{far}} \quad \text{and} \quad \frac{z_{near}}{y_{near}} = \frac{z_k}{y_k} = \frac{z_{far}}{y_{far}} \qquad (2)$$

For our problem, the volume of each sub-frustum must be $\frac{1}{n_z}$-th of the total volume. Hence we have

$$\frac{z_k - z_{near}}{3} \left(A_{near} + A_k + \sqrt{A_{near} A_k} \right) = \frac{k}{n_z} V \qquad (3)$$

where V is the volume of the entire frustum from Eq. (1) and A_k is the area of the k-th plane. Using Eqs. (2) and (3), we can now derive z_k as

$$z_k = \left\{ \frac{k z_{far}^3 + (n_z - k) z_{near}^3}{n_z} \right\}^{\frac{1}{3}} \qquad (4)$$

Dividing the entire frustum with n_z planes that are perpendicular to the optical axis and the distances from the optical center are computed by Eq. (4), we now have $n_x n_y n_z$ frustums of equal volume. Obviously, the size of these sub-frustums can be controlled by varying the values of n_x, n_y and n_z.

5.2 Look-up Operation

We structure the look-up table of a camera's viewing frustum as a 3-dimensional array, denoted as $LookUp[u][v][w]$ where u is the index along the X-axis, v the Y-axis and w the Z-axis. We will now show, given 3D coordinates of a point that lies inside the frustum, how its corresponding address in the look-up table can be obtained. Recall that each camera knows its position and orientation with respect to a common reference coordinate frame. This implies that 3D coordinates with respect to the reference coordinate system can easily be transformed into the camera's pixel coordinate frame. Let a 3D point $\mathbf{p} = (x_p, y_p, z_p)$ defined with respect to the reference coordinate frame has its corresponding pixel coordinates (u_p, v_p). If $Addr(\mathbf{p})$ is a function that returns the address (i.e., three indices) of a look-up table entry that corresponds to $\mathbf{p}$, then

$$Addr(\mathbf{p}) = \left(\left\lfloor \frac{u_p}{n_x} \right\rfloor, \left\lfloor \frac{v_p}{n_y} \right\rfloor, \left\lfloor n_z \frac{(z'_p)^3 - z_{near}^3}{z_{far}^3 - z_{near}^3} \right\rfloor \right) \qquad (5)$$

where $\lfloor . \rfloor$ is the flooring function and z'_p is the perpendicular distance between the point $\mathbf{p}$ and the optical center (i.e., simply the z coordinate of $\mathbf{p}$ with respect to the camera coordinate frame). Note that the third index of the look-up table is derived by solving for k in Eq. (4) . The computation required in a single look-up operation is very efficient since z_{near}^3, z_{far}^3, n_x, n_y, and n_z are all constants, and the variables u_p, v_p and z'_p are all directly computed during the transformation computation from the reference coordinate frame to the pixel coordinate frame.

5.3 Distributed Construction of Look-up Tables

In this section, we describe how individual look-up tables are constructed in a distributed manner. Recall that every look-up table entry contains a sorted list of most favorable cameras for that particular location where the location of the sub-frustum is simply represented by the center of the sub-frustum. Thus, each entry $LookUp[u][v][w]$ in camera C_i contains the following data: (1) the center of this sub-frustum, denoted as $\mathbf{c}_{i,u,v,w}$; and (2) a sorted list of most favorable cameras, which consists of camera IDs and the distance from $\mathbf{c}_{i,u,v,w}$ to the center of viewing frustum of the corresponding camera.

As soon as a camera node is localized with respect to a reference coordinate frame, the camera first constructs its look-up table and for every look-up table entry, it initially inserts its own camera ID along with the distance to the center of the viewing frustum at the top of the list. Then, the camera broadcasts the vertices of the viewing frustum and its camera ID to the whole network. An incremental process of look-up table construction now takes place as follows:

1. A receiving camera C_i calculates the overlap between its own viewing frustum and the viewing frustum of the sending camera C_j by exhaustively running through the center coordinates of all sub-frustums $\mathbf{c}_{i,u,v,w}$ whether or not it lies within the viewing frustum of C_j. Note that much more efficient algorithms that use the concept of quad-trees and oct-trees have been developed, some of which have been discussed in [1].
2. For every point $\mathbf{c}_{i,u,v,w}$ that lies within the viewing frustum of camera C_j, compute $dist(\mathbf{c}_j, \mathbf{c}_{i,u,v,w})$ where $\mathbf{c}_j$ is the center of viewing frustum of camera C_j, then insert camera C_j to the sorted list of favorable cameras at $LookUp[u][v][w]$ of camera C_i.

Figure 3 illustrates the look-up table updating process with a 2-D example for simplicity. Suppose camera C_i is the receiving camera which gets the frustum coordinates of camera C_j. In the first step, as shown on the left image, C_i detects that $\mathbf{c}_{i,u,v,w}$, the center of the sub-frustum at (u,v,w), lies inside the viewing frustum of camera C_j. Since $dist(\mathbf{c}_j, \mathbf{c}_{i,u,v,w}) > dist(\mathbf{c}_i, \mathbf{c}_{i,u,v,w})$, the sorted list of favorable cameras at $LookUp[u][v][w]$ of camera C_i is updated as camera i at the top followed by camera j. On the right image, C_i now receives the frustum coordinates of a new camera C_k and detects $\mathbf{c}_{i,u,v,w}$ also lies inside the viewing frustum of C_k. Again, since $dist(\mathbf{c}_j, \mathbf{c}_{i,u,v,w}) > dist(\mathbf{c}_i, \mathbf{c}_{i,u,v,w}) > dist(\mathbf{c}_k, \mathbf{c}_{i,u,v,w})$, the list is updated accordingly as shown in the image.

The above method of look-up table generation is incremental because for any changes in the system like addition, deletion or moving of a camera (in case of dynamic networks), the new frustum coordinates need to be broadcast and each camera has to update its look-up table only in the affected locations.

In developing the favorable list of cameras, we have used the distance of the world coordinate from the center of viewing frustum as the criterion. As mentioned before, this model is highly simplistic and has been used here only because our aim is not to furnish new methods of choosing the best cameras. We emphasize that even for a much more sophisticated camera ranking technique, our algorithm can be used as such to generate look-up tables in a distributed network. In case of wireless cameras, where every node has only limited range of communication, the number of hops to be made while communicating can be used as one of the criteria for developing a favorable sort.

The look-up table generating process is optimal as each camera node stores only the information about the coordinates observed by it. During any applications, no real-time camera selection protocol is needed, drastically reducing communication and computational load of the network. Instead of having to use a fixed number of cameras, these look-up tables provide the flexibility of making the decision of how many cameras to use *on the fly,* as desired by the application.

6 Experimental Results

We have developed a simulation program for testing our look-up table generation process. The program allows a user to place cameras either at user-specified or random positions. In either case, the system assumes that all cameras are localized. As soon as a new camera is added, the camera broadcasts its frustum coordinates to the rest of the cameras. Each camera that receives this the new frustum coordinates updates its own look-up table. Figure 4(a) shows a simple example illustrating the process of look-up table update. In this example, all look-up tables consist of $5 \times 5 \times 5$ sub-frustums (i.e., $n_x = n_y = n_z = 5$). For visualization purpose, we show the look-up table only for camera C_1 and only the middle row (i.e., $n_z = 3$) of this look-up table. Some of the possible favorable camera lists for C_1 are represented by various colors as shown at the far left of Figure 4(a). The first image shows the initial state of the look-up table where all entries have their own camera IDs, 1, at the top of the favorable camera list. The image in the middle shows the updated look-up table after camera C_2 has been added. Notice the sub-frustums closer to the center of the viewing frustum of C_2 now have the favorable camera list sorted as 2 followed by 1 (shown as light green sub-frustums). Other sub-frustums that are also within the frustum of C_2, but closer to the center of the viewing frustum of C_1 have the favorable camera list sorted as 1 followed by 2 (shown as brown sub-frustums). The image on the right shows the look-up table after camera C_3 has been added.

Figure 4(b) shows a network of 20 cameras, each with a distinct color. A color-coded look-up table for one camera is shown where the color at each sub-frustum represents the color of the most favorable camera. All look-up tables in this example consist of $10 \times 10 \times 10$ sub-frustums. In this experiment, the average processing time for updating a look-up table (e.g., adding a new camera) was less than 0.001 seconds.

7 Conclusion and Future Work

We have demonstrated a robust, distributed algorithm to efficiently create look-up tables for storing information about the most favorable cameras for viewing a given point in the

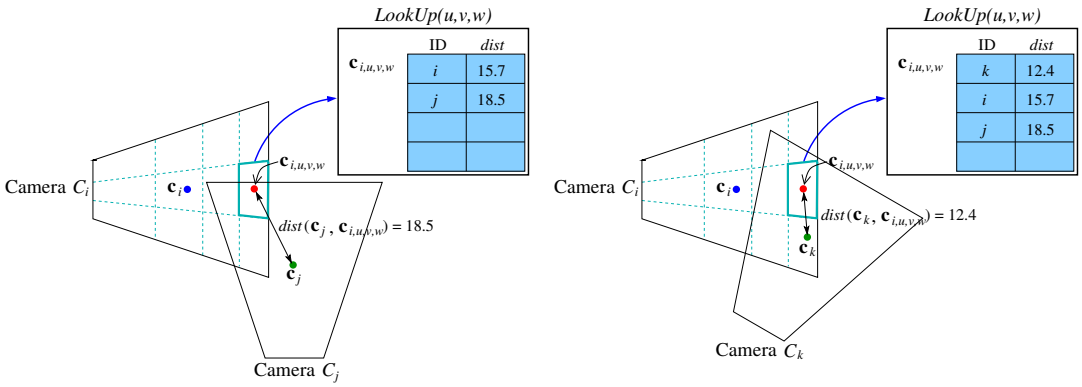

Figure 3. An illustration of the look-up table updating process. On the left, Camera C_i receives the frustum coordinates of Camera C_j and detects that the center of the sub-frustum $\mathbf{c}_{i,u,v,w}$ is also seen by C_j; further, $dist(\mathbf{c}_j, \mathbf{c}_{i,u,v,w}) > dist(\mathbf{c}_i, \mathbf{c}_{i,u,v,w})$, making C_i more favorable than C_j. The entry corresponding to (u,v,w) in the look-up table of C_i is shown. On the right image, C_i receives the frustum coordinates from a new camera C_k. The updated look-up table entry is shown.

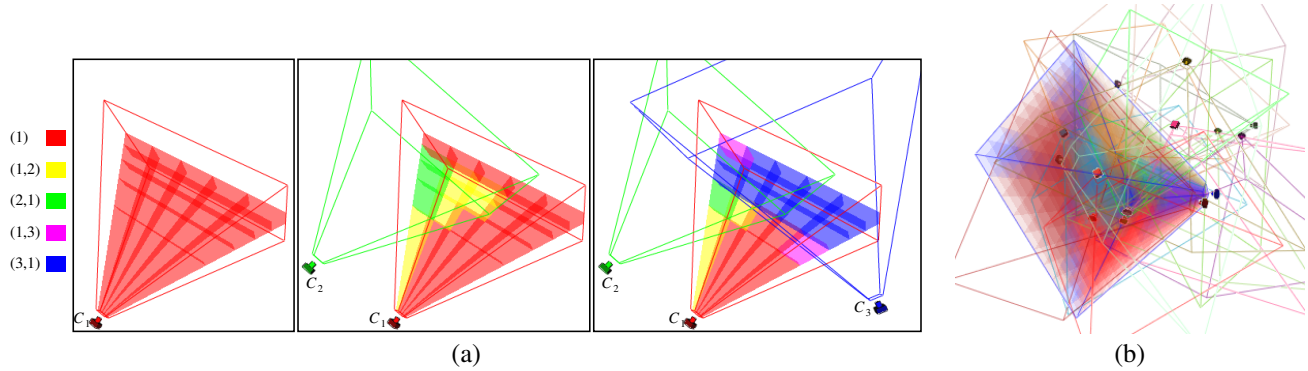

Figure 4. (a): A simple example showing the process of look-up table update. (b): A network of 20 cameras each with a distinct color. A color-coded look-up table of one camera is shown where the color at each sub-frustum represents the color of the most favorable camera.

frustum of a given camera. This information can be readily used for efficient camera hand-off as a target moves around in the 3D space.

8 References

[1] U. Assersson and T. Möller. Optimized view frustum culling algorithms for bounding boxes. *Journals of Graphics Tools*, 5(1):9–22, 2000.

[2] S. Calderara, R. Vezzani, A. Prati, and R. Cucchiara. Entry edge of field of view for multi-camera tracking in distributed video surveillance. In *IEEE Conference on Advanced Video and Signal Based Surveillance*, pages 93–98, 2005.

[3] D. Devarajan and R.J. Radke. Distributed metric calibration of large camera networks. In *Proc. International Conference on Broadband Networks*, 2004.

[4] V. Isler and R. Bajcsy. The sensor selection problem for bounded uncertainty sensing models. In *Fourth International Symposium on Information Processing in Sensor Networks*, pages 151–158, 2005.

[5] V. Isler, S. Khanna, J. Spletzer, and C. Taylor. Target tracking with distributed sensors: the focus of attention problem. *Computer Vision and Image Understanding*, 100(1-2):225–247, October 2005.

[6] H. Lee and H. Aghajan. Collaborative self-localization techniques for wireless image sensor networks. In *Proc. Asilomar Conference on Signals, Systems and Computers*, 2005.

[7] X. Liu, P. Kulkarni, and P. Shenoy. Snapshot: A Self-Calibration Protocol for Camera Sensor Networks. Technical report, Department of Computer Science, University of Massachusetts, 2005.

[8] S. Martinez and F. Bullo. Optimal sensor placement and motion coordination for target tracking. *Automatica*, 2004.

[9] K. Ng, H. Ishiguro, M. Trivedi, and T. Sogo. Monitoring dynamically changing environments by ubiquitous vision system. In *Second IEEE Workshop on Visual Surveillance*, pages 67–73, 1999.

[10] G.J. Pottie and W.J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.