# Counting People in Crowds with a Real-Time Network of Simple Image Sensors

Danny B. Yang
dbyang@cs.stanford.edu
Computer Science Dept.
Stanford U., CA 94305

Héctor H. González-Baños
hhg@hra.com
Honda R&D, Americas
Mountain View, CA 94041

Leonidas J. Guibas
guibas@cs.stanford.edu
Computer Science Dept.
Stanford U., CA 94305

## Abstract

*Estimating the number of people in a crowded environment is a central task in civilian surveillance. Most vision-based counting techniques depend on detecting individuals in order to count, an unrealistic proposition in crowded settings. We propose an alternative approach that directly estimates the number of people. In our system, groups of image sensors segment foreground objects from the background, aggregate the resulting silhouettes over a network, and compute a planar projection of the scene's visual hull. We introduce a geometric algorithm that calculates bounds on the number of persons in each region of the projection, after phantom regions have been eliminated. The computational requirements scale well with the number of sensors and the number of people, and only limited amounts of data are transmitted over the network. Because of these properties, our system runs in real-time and can be deployed as an untethered wireless sensor network. We describe the major components of our system, and report preliminary experiments with our first prototype implementation.*

## 1. Introduction

Real-time estimates of a crowd size are valuable in many situations. A real-time count can be used to enforce the occupancy limit in a building, to actively manage city services and allocate resources for public events, to aid with crowd control during rallies, and to detect unusual situations at an airport. Currently, there are no proven techniques for estimating crowd size. Existing techniques either use aerial photographs or require people to estimate how many people pass by several checkpoints. The first approach is only possible outdoors. Also, automated counts from aerial photographs are difficult because of limited resolution and occlusions. Furthermore, this count is only valid for the instance in time when the photographs were taken. In the second approach, the counts from the different checkpoints can be combined to approximate the total count of the crowd. However, people may have difficulty estimating the count at a busy checkpoint. Both of these approaches are labor intensive, and neither can produce real-time results.

Counting crowds is difficult because there are many occlusions. Even with strong prior assumptions and no computational limitations, often it is impossible to count the crowd from a single view. A possible solution to this problem is to use many sensors in a sensor network. The sensor network can form clusters according to the geometry so that each cluster can count the number of people at a local checkpoint. The clusters can communicate with each other to determine the global count of the crowd in the area that all the checkpoints enclose. The sensor nodes have limited computation and the network has limited bandwidth. Thus, only simple processing can be done on the images and the data must be aggregated efficiently. Moreover, the algorithms must be lightweight enough to run in real-time.

In this sensor network setting, we describe an approach to count crowds. Our current prototype cluster consists of 8 image sensor nodes networked to a central node that counts people in real-time. The system is scalable and robust, so many of these clusters can be combined into a much larger sensor network to count over a much larger area.

Traditionally, counting involves first locating all the individual objects. However, locating all the objects is a demanding task because objects often look alike or occlude each other, making data association difficult. In crowded situations some objects may be completely hidden from all views and therefore impossible to localize individually. To avoid these pitfalls, our technique is based on the computation of bounds on the number of objects in a region and not on localizing individual objects. This crucial difference allows our system to function well even in crowded settings.

In our system, each sensor extracts foreground objects from the background and sends the resulting bitmaps (i.e., silhouettes) over the network. Both the processing and network bandwidth required for this are low. The data from all sensors is aggregated in order to compute a planar projection of the scene's visual hull. This projection is used to bound the number and possible locations of people, a non-trivial task given that portions of the visual hull may in fact be empty. The system tracks regions of space that are determined to be occupied. Our approach is not based on the explicit detection of people in the images, so there is no pairwise matching of people across frames. Thus, our computational requirements scale well with the number of sensors and the number of people. Also, the resulting system is robust to failures of individual sensors.

The paper is organized as follows: Section 2 describes background work. Section 3 introduces the techniques developed for our sensor network. The system architecture and experiments are presented in Section 4. Finally, in Section 5, we make closing remarks and describe future work.

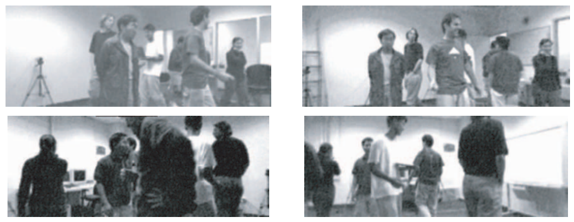


Figure 1: Typical views of 7 people in our setup.

## 2 Background

**Detecting People** Several multi-camera systems tackle the problem of tracking objects across multiple views [1, 4]. Some use stereo techniques to aggregate the information from different cameras. For example, in [5, 8], a stereo camera is used to locate the people, whereas in [17] two sets of stereo cameras are used. Similarly, in the real-time system proposed in [24] a stereo-like algorithm is applied to all pairs of omnidirectional cameras. Although multi-baseline stereo could be used to aggregate the data from more than two pairs of cameras, the applicability of this method is limited due to its computational cost.

Other techniques depend on using a shape [8, 9] or color [5, 17, 22] model to distinguish different objects in each view. For instance, in [21], individual objects are extracted using both color and shape, and their locations are determined by pairwise matching between cameras. Probabilistic models [12, 21] can be used to determine whether objects observed in different views are the same. However, matching objects across pairs of views can be computationally intensive due to the large number of possible matches.

Multiple objects have also been tracked from a single view. In [15], a particle filter is used to track several people. In [27], an MCMC approach is used to segment individual people from a crowd. However, these two approaches are currently too expensive in the sensor network setting.

Counting often follows tracking. If we can track objects, then we can count them. One scenario for counting is to use non-overlapping sensors placed along a route [12, 16] and count the number of passing objects. Object correspondences across views can be calculated using motion models. This approach still depends on tracking individual objects as they move between sensor and does not scale as the number of objects gets large as in the case of a crowd.

Counting becomes most interesting when it is infeasible to track all the people in a scene. This situation typically arises in crowded scenes (Figure 1), where people are often occluded from all views and therefore impossible to detect.

**Sensor Network Architecture** Our sensor network is assumed to be composed of simple elements. Modern CMOS fabrication techniques allow groups of logic gates to be etched next to each pixel without significantly increasing the cost of the sensor [18]. But this added power is only useful for computations involving neighboring elements to each pixel. Thus, complex operations such as object detection or separation still require the use of a dedicated computer, and are therefore disallowed in our architecture.

The sensor network also has limited bandwidth. It is undesirable to transmit images from all sensors to a powerful central computer in order to compute pairwise matching between all sensors. At best we can assume that images can be compared only among neighboring sensors. But even this assumption causes complications in practice.

Additionally, our simple sensors have no object model and only do background subtraction at the local level. No object detection or separation is done for individual views.

For more background on sensor networks and their architectural constraints, see [6, 10, 13, 20, 23].

**Visual Hulls** The visual hull is the intersection of all cones swept out by the silhouettes of objects seen from all camera views. It is the largest volume in which objects can reside that is consistent with all the silhouette information.

The exact visual hull is computed in [19]. Voxel approximations are computed in [3, 25, 26]. [3, 19] compute the visual hull in real-time (15 fps) on a dedicated computer. In this paper, only a planar projection of the visual hull is required, reducing the computational cost even further.

Our projection approximates a top view of the scene. In [14], an overhead camera tracks objects in a closed-world setting. However, their technique represents objects explicitly and requires knowledge of the exact object count, which may be unavailable, especially for crowded environments.

## 3 Techniques and Algorithms

Our goal is to determine bounds for the count and location of people in a room from a planar projection of the visual hull. The first step is to compute this projection from the silhouettes measured by the sensors through background subtraction. The projection is a set of polygons. The second step is to compute bounds to the number of objects in each polygon. As objects move, these bounds change and can be improved over time. A tree is used to record their history. Finally, the tree and its associated polygons are used to lo-

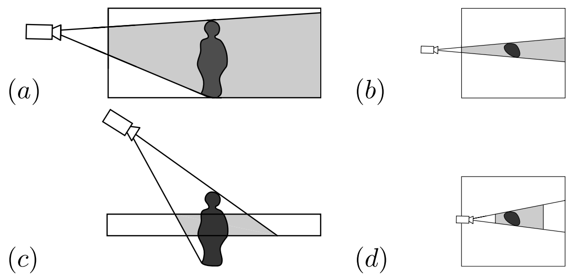


Figure 2: Projection of a silhouette cone. $(a)$ and $(c)$ are side views; $(b)$ and $(d)$ are the corresponding top views showing the projection onto the ground plane.

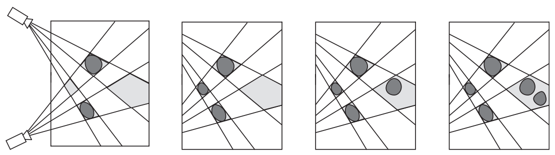


Figure 3: Different object arrangements can be consistent with a given visual hull. Polygons devoid of objects are called phantoms.

calize those workspace regions that are occupied by people. All of these steps are explained in detail in this section.

## 3.1 Projection of the Visual Hull

Ordinarily, people move along a plane. Therefore, the projection of the 3D visual hull onto this plane contains the information most useful for counting and localizing people.

Figure 2 shows how we project the 3D visual hull. Each measured silhouette sweeps a cone in 3D space. These cones are projected onto a plane and intersected in 2D. The side view of a camera looking at a person is depicted in $(a)$, with the plane of projection parallel to the floor. The box shown in the figure marks the lower and upper bounds of the projection that is to be flattened to the plane. The shaded region is a cross-section of the 3D cone formed by the silhouette of the person. This cone is projected onto the plane as seen from the top in $(b)$. The shaded area in $(b)$ represents the projected visual hull from one camera. $(c)$-$(d)$ show the projection with a different camera position and narrower upper and lower bounds for the projection.

Our planar projection of the visual hull is the intersection of the projected silhouette cones from all cameras. Note that we project the silhouette cones and then intersect them in 2D. This is not exactly equal to first computing the 3D visual hull and then projecting the result. The latter can be a subset of the former, but in our problem setting, both are close to equal because people tend to occupy most of the vertical space of their projection. The advantage of our projection is that it is much cheaper to compute.

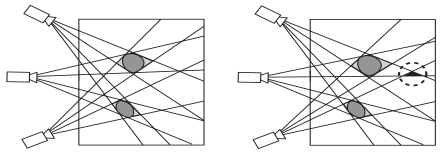


Figure 4: Two successive views showing an appearing phantom.

**Pruning Polygons** The polygons composing the projected visual hull represent all possible regions in the plane that may contain an object. The projected visual hull is ambiguous, because several arrangements of objects can be consistent with a given visual hull (Figure 3). Some of the polygons in the projection may be empty and we call these polygons *phantoms*. The next step is to prune as many of the phantoms as possible using geometric constraints.

Polygons are pruned based on size and temporal coherence. The following must be phantoms and are pruned:

- Polygons smaller than the minimum object size.
- Polygons that appear from nowhere.

The second type of phantom is a property of three or more cameras as shown in Figure 4. Two objects are observed in two successive time steps, and a phantom appears on the right. To check the temporal coherence of a polygon we test if it intersects a polygon in the previous step. This assumes that objects cannot leave the area created by their visual hull in a single step. This maximum speed assumption can be adjusted by growing the polygons in the previous step before computing the intersection.

## 3.2 Object Bounds Using a History Tree

After pruning, the next step is to bound the number of objects inside each polygon. We cannot do this exactly because objects may be fully occluded by other objects. Instead, we keep track of the lower and upper bounds of the number of objects in each polygon. If these bounds converge over time then we have an exact count of the objects in that polygon.

**Upper Bound Constraint (UBC )** A constraint on the upper bound is the area of the polygon divided by the minimum object size. This bound is very loose because it assumes the worst possible scenario: objects cluster and move collusively together as a single target. This bound also assumes that objects fill the entire area of the polygons regardless of their geometry (acting like water).

**Lower Bound Constraint (LBC )** A ray from a camera intersects a polygon only if the corresponding line of sight was blocked by an object. If only one polygon intersects that ray then said object must be contained in the polygon. Therefore, a polygon contains at least one object if there exists a ray from a camera that intersects only that polygon.

This constraint is different from UBC in that it counts distinct objects directly. A real object was observed along a ray and counted. In contrast, UBC hypothesizes about the maximum number of objects that *could* fill a polygon.

**Tree Structure ($\mathcal{T}$)** Although LBC only tells us if a polygon contains at least one object, its behavior through time conveys additional information. To this end, we keep track of the bound's history with a tree structure updated at each time step. By propagating the bounds along this tree the number of objects in the scene can be further constrained.

At time $t$, each leaf in the tree stores a newly observed polygon and its associated object bounds. A node in the tree represents the implicit union of all the polygons of its descendants — it contains the bounds to the number of objects inside this union. From this, we have the following 4 properties on the object bounds across the tree:

$$l_i = \max(l_i, \sum_{\forall j \in \textit{children}(i)} l_j) \quad (1)$$

$$l_i = \max(l_i, l_{\textit{parent}(i)} - \sum_{\forall j \in \textit{siblings}(i)} u_j) \quad (2)$$

$$u_i = \min(u_i, \sum_{\forall j \in \textit{children}(i)} u_j) \quad (3)$$

$$u_i = \min(u_i, u_{\textit{parent}(i)} - \sum_{\forall j \in \textit{siblings}(i)} l_j) \quad (4)$$

Eqn. (1) states that if there are at least $\sum l_j$ objects in the children polygons then the original parent must contain at least this many objects. Eqn. (2) states that if there are at least $l_{\textit{parent}}$ objects in the parent, and $\sum u_{\textit{siblings}}$ objects fit inside the sibling polygons, then the difference must be in the child. Reverse constraints apply for the upper bounds. At the leaf level we have the constraints UBC and LBC for individual polygons described before.

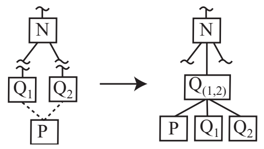
## 3.3 Updating the Tree

Let $\mathcal{T}$ be the structure of the tree at time $t$. Let $\Pi(t+1)$ be the polygons observed at time $t+1$. $\mathcal{T}$ is updated in three steps: add new leaves to $\mathcal{T}$, remove redundant nodes from $\mathcal{T}$, and update bounds across $\mathcal{T}$.

**Add Leaves** Each $P \in \Pi(t+1)$ is added as a leaf to $\mathcal{T}$ using the following operations:

ADD $(P)$ TO $(Q)$ **:** $P$ intersects exactly one polygon $Q \in \Pi(t)$ —that is, objects in $P$ must have originated from $Q$. $P$ is added as a child of $Q$. The bounds of the new leaf containing $P$ are initialized by UBC and LBC .

ADD $(P)$ TO $(Q_1, Q_2)$ **:** $P$ intersects exactly two polygons $\{Q_1, Q_2\} \in \Pi(t)$. Objects in either $Q_1$ or $Q_2$ could have moved to $P$. Adding $P$ as a child to both $Q_1$ and $Q_2$ creates a cycle in $\mathcal{T}$. Instead, we create a new node $Q_{(1,2)}$, added as a child to node $N \in \mathcal{T}$ —the closest common ancestor of $Q_1$ and $Q_2$. Now $P$, $Q_1$ and $Q_2$ become children of $Q_{(1,2)}$:



The bounds of the leaf containing $P$ are initialized by UBC and LBC . Additionally, in order to keep the properties of the tree correct, the bounds of $Q_{(1,2)}$ are initialized to be the combined bounds of $Q_1$ and $Q_2$. The lower bounds of all the nodes along the path from $Q_1$ to $Q_2$ (before they were moved) are decreased by the upper bound of $P$. These include $Q_1$ and $Q_2$, but not $N$.

ADD $(P)$ TO $(Q_1, \ldots, Q_k)$ **:** $P$ intersects $\{Q_1, \ldots, Q_k\} \in \Pi(t)$. This is handled by several nested calls to ADD. Initially, ADD $(P)$ TO $(Q_1, Q_2)$ is called. For subsequent calls, $P$ is removed from $Q_{(i-1,i)}$ and ADD $(P)$ TO $(Q_{(i-1,i)}, Q_{i+1})$ is called.

**Remove Redundant** Once all new polygons in $\Pi(t+1)$ are added, we proceed to remove those nodes in $\mathcal{T}$ that are redundant:

REMOVE REDUNDANT$(\mathcal{T})$ **:** Every element $N \in \mathcal{T}$ with one child or less is removed, unless $N \in \Pi(t+1)$ (i.e., $N$ is a newly added polygon). The bounds for the child of $N$ (if any) are updated to be the tighter among the two.

This stage guarantees that only those polygons in $\Pi(t+1)$ are leaves of $\mathcal{T}$, and that any other node in $\mathcal{T}$ has at least two children. Therefore, the number of leaves in $\mathcal{T}$ is equal to the number of observed polygons $n$, the depth of the tree is less than or equal to $n$, and $|\mathcal{T}| < 2n$.

**Update Bounds** Object bounds are updated across the tree to ensure that Eqns. (1-4) hold for every node. This involves two sweeps: First, new information from the leaves is propagated up to the root of $\mathcal{T}$. Afterwards, the updated bounds for the root are propagated back down to the leaves.

## 3.4 Counting Algorithm

The basic structure of the counting algorithm is as follows:

1. *Get new readings from each sensor. Compute the planar projection of each silhouette.*
2. *Compute the intersection of all projections. Store the resultant polygons in* $\Pi(t+1)$.
3. *Remove small polygons and phantoms from* $\Pi(t+1)$.
4. *Update the tree structure* $\mathcal{T}$.
5. *Report the new bounds on the number of objects in the workspace.*

*Repeat for* $t \leftarrow t+1$.

At first glance, Step 2 appears to be an expensive operation. In fact the cost is only linear in the number of cameras and takes very little time. In our implementation, the workspace is discretized to be a $144 \times 144$ grid (corresponding to 12 feet squared). The projections of all silhouettes can be discretized and intersected in real-time using graphics acceleration hardware. This operation is trivial in a modern graphics card.

Step 4 is $O(n^2)$, where $n$ is the number of observed polygons. This is because the amortized cost of adding one leaf is $O(n)$. However, this cost is very different from the quadratic number of comparisons associated with most pairwise-matching algorithms. There, each comparison is expensive because it is dependent on both the image resolution and the number of cameras. Their overall cost is more than quadratic. In contrast, our overall cost is $O(n^2 + c)$, where $c$ is the number of cameras. Moreover, the actual cost of maintaining $\mathcal{T}$ in practice is small and $\mathcal{T}$ can be maintained in real time for very large values of $n$.

**Objects Entering or Exiting the Workspace** If we allow objects to enter and exit the workspace, then some polygons in $\Pi(t+1)$ will not intersect any polygons in $\Pi(t)$. The basic counting algorithm must be extended.

As objects enter or exit, their corresponding polygons touch the workspace boundary. We call these *border polygons*. Border polygons at $t+1$ that do not intersect polygons in $\Pi(t)$ could be objects entering the workspace. Therefore, these polygons are added as children of the root node of $\mathcal{T}$ and its upper bound is increased accordingly.

In contrast, border polygons that intersect polygons in $\Pi(t)$ could be objects either entering or exiting the workspace. But these polygons have already been added to $\mathcal{T}$. For each of these border polygons, we set the lower bound to zero. We decrease the lower bounds of its ancestors by the upper bound of the border polygon to account for the possibility of an object exiting. Likewise, the upper bound of the ancestors of a border polygon are increased to account for the possibility of an object entering.

**Localizing Objects** The tree structure can be used to localize objects by using the bounds and leaf polygons. Each leaf polygon is fitted with circles, which approximate the cross sections of people. If a polygon contains a single object, then the polygon — and its corresponding circle — is a good approximation of the object inside. When these polygons merge, we compute the locally optimum arrangement of circles inside the new polygon. The circles inside all polygons are an estimate of all possible locations of objects. These locations are pruned by choosing only circles inside polygons with lower bound greater than or equal to the number of circles, to prune out possible phantoms.

## 4 Implementation and Experiments

Cameras were placed pointed horizontally. We chose to test this case because it is the most difficult case and results in the most occlusions. The projection of the silhouette cones onto the ground plane is the least constraining for horizontal cameras. In Figure 2, the projection from a horizontal camera (b) is much larger than that from a camera looking downwards (d). The other extreme is to place the cameras directly overhead pointed downwards. This makes the problem trivial because there is almost no occlusion and the projection from an overhead camera matches the actual objects very accurately. Our counting algorithm becomes more accurate with cameras that are more directly overhead. However, in many real world situations, it is not possible to cover a space with overhead cameras. So we decided to test the worst case scenario when all the cameras are horizontal.

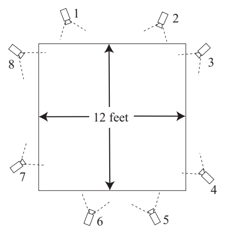


Figure 5: Top view of workspace with the cameras' FOV lines.

Our system consists of 8 calibrated cameras arranged around a rectangular room. Each camera is roughly 4 feet off the ground and points horizontally towards the room's center. The cameras surround a $12 \times 12$ ft workspace. No single camera covers the entire area (Figure 5).

The 8 cameras are connected to a pair of dual processor 933 MHz Pentium III computers. Background subtraction is done using the technique described in [11]. Each computer grabs images from 4 cameras, and undistorts and runs background subtraction on each image. Each camera, with its background subtraction process, is modelled as a simple image sensor and establishes its own TCP/IP connection to the central computer. The central computer, a 800 MHz Pentium III, queries each sensor over TCP/IP for the silhouettes, computes the projected visual hull, and runs the counting algorithm described in Section 3

The sensors are not synchronized and only send data when queried. At a resolution of 640x240, the sensors do background subtraction at 15 fps and constitute the system bottleneck. With specialized hardware for background subtraction, the rate can be much higher. The foreground bitmaps can be transmitted very efficiently over the network. The central computer is actually capable of running the counting algorithm at 60 fps.

In the implementation, polygons are represented as a collection of grid points. This makes the polygons more robust to noisy silhouettes. Grid points are incremented when they become part of a projected silhouette. However, as a grid point ceases to be part of the visual hull, its value decays exponentially instead of being reset to zero — grid points

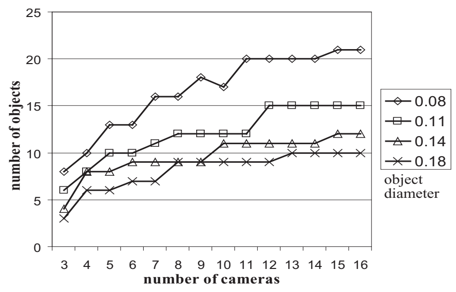


Figure 6: Synthetic objects moving in a square room with width 1. Plotted are the maximum number of objects that a given number of cameras was able to count exactly.

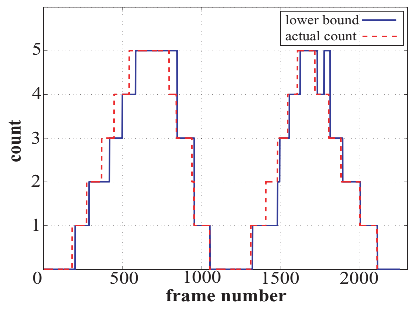


Figure 7: Count of 5 people walking into and out of the workspace.

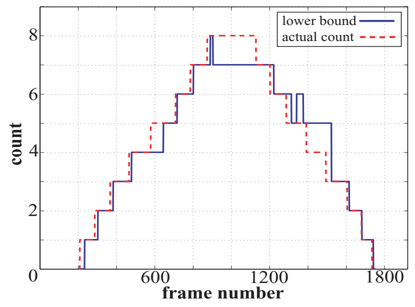


Figure 8: Count of 8 people walking into and out of the workspace.

are persistent. If there is a sudden erroneous cut through a polygon because of a noisy silhouette, persistent grid points prevent the polygon from fragmenting into smaller pieces. The overall effect is a temporal smoothing of the polygons (the weight drops to 10% in 0.22 seconds).

## 4.1 Synthetic Experiments

Synthetic experiments with noise-less data were done to determine the best-case results for a group of horizontal cameras. The synthetic cameras were evenly located along a circle surrounding a square room and directed to its center. Objects (people) were modeled as vertical ellipsoids of known dimensions. The objects moved randomly around the room, while avoiding collisions.

The simulation was run for different number of cameras and different object sizes. Each simulation lasted the length of time it would take one object to travel ten times the length of the room. The number of objects was fixed in each run.

The maximum number of objects for which we get an exact object count is plotted in Figure 6 (i.e., upper and lower bounds converge). As expected, more objects can be counted as the number of cameras increases. But this eventually levels off when the scene becomes too crowded. When the people density becomes sufficiently high, there will be pairs of people that cannot be separated (and therefore distinguished) by any of the cameras. In the simulations, the lower bound converges faster than the upper bound. This is due to the fact that the LBC constraint is actually counting distinct real objects.

The counting algorithm can be run at very fast rates. For example, an Athlon 1900+ computer can process a scene of 20 objects at 200 fps.

## 4.2 Experiments with Real Data

In the real experiments, we also used 8 cameras to count (Figure 5). We allow people to enter and exit the workspace region through any point in its boundary. In relation to the synthetic experiment, people correspond to an object size between 0.11-0.14. With 8 cameras, 9-12 people could be counted in the ideal synthetic case, so this is the performance ceiling for the real case.

**Counting Experiments** The purpose of these experiments is to test the accuracy of the counting algorithm. We had several people enter, walk around, and exit the workspace. Figures 7 and 8 show two runs with 5 and 8 people, respectively. The lower bound and the actual count is plotted. The upper bound was very high and is not plotted. The lower bound matches the actual count very well. As explained earlier, the lower bound is tighter than the upper bound because of the nature of the LBC constraint. In addition, the UBC constraint is even weaker here because the actual object size is unknown and different for each person. The smallest minimum object size must be used, making the upper bound even bigger.

Also, allowing people to enter and exit weakens the bounds for polygons near the edge. To get a better lower bound for these edge polygons, when a polygon inside the workspace moves to the edge, the lower bound is not immediately set to zero (because people could have exited) as would be required to guarantee correctness. Instead, the lower bound is set to zero when the polygon disappears from the edge. The tradeoff is that when people walk along the edge, the lower bound is tighter, but when people exit, there will be a lag in the lower bound before it catches up to the actual count. This is why in the figures the lower bound lags behind the actual count when people exit.

The erroneous spikes in the lower bound plots are due

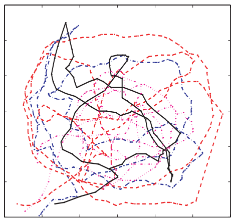


Figure 9: Actual paths of 4 people in the workspace.

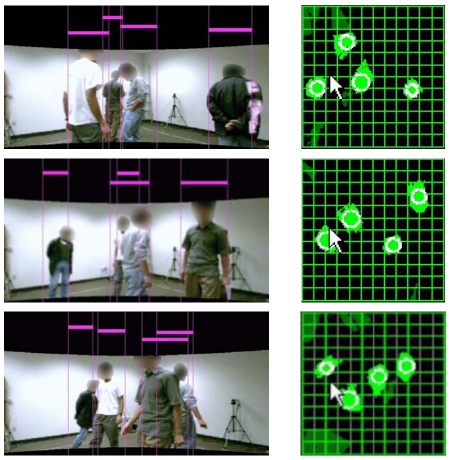


Figure 10: Left: Predicted locations in unused 8th view. Right: Corresponding pruned projected visual hull.

to noise. A noisy silhouette caused a person to be split and thus counted twice. This happened twice over 4000 frames and both times the algorithm quickly recovered. Also, in the run with 5 people, one of the cameras failed, so the whole run only used information from 7 of the cameras.

In these runs, the lower bound is almost always equal to the actual count. This is because the LBC constraint is very good and because people do not always move collusively. If people walk independently, even if their paths cross often, the lower bound should be tight and will tend to be a good approximation of the actual count.

**Localization Experiment** The purpose of this experiment is to test the accuracy of localizing people. We used 7 of the cameras to count and localize. The 8th camera view was not used in the computation and served as the ground truth. We projected the computed locations of the people into the unused 8th view and measured the accuracy by checking when the computed locations were centered on people. The data was generated by 4 people who entered, walked around, and exited the workspace over a run of 700 frames. Figure 9 shows the actual paths of the 4 people.

The video (881.mov) shows the predicted locations of people in the unused view. The vertical lines are the boundaries of the circles. The height of each horizontal bar represents the computed distance of each object from the camera (3 frames are shown in Figure 10).

People were correctly localized in the unused view 87 percent of the time, with 5 percent false positives. The false positives are caused by the lag of the lower bound when people exit because the lower bounds of edge polygons are not immediately decremented. These false positives can be eliminated by immediately decrementing the lower bound of edge polygons at the cost of increased false negatives because people who walk along the edge are missed.

For more examples and future work, visit: `http://xenon.stanford.edu/~dbyang/sensors.html`.

## 5 Discussion and Future Work

We developed a system that counts people in a crowded scene using a network of simple image sensors. We introduced a geometric algorithm that computes bounds on the number and possible locations of people using silhouettes computed by each sensor through background subtraction. The system requires no initialization and runs in real-time. Our system does not compute any feature correspondences across views. Thus, the computation cost increases linearly with the number of cameras. The result is a scalable and fault tolerant system —the effect of a camera failure is a moderate increase in the size of the projected visual hull because there is one less silhouette to intersect.

The current prototype uses centralized communication, which may be effective for a small local cluster of sensors. The next step is to implement a decentralized communication architecture which is appropriate for a much larger sensor network. This would further minimize traffic across the network, and improve robustness and scalability. In the decentralized approach, instead of the central node, local cluster heads would be elected to aggregate the data. Also, instead of sending the foreground bitmap, the sensors can just as easily send the 2D projection of this. For the setup in this paper, the projection would be a $144 \times 144$ bitmap, or a 2.6 kbyte packet uncompressed. Many of these sensors can easily communicate their data over a bandwidth limited network such as a 11 megabit wireless ad-hoc network.

In the larger network, the local leaders compute the aggregated 2D projection and pass either this or the count up the network. Non-overlapping sensors do not need to communicate with each other. Redundant nodes can be added because the decentralized network is scalable, making the system more robust. The larger network also allows us to count a much larger crowd in a much larger area, perhaps using the checkpoints strategy described in the introduction.

A drawback of the system is the sensitivity of silhouette intersection to noise. Noisy silhouettes that underestimate the size of objects may lead to an undercount. To prevent this, we set a low background-subtraction threshold to force silhouettes to become overestimates. This converges to the correct visual hull as the number of sensors increases. Another effect of noise is spurious cuts through silhouettes that are otherwise solid. We addressed this problem with persistent grid points on the projection plane. Values at these points decay exponentially once an object disappears.

Using a voting scheme among cameras may improve robustness at the cost of more conservative bounds. This may be advantageous in a larger sensor network. Another approach is to use un-thresholded data from the background subtraction, which can be interpreted as occupancy probability measures. This requires more communication bandwidth but allows the background thresholding to be pushed forward into the silhouette intersection stage, resulting in a more reliable visual hull.

In our experiments, we placed the cameras on the perimeter of the region of interest. This allows a small number of cameras to cover a relatively large region, assuming the region is convex. Our framework also allows for cameras inside the region, convex or not and over/above the region when feasible. The number of sensors and their placement affect the accuracy of the counting. The optimal sensor placement problem needs further exploration [2, 7].

## Acknowledgments

The authors wish to thank the DARPA Software for Distributed Robotics Program, the Stanford Networking Research Center, and Honda R&D Americas for their support.

## References

[1] Q. Cai, J.K. Aggarwal, "Automatic Tracking of Human Motion in Indoor Scenes Across Multiple Synchronized Video Streams," in *ICCV*, pp. 356-362, 1998.

[2] Xing Chen, "Designing Multi-Camera Tracking Systems for scalability and Efficient Resource Allocation," Ph.D. dissertation, Stanford University , June 2002.

[3] K.M. Cheung, T. Kanade, J. Bouguet, M. Holler, "A Real Time System for Robust 3D Voxel Reconstruction of Human Motions," in *CVPR*, v.2, pp. 714-720, 2000.

[4] R. Collins, A. Lipton, T. Kanade, "A System for Video Surveillance and Monitoring," *American Nuclear Soc. 8th Int. Topical Meeting on Robotics and Remote Systems*, 1999.

[5] T. Darrell, G. Gordon, M. Harville, J. Woodfill, "Integrated person tracking using stereo, color, and pattern detection," in *CVPR*, pp. 601-609, 1998.

[6] L. Doherty, B.A. Warneke, B.E. Boser, K. Pister, "Energy and Performance Considerations for Smart Dust," *Int. J. of Parallel Distributed Systems*, 2001.

[7] H.H. Gonzalez-Banos and J.C. Latombe, "A Randomized Art-Gallery Algorithm for Sensor Placement," *Proc. 17th ACM Symp. on Computational Geometry* (SoCG'01), pp. 232-240, 2001.

[8] I. Haritaoglu, D. Harwood, L.S. Davis, "W$^4$S: A Real-Time System for Detecting and Tracking People in 2 1/2 D," in *ECCV*, 1998.

[9] I. Haritaoglu, D. Harwood, L.S. Davis, "Hydra: Multiple People Detection and Tracking Using Silhouettes," *Int. Conf. on Image Analysis and Processing*, 1999.

[10] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, "System Architecture Directions for Networked Sensors," *ASPLOS*, 2000.

[11] T. Horprasert, D. Harwood, L.S. Davis, "A Robust Background Subtraction and Shadow Detection," in *Proc. Asian Conf. on Comp. Vision*, January 2000.

[12] T. Huang, S. Russell, "Object identification: a Bayesian analysis with application to traffic surveillance," *Artificial Intelligence*, 103:1-17, 1998.

[13] C. Intanagonwiwat, R. Govindan, D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," *MobiCom*, 2000.

[14] S. Intille, J.W. Davis, A. Bobick, "Real-Time Closed-World Tracking," in *CVPR*, pp. 697-703, 1997.

[15] M. Isard, J. MacCormick, "BraMBLe: A Bayesian Multiple-Blob Tracker," in *ICCV*, v. 2, pp. 34-41, 2001.

[16] V. Kettnaker, R. Zabih, "Counting People from Multiple Cameras," *ICMCS*, pp. 267-271, 1999.

[17] J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale, S. Shafer, "Multi-camera Multi-person Tracking for EasyLiving," *IEEE Inter. Workshop on Visual Surveillance*, 2000.

[18] S.H. Lim, A. El Gamal, "Integration of Image Capture and Processing – Beyond Single Chip Digital Camera," *Proc. of SPIE Electronic Imaging Conf.*, 2001.

[19] W. Matusik, C. Buehler, L. McMillan, "Polyhedral Visual Hulls for Real-Time Rendering," *Eurographics Workshop on Rendering*, 2001.

[20] W.M. Merrill, K. Sohrabi, L. Girod, J. Elson, F. Newberg, W. Kaiser, "Open Standard Development Platforms for Distributed Sensor Networks," *Proc. of SPIE, Unattended Ground Sensor Technologies and Applications IV*, 2002.

[21] A. Mittal, L.S. Davis, "M2Tracker: A Multi-View Approach to Segmenting and Tracking People in a Cluttered Scene Using Region-Based Stereo," in *ECCV*, 2002.

[22] J. Orwell, P. Remagnino, G.A. Jones, "Multi-Camera Color Tracking," *IEEE Workshop on Visual Surveillance*, 1999.

[23] G.J. Pottie, W.J. Kaiser, "Wireless integrated network sensors," *CACM*, v.43, n.5, pp. 51-58, 2000.

[24] T. Sogo, H. Ishiguro, M. Trivedi, "Real-Time Target Localization and Tracking by N-Ocular Stereo," *IEEE Workshop on Omnidirectional Vision*, pp. 153-160, 2000.

[25] R. Szeliski, "Rapid Octree Construction from Image Sequences," *CVGIP: Image Understanding*, v.58, n.1, pp. 23-32, 1993.

[26] T. Wada, X. Wu, S. Tokai, T. Matsuyama, "Homography Based Parallel Volume Intersection: Toward Real-Time Volume Reconstruction using Active Cameras," *IEEE Workshop on Comp. Arch. for Machine Perception*, pp. 331-339, 2000.

[27] T. Zhao, R. Nevatia, "Stochastic Human Segmentation from a Static Camera," *IEEE Workshop on Motion and Video Computing*, 2002.