Sensor Tasking for Occupancy Reasoning in a Network of Cameras

Danny B. Yang[1] Jaewon Shin[2] Ali Ozer Ercan[2]
Leonidas J. Guibas[1]
Computer Science Department[1] Electrical Engineering Department[2]
Stanford University
Stanford, CA 94305
{dbyang,jwshin,aliercan}@stanford.edu, guibas@cs.stanford.edu

Abstract

In this paper, we study how to task camera sensor nodes to reason about the occupancy of the area around them. Occupancy information is valuable because it can be used to answer many other queries such as determining object tracks or the count of the number of people in an area. Camera sensors are challenging to include in a wireless sensor network (WSN) because they are high data rate devices. To save energy and to satisfy the bandwidth constraint, our camera nodes will only send a very limited amount of data and only a limited number of camera nodes will be tasked. Our first result, from simulation, gives an upper bound on the number of cameras needed for a given accuracy in the occupancy. Given this number of cameras, we then compare several approaches to tasking the most relevant cameras both in simulation and in a real system of 16 camera nodes. Our incremental greedy tasking algorithm performed the best. Finally, we applied this tasking algorithm to a tracking application. We show that the tracker that used tasking outperformed the same tracker without tasking.

1 Introduction

Recent advances in CMOS fabrication have made it possible to integrate processing (and communication) circuits with the image sensor [7, 13]. With this integration capability and today's feature sizes, it is possible to produce image sensor nodes with processing and communication capability at very low cost. This type of camera node is an ideal candidate for a sensor node in a wireless sensor network (WSN).

The main limitation of WSNs is energy, which translates into constraints on communication and computation. For camera nodes, these constraints are particularly limiting because cameras are high data rate devices (as opposed to low data rate devices like microphones and thermometers) and the vision algorithms required to process the image data are often very computationally expensive. To overcome these constraints, it is critical to both have cheap local processing, and be able *to task useful subsets of cameras* to collaboratively answer queries.

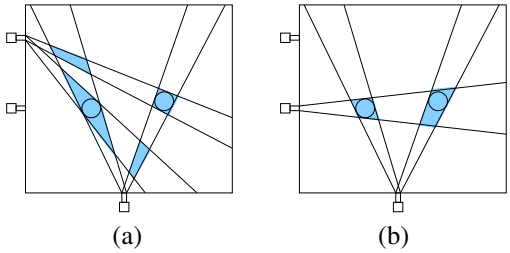


Figure 1. Comparison of 2 different camera taskings to localize 2 circular objects in the plane

Tasking camera nodes is particularly important in a WSN. A good tasking scheme will save energy, because less cameras are needed to answer a particular query. Tasking allows the network to scale to very large numbers of camera nodes. However, tasking cameras is more difficult than tasking many other types of sensors because cameras can see far away and consequently the data is nonlocal. Simply tasking local camera nodes may lead to poor results.

In this paper, we task camera nodes to reason about occupancy. Figure 1 illustrates two different tasking strategies of choosing two cameras out of three, and shows that intelligent tasking makes a huge difference in the quality of information that can be obtained from the collaboration. Tasking the two cameras in (b) yields better information than in (a) because the tasking in (b) is able to localize the two circular objects.

In addition to the information quality of the tasked cameras, we should also consider efficient updates. Since ob-

jects are moving around smoothly, after a small time step, a good subset of cameras is not likely to change much. This suggests that it might be possible to compute the new set of cameras by *updating* the previous set. This is also very important in terms of energy consumption.

We introduce efficient camera tasking heuristics that minimize *areas that are potentially occupied by moving objects* and compare their performance in simulation and in a real system of 16 cameras. We argue that the occupancy information is useful for many applications like tracking, counting and monitoring. The result shows that our heuristic is energy-efficient and selects good subsets of cameras. We also implement a simple tracking algorithm based on the occupancy information provided by our tasking heuristic and shows that the tracker based on our tasking outperforms the tracker with no tasking.

This paper will be organized as follows: In section 2, we will introduce the sensor tasking problem. In section 3 and 4, we will explain the number of cameras tasked and the heuristics for tasking the cameras. In section 5, we will give simulation and experimental results. A tracking application will be presented in section 6. In section 7, we will give a brief overview of related work and in section 8, we conclude and discuss future work.

2 Sensor Tasking Problem

2.1 Local Processing

In a WSN, each sensor node is operated under energy constraint. It is crucial to perform as much local processing as possible to reduce the amount of information that needs to be communicated to other nodes because the communication cost is much higher than the computation cost [12]. For image sensors, a light-weight local processing algorithm is also important, since many image processing and computer vision algorithms are extremely computationally expensive and can cause significant energy consumption at each node.

One of the simplest image processing algorithms is *background subtraction*, where a background image ($M \times N$ matrix) is subtracted from each new image to segment the object regions. We further compress the background subtracted image by summing the columns to get a 1-D scanline (N dimensional vector) that describes the foreground objects.[1] Figure 2 illustrates the background subtraction and 1-D scanline. A scanline is a very compact summarization, since only N numbers are needed, and this can be further compressed by run-length coding. This will result in big savings in the communication cost.

[1]To increase robustness, our actual implementation was more sophisticated. Please see section 5.2.2 for details.

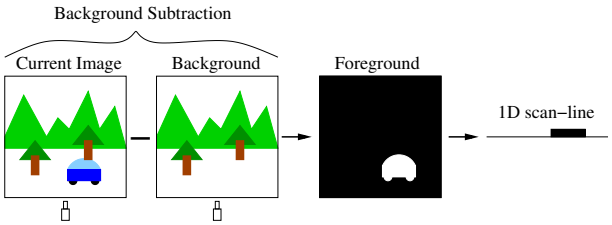


Figure 2. Background subtraction and 1-D scanline

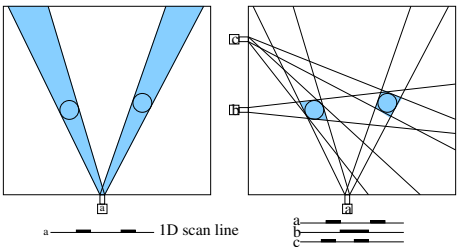


Figure 3. Visual hulls

2.2 Occupancy and Visual Hulls

Although a single scanline can be computed very efficiently and has a very compact representation, it provides little information, since it only tells whether a specific direction (cone) is occluded by one or more objects. However, by combining many 1-D scanlines we can reason about the regions that are *potentially* occupied by objects. The intersection of the occluded cones from the 1D scanlines is called *visual hulls*. This is the maximal area that could be occupied (figure 3).

Many applications like people counting, multi-object tracking, and group behavior monitoring depend on occupancy information, so it is critical to have an accurate visual hull that result in the tightest occupancy. If we do not have energy constraints, we can use all the cameras available to compute the best visual hull since the area monotonically decreases as the number of cameras increases. However, this is not a realistic scenario for a WSN because as the number of used cameras nodes increases, the network will quickly run out of energy. Therefore, it is important to be able to *task a subset of cameras nodes* to save energy.

2.3 Problem Formulation and Assumptions

Given M objects and N cameras, we want to task k ($< N$) cameras that minimize the visual hull area. For simplicity, we make the following assumptions. All the cameras are modeled by perspective projection and have a 180^o

field of view. The cameras are pointing toward the center of the room, so the only parameter is the position on the perimeter of the room. The objects are discs.

Problem 1: *For M randomly placed objects, what is the expected number of cameras needed to guarantee a visual hull area that is within a certain factor of the area actually occupied?*

For a given camera and object placement, we can compute the visual hull area, but we do not know how to express this area as a function of the camera and object positions that can be easily used to calculate the expected number of cameras needed. We solve this problem in the next section using a Monte-Carlo simulation.

Problem 2: *For M randomly placed objects and N cameras, what is the optimal camera placement for minimizing the visual hull area?*

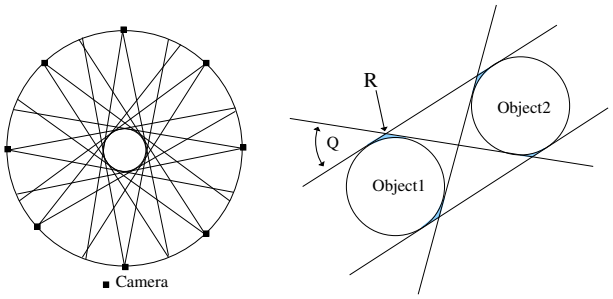
It makes sense that the optimal camera placement for randomly placed objects (with unknown positions) should be uniform, but we have not proved this yet. We will consider the following simpler problem of tasking a subset of positioned cameras.

Problem 3: *For M randomly placed objects and N placed cameras (known positions), what is the optimal subset of k cameras for minimizing the visual hull area? If the object positions are known, what is the optimal subset?*

For randomly placed objects, once again it makes sense that tasking a uniform (as evenly spaced and separated as possible) subset of cameras should be optimal. Given the object positions (or approximate positions from the previous timestep), the optimal tasking can be computed by evaluating all $\binom{N}{k}$ different camera taskings. However, each evaluation involves computing a visual hull, so this approach is too costly. Therefore, in section 4 we compare heuristics for finding good camera taskings quickly. There is also an additional consideration that as objects move, the new subset of cameras tasked should not be too different from the previous subset, so that cameras are not turned on and off too frequently. This will also be accounted for in designing our heuristics.

3 Number of Cameras for Tasking

In this section, we present an empirical solution to the problem of determining the number of cameras required to guarantee a visual hull area within a factor of the minimum. Specifically, we want to know, for a given number of objects in a room, how many cameras we need. To this end, we performed a Monte-Carlo simulation. We randomly dropped circular objects of radius 0.1 inside a circular room of radius 1. Objects were not allowed to overlap - overlapping objects were removed and dropped again. Cameras were placed uniformly around the perimeter of the room. For a given number of objects and a given number of cameras, the visual hull area was computed. For each setting, we repeated this 500 times and averaged, to compute the expected area of the visual hull.



(a) (b)

Figure 4. Residual error in the VH due to occlusion

An important note is that the visual hull area does not converge to the actual total object area as the number of cameras goes to infinity for 2 or more objects. This is because of occlusion. In the case of 1 object, there is no occlusion, so as the number of cameras increases, the circumscribing tangent lines of the visual hull close in on the object from all sides and the visual hull converges to the object (figure 4a.). In the case of 2 objects (figure 4b.), Object2 occludes all tangent lines in the region Q, resulting in the residual area R that overestimates Object1 even with an infinite number of cameras. For more than 2 cameras, in addition to these occlusion from pairs of objects, there are also occlusions from multiple (> 2) objects. Figure 5 shows the residual area of the visual hull, normalized by the area of an object, of 1 to 10 objects seen by 2000 cameras. As the number of objects increases, the residual area increases due to more occlusions. The 2000 camera case was used to approximate the case when the number of cameras goes to infinite.

The main result for this section is plotted in figure 6, which shows the accuracy of the visual hull for different numbers of objects and cameras. The residual of the visual hull is scaled by the "infinite" camera case for the corresponding number of objects, because that corresponds to the minimum possible visual hull area.

Observation 1 *For a single object, the residual of the expected visual hull area decreases as $O(\frac{1}{N^2})$ for N cameras. This is as expected because the residual of a regular circumscribed N-gon of a circle also decreases as $\frac{1}{N^2}$. For 2 or more objects, the residual decreases at a slower and slower rate due to an increasing number of occlusions.*

For an application that requires the visual hull area to be within a certain factor of the minimum, then the plot in fig-

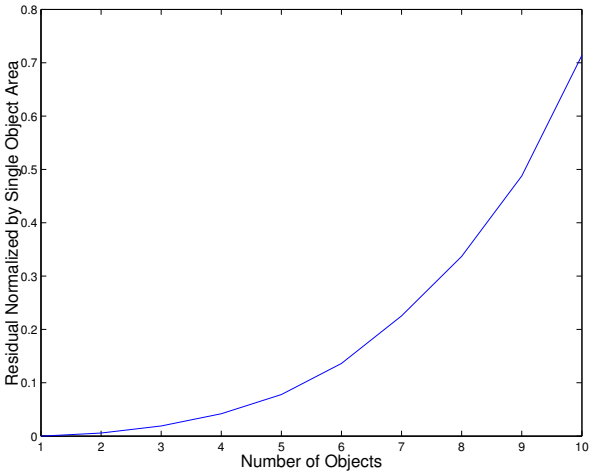


Figure 5. Residual of the expected VH with "infinite" number of cameras

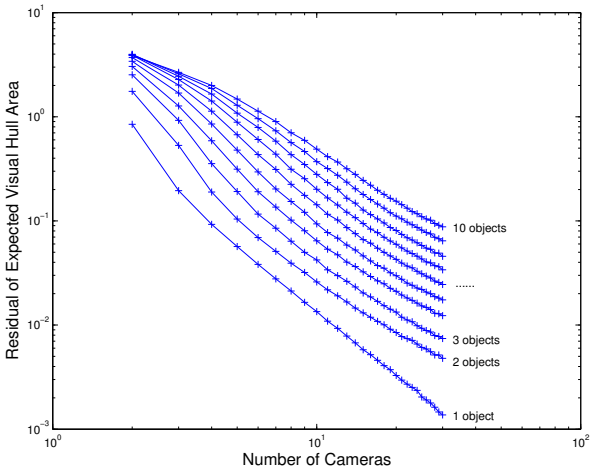


Figure 6. Residual of expected visual hull area as a function of the number of cameras

ure 6 can be used to decide the expected number of cameras needed. This result was for uniformly placed cameras, but for camera tasking heuristics that perform better than the uniform case, we will get a lower residual area, hence this result provides an empirical upper bound on the number of cameras (k) to task.

4 Camera Tasking Algorithms

Given the number of cameras to task (k), we want to task the most relevant ones. Assuming there are N cameras ($N > k$), we wish to find the subset of k cameras that will minimize the visual hull area. We tested the following approaches to tasking cameras:

4.1 Uniform

The simplest approach is to pick cameras to cover the space as evenly as possible. Given no other information, this is probably the best approach. Besides initialing by tasking the uniform set of cameras, this approach does no additional tasking.

4.2 Clustering

For 2 objects and 2 cameras, if v is the vector connecting the objects, and c_1 and c_2 are the unit vectors describing the direction of the cameras, we know that choosing $c_1 \parallel v$ and $c_2 \perp v$ gives a good placement because the resulting visual hull has no phantom areas (figure 7). Motivated by this, we generalize this idea to $N > 2$ objects and $k > 2$ cameras. Now, for every pair of objects, we try to maximize the $\parallel$ and $\perp$ placement for some camera. Specifically, given the object locations, we maximize:

$$\sum_{i<j} \left(\max_{1 \le l \le k} \left(|v_{ij}^T \cdot c_l| \right) + \max_{1 \le l \le k} \left(|v_{ij}^{\perp T} \cdot c_l| \right) \right) \qquad (1)$$

where v_{ij} is the vector connecting the i^{th} and j^{th} object. It turns out that the solution to this optimization problem can easily be approximated by k-means clustering [1]. We simply perform k-means clustering on the angles $\angle v_{ij}$ and $\angle v_{ij}^{\perp}, \forall i, j$, to get the k optimal camera directions. The cameras with the closest directions are then tasked.

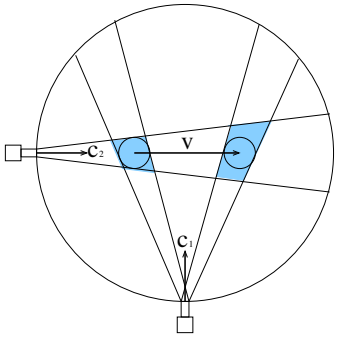


Figure 7. Good placement for 2 objects and 2 cameras

4.3 Optimal

Given the object and camera locations, the optimal approach computes the subset of cameras that gives the tightest occupancy . This requires the brute force search of all possible camera configurations, and returns the configuration with the smallest visual hull area. The cost is $O(\binom{N}{k})$.

4.4 Greedy

This approach involves greedily searching for the subset of cameras that gives the tightest occupancy. Instead of searching all possible subsets, it greedily picks the best camera one at a time. The following pseudo-code describes the algorithm in detail:

Algorithm Greedy Tasking
Select k of N cameras greedily to minimize the visual hull area
Output: a set S of the k selected cameras
1. S = empty set
2. for i = 1 to k
3. MinimumArea = infinite;
4. for Camera = 1 to N
5. if (Camera $\notin S$)
6. add Camera to S
7. A = visual hull area using cameras in S, given the object locations
8. if (A <MinimumArea)
9. MinimumArea = A
10. BestCam = Camera
endif
11. remove Camera from S
endif
end for Camera
12. add BestCam to S
end for i

Note that the the object locations are used in the algorithm. When this information is not available, an estimate is used based on the visual hull from the previous time step. The polygon centers of the previous visual hull are used as the object locations. The cost of greedy tasking is $O(kN)$.

4.5 Incremental Greedy

This assumes that a subset of k cameras has previously been tasked and the objects have not moved far, so only some of the cameras need to be updated. Instead of greedily picking all the cameras each time, only a subset m of the k cameras is greedily updated.

First, m cameras are greedily removed one at a time from the tasked subset. Next, m cameras are greedily added one at a time. Greedy remove and add operations are very similar to the full greedy algorithm described above. For removing greedily, the following lines need to be replaced:

2. for i = 1 to m
5. if (Camera $\in S$)
6. remove Camera from S
11. add Camera to S
12. remove BestCam from S

After removing the cameras, m new cameras added with the same algorithm as the full greedy algorithm, with only the for loop at line 2 running from 1 to m. The cost of this is $O(mN)$. For our experiments, we will use $m = 1$.

5 Simulations and Experiments

We tested the above mentioned heuristics in both simulation and in the real system.

5.1 Simulations

The simulations assume there are 5 circular objects in a room of radius 1. The objects have radius 0.1. The cameras are assumed to be placed along the perimeter of the room and have 180^o field of view. Perspective projection is assumed in constructing the visual hull of a given object and camera configuration. For all the simulations and experiments, the polygons in the visual hull that are too small to be real objects (smaller than the object area) are removed. Simulations for both static objects and moving objects are performed.

5.1.1 Static Objects

For these simulations, 5 objects are randomly dropped in the room. Then the cameras are placed according to the uniform, clustering, greedy, and optimal algorithms, using the actual object locations. Actual locations are used since objects are randomly dropped and there is no previous estimate of their positions.

Uniform placement requires no computation, and it is the best bet when there is no a priori information on the objects' locations. It is chosen as a reference to compare against the other algorithms. The residual area vs. number of cameras tasked are shown for these algorithms in figure 8. For a given number of cameras and tasking scheme, many random drops are simulated and averaged to generate the expected visual hull area.

Clustering does much worse than the greedy and optimal algorithms. The reason for this is because the greedy and optimal algorithms try to minimize the visual hull area directly, while the clustering algorithm tries to maximize the metric we had in equation 1. The hope was this would help minimize the visual hull area. While it still can do better than uniform placement for small number of tasked cameras, it is inferior compared to the greedy algorithm.

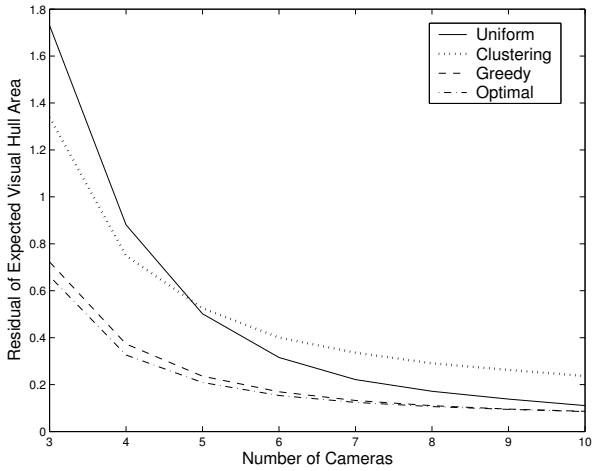


Figure 8. Performance of the tasking algorithms with static objects

Optimal placement performs the best, as expected, but is expensive to compute. Still, the greedy placement algorithm can achieve very close results to optimal. This algorithm and the incremental variation of it when there is object motion will be discussed in the next section.

5.1.2 Moving Objects

For these simulations, the objects are initialized with random drops, and assumed to be moving according to the random waypoints motion model. The objects each pick a random destination in the room and try to head there in a straight line at a constant velocity. If two objects block each other, each tries to get around the other by circling around to the right.

At each time step, the centers of the polygons from the visual hull of the previous time step are found. These polygon centers are then used by the tasking algorithms as the object centers for the current time step. The visual hull area is computed over a period of time, and a time-average of the residual visual hull area is computed for the different tasking algorithms and for different number of cameras.

Another heuristic that is used here, but not in the static objects case, is the incremental greedy algorithm (explained in section 4.5). It is less expensive computationally compared to the greedy algorithm and is suitable for moving objects because the scene does not change drastically in a time step.

The results for uniform, greedy and incremental greedy tasking algorithms are shown in figure 9. It is clear that both greedy tasking algorithms perform much better than uniform tasking. Incremental greedy placement achieves almost as good performance as greedily placing all cameras at each time step.

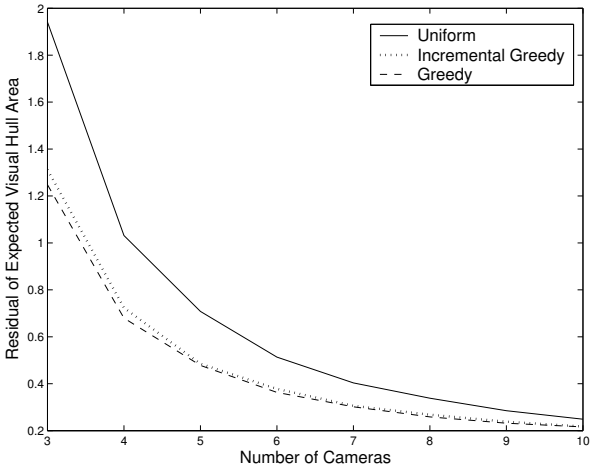


Figure 9. Performance of the tasking algorithms with moving objects

Incremental greedy algorithm is also tested with different update rates. That is, instead of incrementally updating the tasked cameras at each time step, we updated every 3^{rd}, 5^{th} and 7^{th} time step. The results of these simulations are shown in figure 10

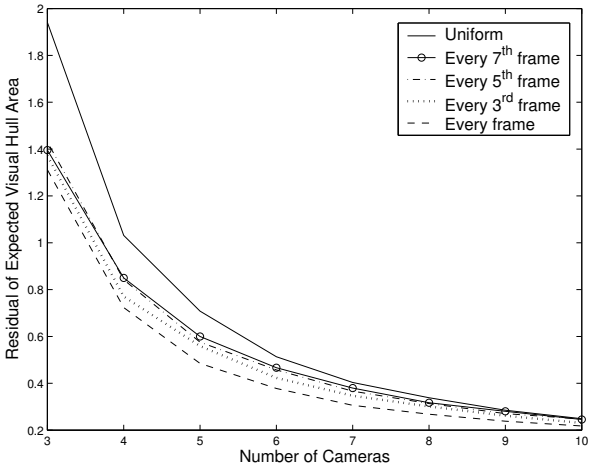


Figure 10. Performance of the incremental greedy algorithm with different update rates

As expected, the performance gets worse as update rates get slower, however, it could be argued that even with very slow update rates such as every 7^{th} time step, much better performance than uniform is achieved, and update rate

can be traded off with less computation and communication cost.

5.2 Real Experiments

We tested the above algorithms in an experimental setup with real networked cameras. The experimental setup is explained below.

5.2.1 Experimental Setup

The setup consists of 16 web cameras placed around a $22' \times 19'$ room. The horizontal FOV of the cameras is 49^o, and they all look inward. The relative positions of the cameras in the room can be seen in figure 11. A view of the room (from camera 5) shows a typical scene with three people in the FOV (figure 12).

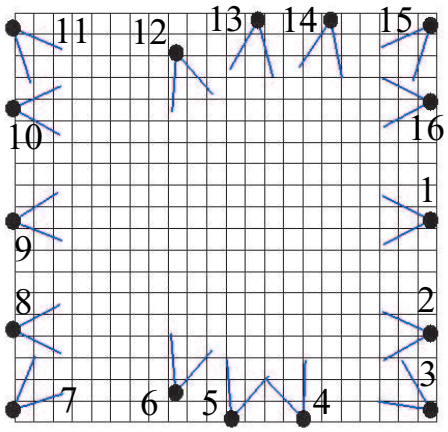


Figure 11. Positions of the cameras in the real setup. The cones show the FOV of the cameras and the grid spacing is 1 foot

The cameras are hooked up to a PC via IEEE 1394 (FireWire) interface and can provide 8-bit 3-channel (RGB) raw video at 7.5 frames/s. The PC connected to a camera models a sensor node in the network with processing power. Each PC is connected to 2 cameras, but the data from each camera is processed independently from the other. These PCs are then networked to a central PC, where further processing is performed. The networking is implemented in a wired medium, because the limited bandwidth is our only concern of WSNs and other concerns like collusion, fading, etc. are not topics of this research.

Raw data from each camera is gathered at the node PCs and are locally processed toward our application's needs. This will be explained in detail in section 5.2.2. When

Figure 12. View of part of the room seen by camera #5

tasked, the reduced data is then sent to the central processor, where the occupancy is computed. The PC's are all Intel Pentium based PC's running RedHat Linux 9.0

5.2.2 Local Processing

Each frame consists of 3 channels (RGB), and each channel is 640×480 pixels, 8 bits per pixel. The local processing first implements a simple version of the background subtraction algorithm given in [11]. This step mainly calls a pixel foreground, if its [R,G,B] color vector's angle is more than a threshold different from the background's corresponding pixel. This gives a binary indicator image of foreground. To suppress false positives, a morphological opening operation is also implemented. The resulting image is corrected for lens distortion. Then this image is summed vertically to generate the scanline. A spatial and temporal smoothing, and a median filter is applied to the scanline to further suppress false negatives, which would otherwise cause splits. The resulting scanline is just 640 bits, which is sent over to the central computer for final decision making. All of the above is done in real time at 7.5 frames/s.

5.2.3 Experimental Results

For this section, data was taken using the experimental setup with 3 people. Fewer people were used here because the cameras have a small field of view and the real data includes noise, thus making the problem more difficult.

Uniform tasking needed to be modified for the experimental setup because the room in our setup is not circular and the cameras have fixed positions. For 8 cameras, the odd numbered cameras are selected (See figure 11). For 6

cameras, cameras numbered 5 and 9 are removed from the 8 camera case. For 10 cameras, cameras numbered 8 and 16 are added to the 8 camera case.

The performance of the tasking algorithms can be seen in figure 13. The plots show the normalized residual visual hull area vs. number of cameras used. The normalization is done by dividing all visual hull areas by the minimum area that was achieved by all 16 cameras over all times. We do not know the actual object areas, so this is our best approximation. Time averages were taken over the whole run.

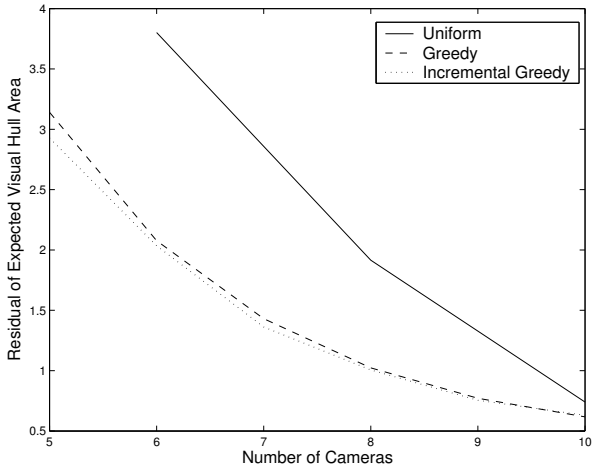


Figure 13. Performance of the tasking algorithms on the real system

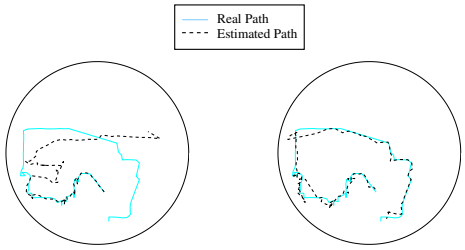
Again, the greedy algorithms performed better than uniform. For certain number of cameras, the incremental greedy does slightly better than the greedy algorithm, which is unexpected. These algorithms assume that the centers of the polygons from the previous time step are the real object centers, which is only a rough approximation. Several bad approximations probably caused the full greedy algorithm to perform worse. In the long run, we expect greedy to perform better than incremental. The important result is the incremental greedy performs well in the real setting.

6 Application: Tracking

The visual hull is a powerful primitive that can be used by many other applications as we stated before. Applications such as tracking people, detecting unusual events, and counting people in an area usually depend on having good occupancy information. Our camera tasking scheme precisely minimizes the maximal area that can be occupied. This maximum occupancy information can then be used by all those occupancy dependent applications. We will demonstrate the benefits of using our tasking scheme in one particular application: tracking.

Suppose we want to track moving objects in a room with our camera network. We can do this by first generating the visual hull of the moving objects. If there is enough data, the visual hull polygons will model the objects very well, and these polygons can be used to track the objects. For each frame, we compute the centers of the polygons and use temporal coherence to track the motions of these centers over multiple frames. These polygon center tracks will be the estimates of the actual object tracks in our tracking algorithm. Clearly, the tighter the visual hull is to the actual object areas, the more accurate this tracking algorithm will be.

We implemented this tracking algorithm on top of our tasking algorithm and compared the performance of the incremental greedy camera tasking case with a uniform camera case (no tasking). The simulation was set up with six circular objects of radius 0.1 moving in a circular room of radius 1 with the cameras arranged along the perimeter. For the uniform camera case, cameras were placed uniformly around the perimeter and all the cameras were used. For the incremental greedy case, a subset of 16 uniformly spaced cameras was tasked at each time step.



(a) 6 uniform cameras mean sqr error=0.370

(b) Tasking 4 cameras with incremental greedy mean sqr error=0.037

Figure 14. Comparison of the trackers showing the real and estimated tracks of one of the six objects

A comparison of the tracks of one of the six objects is shown in figure 14. The uniform case with 6 cameras is compared against the incremental greedy tasking case where only 4 cameras are tasked. The figure shows that with 6 cameras, the uniform case lost track of the object, while the tasking case did not. The tasking case used fewer cameras, thus saving energy because additional cameras can be turned off and saving networking bandwidth because data from less cameras are sent, and still outperformed the uniform case. With 7 cameras, the uniform tracker was able

to successfully track the object with a mean square error of 0.014, but tasking 7 cameras still outperforms the uniform case, with a mean square error of 0.009.

7 Related Work

Much work exists in the computer vision community on video surveillance and monitoring [4, 5, 9], of which the goal is larger-scale monitoring of activities over long time periods. Distributed resource tasking, however, is not the main concern of those works, since they use all the cameras for their tasks - tracking objects and detecting activities. One thing they have in common is that each camera sends summaries of events to a centralized computer, not the video stream itself. However the local processing may be computationally very expensive.

In robotics, occupancy maps are often computed for robot navigation tasks. Typically, various range sensors such as sonar or laser range finders are used. Image sensors have also been used. In [10], multiple cameras detect foreground to determine the occupancy in real-time. The drawback of this system is that a central computer collects the video stream from all the cameras to compute the occupancy.

In computer graphics, a related problem is the art-gallery problem [16]. The goal of this problem is to cover a given space using the minimum number of cameras. The setting, however, is mainly of theoretical interest and the results are not applicable to our context, since their definition of occlusion and utilities are different from our context.

Another related problem in robotics and graphics is the selection of the next best view. One scenario is to select the best views to cover the surface of an object [19]. This allows a 3d range finder to build the best 3d model with the least number of views. Another scenario is to pick the best views for scene understanding. [18] defines a viewpoint entropy that favors scenes with more "information." Projected areas of scene parts are converted into probabilities for the entropy computation. The common theme between the next best view work and ours is the definition of an objective function that is then maximized by searching efficiently over possible camera positions.

In WSNs, the resource tasking problem has been studied extensively. In [3, 20], the authors propose a utility-based framework that combines information utility of a sensor and the cost of tasking it in a single target tracking application. Many other recent works [2, 17] in WSN study problems of resource allocation and load balancing based on utilities and game theory, but their setting is theoretical and not applicable to a camera network. Not much work exists on using cameras in a WSN. [15] surveys the various issues with using cameras - balancing the computational cost of vision algorithms with the networking cost of cameras, which are high data rate devices.

The most similar work to ours is probably [14]. Here they try to task a subset of cameras to localize a few objects. The tasking problem is formulated as a constraint satisfaction problem and solved in a centralized setting. Their work, however, does not fully address the issues related to occlusions and is simply based on the simulation results from a four camera case.

8 Discussion

We have presented heuristics for reducing the visual hull areas both for static and moving objects. If an application were to use the visual hull information, like multi-object tracking or counting, then the visual hull area could be further reduced by pruning the phantom polygons that appear from nowhere (because real objects have temporal coherence). This additional processing can be fed back to the occupancy tasking algorithm to improve the visual hull. Also, additional low level local processing such as color histograms and motion vectors can be used to further reduce the visual hull area.

For the current experimental setup, the scanlines from the tasked nodes are sent to a fixed central node that aggregates the scanlines to compute the occupancy. We can further distribute the computation, by automatically selecting the aggregating node. One idea to elect cluster heads to task the nearby cameras and aggregate the information. Cluster heads can then be queried for the occupancy in a particular region. A distributed algorithm for cluster formation is described in [8] and can be used for this purpose.

For simplicity, we have so far only placed cameras along the perimeter of a simple convex shape (circle and rectangle). Our tasking heuristic, however, can be extended to general non-convex shape spaces (such as hallway with branching rooms) and to cameras placed inside the space. The only difference is that some of the visual hull polygons will be non-convex, which is not a problem because our algorithms do not depend on the shape of the polygons.

Some theoretical studies are necessary for better understanding of visual hulls and camera tasking. For example, in section 3, the residual of the expected visual hull area scales as $\frac{1}{N^2}$ for one object and it has nice theoretical explanation in [6]. For more than one object, however, we do not know how the residual scales except for the empirical observation we made. It would be nice to understand if the residual actually scales as $\frac{1}{N^\alpha}$ and, if so, determine the values of the exponent α for different conditions, to give a better estimate on the number of cameras for tasking. Another interesting theoretical study is to prove if the uniform camera placement is optimal in terms of the visual hull area when object positions are not known.

9 Acknowledgement

The authors wish to acknowledge support from the Stanford Networking Research Center, the Stanford Media-X Consortium, ONR MURI grant N0014-02-1-0720, and NSF grant CCR-0204486.

References

[1] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, England, 1995.

[2] J. Byers and G. Nasser. Utility-based decision making in wireless sensor networks. In *Proceedings of the 1st ACM International Symposium on Mobile Ad-hoc Networking and Computing*, pages 143–144, 2000.

[3] M. Chu, H. Haussecker, and F. Zhao. Scalable information-driven sensor query and routing for ad hoc heterogeneous sensor network. In *International Journal of High Performance Computing Applications*, 2002.

[4] I. Cohen and G. Medioni. Detecting and tracking moving objects in video from and airborne observer. In *Proc. DARPA Image Understanding Workshop*, pages 217–222, Monterey, November 1998.

[5] R. T. Collins, A. J. Lipton, T. Kanade, H. Fujiyoshi, D. Duggins, Y. Tsin, D. Tolliver, N. Enomoto, O. Hasegawa, P. Burt, and L. Wixon. A system for video surveillance and monitoring. Technical Report CMU-RI-TR-00-12, The Robotics Institute, Carnegie Mellon University, 2000.

[6] R. M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *Journal of Approx. Theory*, 10:227–236, 1974.

[7] A. E. Gamal, D. Yang, and B. Fowler. Pixel level processing – why, what and how? In *Proc. of the SPIE Electronic Imaging '99 conference*, volume 3650, 1999.

[8] J. Gao, L. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Discrete mobile centers. In *Proceedings of the 17th Annual Symposium on Computational Geometry*, pages 188–196, 2001.

[9] W. E. L. Grimson, C. Stauffer, R. Romano, and L. Lee. Using adaptive tracking to classify and monitor activities in a site. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 22–29, Santa Barbara, CA, June 1998.

[10] A. Hoover and B. Olsen. A real-time occupancy map from multiple video streams. In *Proc. International Conference on Robotics and Automation*, Detroit, Michigan, May 1999.

[11] T. Horprasert, D. Harwood, and L. S. Davis. A robust background subtraction and shadow detection. In *Proc. Asian Conference on Computer Vision*, January 2000.

[12] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: Mobile networking for "smart dust". In *Proc. 5th Annual International Conference on Mobile Computing and Networks (MobiCom 1999)*, pages 271–278, Seattle, WA, 1999.

[13] S. Lim and A. El-Gamal. Integrating image capture and processing – beyond single chip digital camera. In *Proceedings of the SPIE Electronic Imaging '2001*, volume 4306, San Jose, CA, January 2001.

[14] T. Matsui, H. Matsuo, and A. Iwata. Dynamic camera allocation method based on constraint satisfaction and cooperative search. In *Proc. 2nd International Conference on Software Engineering, Artificial Intelligence, Network and Parallel/Distributed Computing*, Nagoya, Japan, February 2001.

[15] K. Obraczka, R. Manduchi, and J. Garcia-Luna-Aveces. Managing the information flow in visual sensor networks. In *Symposium on Wireless Personal Multimedia Communications*, Honolulu, Hawaii, October 2002.

[16] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, New York, NY, USA, August 1987.

[17] N. Sadagopan and B. Krishnamachari. Decentralized utility-based design of sensor networks. In *The 2nd Workshop on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt '04)*, University of Cambridge, UK, March 2004.

[18] P. Vazquez, M. Feixas, M. Sbert, and W. Heidrich. Viewpoint selection using viewpoint entropy. In *Vision, Modeling, and Visualization*, 2001.

[19] L. Wong, C. Dumont, and M. Abidi. Next best view system in a 3-d object modeling task. In *Proc. IEEE Conference on Computational Intelligence in Robotics and Automation*, Monterey, California, November 1999.

[20] F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration for tracking applications. *IEEE Signal Processing Magazine*, 19(2):61–72, March 2002.