Abstract of "Supporting Complex Tasks in Visual Sensor Networks" by Jie Mao, Ph.D., Brown University, May 2012.

Visual sensor networks (VSN) are networks of smart cameras capable of local image processing and data communication. Unlike traditional camera-based surveillance network in which cameras stream all image data to a centralized server for processing, cameras in VSNs form a distributed system, performing information extraction and collaborating on application-specific tasks. This thesis studies how complex vision tasks can be integrated with system resource constraints such as computation capacity, battery power and bandwidth, in two different VSN contexts.

The first context is large-scale ad-hoc wireless smart cameras working on battery power which resembles the architecture of general wireless sensor networks. In this context we build geographic hash table based network protocols that are adapted to the nature of image sensors. These protocols decouple the event sensing from the camera location. Simulation results show that these protocols allow efficient distributed camera calibration and event-based constraint processing.

The second context is smaller-scale static wired smart cameras with constant power supplies which can be found in public spaces that need surveillance such as airports and casinos. We study the performance advantages of applying probabilistic fusion methods in cross-camera object tracking. Object tracking based on a single feature type can produce high error rates due to environmental and view changes. We present a probabilistic object matching framework which employs multiple object features and a decision mechanism to combine results from multiple features. The framework builds matching probability distributions for each feature algorithm based on empirical data and combines these historical results into an aggregated result. Our experimental studies on realistic data show that while there is no single feature algorithm works best all the time, our probabilistic integration method on multiple features can almost always achieve better object matching accuracy than the best individual feature algorithm.

Supporting Complex Tasks in Visual Sensor Networks

by

Jie Mao

B. E., University of Science and Technology of China, 1999

M. S., University of Rhode Island, 2005

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island

May 2012

This dissertation by Jie Mao is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____ _____
John Jannotti, Director

Recommended to the Graduate Council

Date _____ _____
Uğur Çetintemel, Reader

Date _____ _____
Rodrigo Fonseca, Reader

Approved by the Graduate Council

Date _____ _____
Peter M. Weber
Dean of the Graduate School

Vita

Jie Mao was born in Chengdu, China in 1975. He completed his undergraduate studies in Electronic Information Engineering and double majored in Law at the University of Science and Technology of China in 1999. Three years' experience as a software engineer after graduation inspired his interest in computer science. He returned to graduate school in 2003 at the University of Rhode Island and earned his M.S. degree in Computer Science in two years. Then he started his doctoral program in the Computer Science Department at Brown University in 2005. There he worked with his advisor John Jannotti on Visual Sensor Networks.

Acknowledgements

I would first like to thank my advisor, John Jannotti, who has guided me over these past years on my research with his wisdom, insight, and patience. He is always available to advise me on all research related issues. I own the deepest gratitude to all his time and ideas contributed to my research. This thesis would not have been possible without his consistent support and encouragement. I am also grateful to Uğur Çetintemel, who has shared lots his knowledge in my research. I appreciate his generous support on my research work and life in general. I would also like to thank Rodrigo Fonseca for serving on my thesis committee, and his kind assistance to my dissertation.

I thank the Computer Science faculty at Brown, especially Meinolf Sellmann and Amy Greenwald, and Hui Wang from the Applied Mathematics Department, for willing to discuss my research problems and giving me invaluable suggestions.

I would like to thank Gabriel Taubin for serving on my research comps, and his guidance on my research from the engineering perspective. I would also like to thank the past members of the Visual Sensor Network System research group, Yong Zhao, Mert Akdere, Onur Keskin, Dongbo Wang, Zhengyuan Zhao and Juexin Wang, for creating a nice and warm research environment.

I would like to thank my friends at Brown, Olya Ohrimenko, Serdar Kadioglu, Aparna Das, Layla Oesper, Marek Vondrak, Wenjin Zhou, Deqing Sun, Peng Guan, Eric Lim, Dafei Jin, Nicole Seah and Wayne Chou, for their support to myself and my research.

Lastly, I would like to thank my wife and my other family members for their endless and unconditional support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the past decade, *Visual Sensor Network* has emerged as an important class of sensor network which uses cameras to acquire image data in a distributed system. Example applications include surveillance [4], environmental monitoring, and smart homes [22]. These systems share the goal of extracting information from acquired image data to fulfill application specific vision tasks, such as object detection, object tracking, and other advanced signal processing algorithms.

The topic of visual sensor networks is an interdisciplinary research area. It incorporates techniques from embedded system, image processing, network communication and distributed systems. Recent research has been conducted largely in two directions. The first tries to put visual sensors onto existing wireless sensor networks in which sensor nodes have limited computation power, work on batteries, and communicate through low-bandwidth wireless channels. Due to the severe resource limitation, visual sensors used in this context are usually low in image resolution, image quality and frame rate [38, 41]. Researches in this context try to prolong the system lifetime while performing vision tasks. The lifespan of the sensor nodes are limited by their on-board battery power. When sensor nodes deplete their battery, they are unable to exchange information with the other nodes so the entire network may become unavailable. In wireless sensor networks, the dominant power usage are usually spent in radio transmissions. Limiting the bandwidth usage of the sensor nodes

is the main consideration for successful VSN applications in this arena.

The second research direction comes from the traditional camera-based surveillance network systems [48]. These systems are equipped with wired, high resolution, and high frame rate cameras, attaching to constant power supplies. Traditional camera-based surveillance networks gather image data into centralized servers where high-level vision tasks can be performed and usually supervised by human operators. Researches in this context attempt to reduce the network bandwidth usage and computation overhead in the centralized server by equipping cameras nodes with certain processing capabilities to allow in-network information processing. Computer vision technologies are used to automate the surveillance process and increase the surveillance efficiency. However, just as human operators can make mistakes, automated computer vision techniques often makes incorrect predictions on surveillance events. The limited processing capability on the camera nodes prevents the usage of complex vision techniques that may yield high prediction accuracy but at the expense of daunting processing power. This limits the VSNs to adopt generic simple vision techniques which often show high error rates.

In this thesis, we describe novel techniques to support three complex visual tasks in common VSN applications, namely, camera calibration, event detection and object tracking. First, many sensor network applications need a localization service which determines the location of sensor nodes, and allows applications to make geographically sensitive queries. VSNs must not only be localized, but also *calibrated*. Calibration goes beyond localization to include orientation and position information that is sufficiently fine-grained to allow fusion between overlapping camera views. Once the cameras are calibrated, the visual sensor networks can be used to detect application-specific events. However, VSNs are different from other types of sensor networks because of the nature of visual sensors. Common sensors found in traditional sensor networks, such as temperature sensors, humidity sensors, and so on, provide information about the environment in the vicinity of the sensors. Fusion on these data conflates events and sensors because they are often co-located. Visual sensors, on the other hand, are long-range sensors that collect information about distant objects/scenes.

An event detected by a camera can be far away from the camera's location, even beyond the camera node's communication range. In the wireless sensor networks context, we show the advantages of using a geographic hashing technique in distributed camera calibration and event detection.

Another unique characteristic of visual sensors is their large data volume. Most common sensors sample simple scalar data. Visual sensors produce data as sequential 2D images. The large volume of the image data provides much richer information about the environment. Therefore, while general sensor network tasks involve data collection and simple data fusion such as finding the min, max and average value of data readings, the information-rich data from visual sensors enables high-level analysis and reasoning in the VSNs. Typical tasks that are performed in VSNs include object detection/tracking, activity recognition, etc. These tasks often use algorithms that require image data from multiple camera nodes. Considering the large volume of image data, it is important to process the data in-network so that only the smaller sized preprocessed data are sent among camera nodes. In the camera surveillance network context, we investigate the benefits of using probabilistic fusion methods on multiple vision algorithms to boost the prediction performance in cross-camera object tracking.

1.1 Contributions

This thesis presents several novel techniques to support complex visual tasks such as camera calibration, event detection and object tracking.

First, to support camera calibration, this thesis presents *Lighthouse*, a distributed calibration protocol that allows wireless smart camera networks to obtain a unified coordinate system without manual configuration or specialized hardware beyond GPS. The Lighthouse technique uses stereo cameras to obtain robust 3D feature sets which are matched using incrementally built Geographic Hash Tables (GHTs). Lighthouse finds matches between cameras, even between distant cameras, without centralizing observations. Lighthouse also

contributes several advancements in the cooperative creation of GHTs, including bootstrapping, topology determination, and consistent hashing for topology changes. Simulations indicate that Lighthouse significantly outperforms simpler matching schemes at all feature densities, and approximates the centralized solution in all but the most feature-poor environments.

Second, for event detection, we argue that decoupling the event locations from the sensor locations allows programmers to specify their intent more directly, and better supports remote sensing devices such as cameras. This thesis presents *Distributed Constraint Processing (DCP)*, a decentralized, scalable event detection framework that allows for efficient in-network aggregation without coupling events and sensors. In our model, complex events are specified as aggregations of events in time or space, without regard to sensor locations or communication paths. We describe an SQL-style declarative language with spatio-temporal constraints between events that can be used to express complex events, then we show how these complex events can be assembled efficiently. The distributed event detection mechanism scales to very large networks, load balances work across sensors, and is fault tolerant to network partitions and node failure.

The final part of this thesis presents a distributed probabilistic object tracking framework. Visual sensor networks can be quite different in real deployment from simulations in that simulations often assume good features can be extracted from image data to allow reliable distributed computations. In real VSN applications, however, feature data are usually noisy and don't always produce credible results. We develop probabilistic techniques to perform cross-camera object tracking with multiple features. In our framework, individual feature algorithms need not make explicit binary matching decisions, nor are they required to understand and report their own matching confidences. The framework computes matching probability distributions for each feature algorithm based on empirical data and use these historical results to combine feature matches into aggregated results. We build the framework on an actual deployed camera surveillance network. Our experiments show that the probabilistic fusion mechanism outperforms the comparison fusion algorithms

and almost always outperforms the best individual feature algorithm, even though the best algorithm differs in various scenarios.

1.2 Outline

The rest of this thesis is structured as follows. We first provide background information in Chapter 2. Chapter 3 presents Lighthouse, the distributed camera calibration system. Chapter 4 talks about the decentralized, scalable event detection framework. In Chapter 5, we introduce the probabilistic object tracking framework. Related work is discussed in Chapter 6. Finally in Chapter 7, we present our final remarks and ideas for future work.

Chapter 2

Background

In this section, we talk about the research background that motivates our work in this thesis.

2.1 Distributed Camera Calibration

For most applications, sensor networks require localization, often through the use of special purpose hardware. Localization determines the location of sensor nodes, and allows geographic forwarding and location-aware queries. Smart camera networks must go a step further to be *calibrated* across cameras. The cameras in a calibrated network have been so precisely localized that shared views of the same object may be fused to create, for example, three-dimensional models or super-resolution views.

Camera calibration requires precise positions and orientations, beyond the limits of existing localization techniques. Even differential GPS or Cricket [37], each with accuracy in the centimeter range, would be unable to determine the orientation of a small camera sensor. Further, small errors in orientation may result in large absolute errors when estimating the position of distant objects.

Multi-camera geometric calibration is an active research topic [1, 46]. With this technique, the correspondences between camera images are detected and used to compute the

coordinate system transformations from world coordinates to camera coordinates. Current solutions are centralized, usually requiring factorization of very large matrices [29, 16]. The most common approach is based on *structure from motion* algorithms, in which the pose of all cameras and the location of feature points in 3D are simultaneously estimated. Smart camera networks, on the other hand, require a robust distributed solution based on collaborative algorithms. Furthermore, networks with dynamic nodes require incremental approaches.

To find coordinate system transformations between cameras in a distributed system, the correspondences between cameras have to be sent to the same processing node to allow computation. While in the centralized approach, all data are sent to the same server for processing, in a smart camera network, computation has to be distributed throughout the network to avoid hot spots. Traditional wired computer networks solve this problem by use hashing techniques to hash data of the same types to same computer IDs, and send different types of data to different computers to distribute the computation. However, sensor nodes in a wireless network rely on routing protocols to send data from one node to another. A localization service is required to route data to destination nodes by their IDs. This is impossible in VSNs before the network gets calibrated. In this case, an alternative hashing technique, the geographic hash table (GHT) [40], has been proposed to hash data types to geographic coordinates, and a routing protocol called GPSR [21] can be used to send data to sensor nodes that are closest to the hashed geographic coordinates without specifying their IDs. These hashing techniques have motivated our work of a distributed camera calibration technique, called Lighthouse, that allows smart camera networks to incrementally obtain a unified coordinate system.

2.2 Distributed Constraint Processing

Many wireless sensor network applications require the fusion of sensor readings from individual sensors into meaningful *events*. These events draw the attention of human operators,

activate actuators, or contribute to the construction of higher-level events. The events of interest may vary greatly based on different application requirements. Sometimes, a single sensor reading is significant to the application. In other cases, applications are concerned with *complex events* which are the aggregations of geographically and temporally related sensor data. In these applications, sensor data from several different sensor nodes sensed at different moments and places must be fused to create the application-specified events.

To aggregate geographically and temporally distributed sensor data, sensor nodes could send all readings to a single rendezvous where they could be aggregated into application-specific events. There are several obvious drawbacks with this approach. First, sending all sensor data to a single place requires complete connectivity, and creates communication congestion near the base station. Sensor nodes under heavy communication often suffer from rapid battery drain and break communication paths. Second, sensor data are often redundant for complex event detection. Sending all data indiscriminately wastes bandwidth and power and thus shortens the system lifetime. Existing data aggregation algorithms generally address these problems by aggregating sensor readings into complex events at join points while propagating toward a collection point. Unfortunately, this form of aggregation is greatly complicated by long range sensing. It is difficult to determine whether a subevent needs be propagated further up the tree when any given subtree might report an event from a distant location.

Beyond tree aggregation, some systems aggregate among sensor *neighborhoods* [34, 50]. These neighborhoods are based on sensor locations (*i.e.* nodes with 10m of a given sensor), or communication details (*i.e.* nodes within two communication hops of a given sensor). These definitions are sometimes called *data-centric* because they abstract away the details of node identity, and focus on the location of sensor readings.

However, truly data-centric applications will not specify their operations in terms of sensor locations and communication paths. Notions of locality, in space or time, are best tied to events, not sensors or pathways. This separation is critical to supporting long-range sensors (*i.e.* cameras), or complex events that may be deduced to occur at a locations far

from any single sensor (*i.e.* triangulated sounds detection).

For example, consider three successively more complex applications in a network of acoustic and visual sensors deployed in an urban area.

1. Detect gunfire. The application is interested in acoustic data matching the gunshot sound pattern. A single match indicates that gunfire is present. No data aggregation is necessary.

2. Locate gunfire. The arrival time of the sound of a gunshot at multiple sensor nodes must be compared in order to triangulate and locate the point of fire[42]. Triangulation requires the exchange of timing data and processing among several sensor nodes.

3. Locate suspects near gunfire. The location and time of gunfire must be compared to the locations of people detected by surveillance cameras. The locations of the gunfire may be arbitrarily far from the acoustic and visual sensors. Further, the microphones that detect the gunfire may not be co-located with the cameras that observe people. Coordination must occur with constraints expressed on the times and locations of events, not of sensors.

The third application has motivated our work to allow complex event detection based on event constraints rather than sensor-based neighborhoods. In this thesis, we present a complex event detection framework that uses geographic addressing to decouple event locations from sensor locations while allowing maximum flexibility in choosing aggregation nodes.

2.3 Probabilistic Object Tracking

Object tracking is a common application in visual sensor networks. The general procedure to track objects across cameras is as follows.

1. Detect foreground blobs in a sequence of images from a camera. This is usually done

by performing background subtraction on images and segmenting the foreground images into different objects. Basic background subtraction detects foreground objects as the difference between the current frame and an image of the scene's static background or the average/median of the previous frames. Foreground objects consist of pixels that differ from the background for a value larger than predefined thresholds. These basic methods suffer from the limitation that they don't provide an explicit way to choose the threshold and they cannot cope with multiple modal background distributions. Advanced background subtraction algorithms have been proposed to improve the accuracy of the background model, such as mixture of Gaussians, etc. These advanced background subtraction algorithms offer better accuracy at the expense of more computation resources.

2. Compute object descriptors. There are many ways to describe an object. Common object descriptors are positions, color histograms, edges, contours, and so on. These descriptors are easy to compute but can be quite different between cameras because of the change of view angles and distances to the cameras. Advanced object descriptors provide invariance in feature sub-space but can be expensive in computation or feature size. For example, SIFT features [27] provide scale-invariance in feature transformation but the feature extraction can hardly be done in real-time on camera images with moderate image size and frame rate.

3. Compare object descriptors from different cameras to match object across cameras. Different object descriptors have their own matching methods to measure the similarity between two object descriptors. Because the object descriptors are computed based on pixel blobs that are 2D projections of 3D objects, these object descriptors can be noisy and change between cameras. Object descriptor matchings often produce incorrect results.

Traditional object tracking systems gather all image data at a centralized server where the computation of all three steps are performed. VSNs should track objects in a distributed

fashion. Objects can be detected and their descriptors computed locally at each camera node. Cross-camera object matching are enabled by carefully exchanging object descriptors among camera nodes.

By the constrained nature of the network resources, distributed object tracking faces several challenges. First, image feature extraction and comparison are usually resource intensive. When the image quality, frame rate and the number of objects increase, the data volume in exchanging the object descriptors can exceed the available network bandwidth. In addition, the computation required to perform these vision algorithms can also exceed the computation capacity of the camera nodes. Second, vision algorithms are also error-prone. The matching of object descriptors can produce two types of errors: the same object detected as different objects; different objects detected as the same object.

Oftentimes, the object tracking systems have multiple choices of object features for cross camera object matching. The computation and bandwidth resources needed to generate and transmit different features vary greatly. Also, any single object feature may not work the best all time due to changes of environment and object appearances. The matching performance of any object feature also changes for different pairs of cameras due to varying view angles and distances from the objects to the cameras.

Since no single object feature works all the time for cross camera object matching, it is natural to think of using multiple object features together and hopefully at least some features will work at a given time. When color histograms are wrong, shape-based or gradient-based features may match objects correctly. Using multiple features together can avoid severe tracking errors when tracking condition changes. The fundamental problem becomes how to fuse matching informations from several features when there is disagreement. In this thesis, we present a probabilistic technique to fuse multiple feature matching results using historical match probability distributions. We build match probability tables that correlate historical match probabilities for various reported similarity levels from each feature matching algorithm. Then, for online matching, we fuse the values from the match probability tables to make a final match decision.

Chapter 3

Distributed Smart Camera Calibration

3.1 Pairwise Feature Matching

Distributed calibration requires that sensors find similar features in other cameras. Unfortunately, low-level two-dimensional features are very difficult to match between the images of uncalibrated cameras. Instead, we advocate smart cameras with two image sensors. Using two sensors with a known (short) baseline allows for local stereo reconstruction, producing 3D features from the individual 2D images. 3D features are more robust for matching across nodes because they are immune to differences in color and brightness sensitivity.

We have prototyped pairwise 3D feature matching using several *camera pods*. Each camera pod includes four rigidly mounted network cameras capable of small baseline feature matching and stereo reconstruction. Our experiments used two cameras in each pod. First, simple two-dimensional features (corners) were detected separately in the images of each camera. Next, correspondences between the features of the two images were determined. This task was greatly simplified by the short, known base-line between the images. From these correspondences, three-dimensional locations for the features were determined. Closer features exhibit greater parallax in the twin images. In a smart camera network, this work

would be accomplished locally in a dual-imaged smart camera.

3.1.1 Three-dimensional matching

Once each camera pod possessed a set of three-dimension features (points, really), we considered the task of matching those points between pairs of camera pods. In this prototype, all of the features of two pods were brought together, and RANSAC [23] was employed to find the transformation that brought the largest number of 3D points into correspondence.

In a large sensor network, it would be infeasible to share all points between all pairs of cameras. Section 3.2 describes how features can be detected without wholesale feature exchange.

3.1.2 Geometric Hashing

Lighthouse advocates a move away from sharing all low-level features detected by a cameras toward a strategy that shares a few, robust high-level features. A *robust* feature is one that can be recognized easily by various cameras, regardless of pose. Geometric hashing maps a complicated low-level feature set to a single, more robust feature or *category*.

For example, rather than sharing all 3D feature points, Lighthouse might select triples of three-dimensional points and their relative distances. Such a triple can be recognized regardless of camera pose.

The use of Scale Independent Feature Transforms [26] is a more powerful implementation of the same idea. Objects are reduced to (an unfortunately large) number of *SIFT keys*. These keys may be viewed as geometric hashes of the object in question, and are little affected by scale, rotation, or noise.

As we describe Lighthouse's operation, we assume the existence of some geometric hashing function that is capable of finding robust, high-level features, and reducing them to a form that can be used for matching. The effectiveness of these features is abstracted away by assuming that each match may confer some certainty that a particular transform is appropriate to bring two cameras into a common reference frame. Lighthouse may operate

with any threshold level of certainty required to complete a match.

3.2 Matching with GHTs

Using robust three-dimensional features reduces the problem of distributed calibration to the detection of non-empty set intersection among the features of all camera pairs. This is, if node A observes features $\{a_1, a_2, a_3...\}$, and B observes features $\{b_1, b_2, b_3...\}$ we must discover any a_i that is the same feature as some b_j. If so, nodes A and B should learn of the intersection in order to agree upon a shared reference frame. In order to control errors, it may be important for a given node to learn of many shared features with a set of cameras in a given reference frame. Multiple matches may be required to eliminate errors caused by, for example, the repetition of many similar features in real-world settings, such as the seats of a stadium.

As implied by the previous section, a simple way to find many shared features is to attempt pairwise matches between neighbor nodes. Many wireless protocols require periodic beacons in order to establish neighbor tables used during routing. Lighthouse augments these beacons with feature announcements. When a node hears of a feature that it has also observed, a match has been found. Of course, this technique might be extended to announce features over multiple hops. However, extending this technique to flood all features throughout the network would be infeasible for even moderately large networks. We compare Lighthouse to each of these techniques in the following section.

Lighthouse uses a Geographic Hash Table (GHT) to match common features at greater distances. A GHT, like the distributed hash tables of wired networking, allows cooperating nodes to store data at arbitrary nodes in a network, based on the hash of the data's key value. In a GHT, the hash function computes a geographic coordinate, and the data item is stored at the node nearest to that coordinate.

Finding matches in a smart camera network that has already formed a GHT is straightforward. A node first computes a *geometric* (not geographic) hash of its features. The

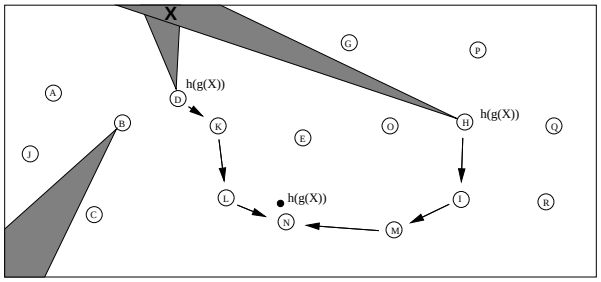


Figure 3.1: The feature X is observed by the two separate camera nodes. The feature is categorized through a geometric hash function, g(), and then a storage location is selected with a geographic hash, h(). Each camera routes the feature toward the designated location, where the closest node, N, stores the feature, detects matches, and informs the observers.

category is used as the key to insert the feature into the GHT. Two nodes with similar features will hash the feature to the same category, and then geographically hash the category to the same coordinate. The same node will therefore be responsible for storing both features, the "collision" may be noted, and the observing nodes notified. Figure 3.1 illustrates the matching process.

Unfortunately, GHTs rely on geographic forwarding which needs localization—which we intended to accomplish through feature matching. The goal of Lighthouse is to bootstrap the construction of ever larger GHTs using only the information gained during the construction of smaller GHTs, using data-directed calibration. The key insight is that any node within radio range of a GHT may use it for matching, even if the node in question is outside the GHTs coordinate space. Geographic forwarding is not needed for the first hop.

3.2.1 Singleton GHTs

We begin by considering the network at the beginning of a simultaneous startup. Each camera node is able to observe visible features, transmit in a local radio range, and listen for broadcasts from nearby nodes. We consider each of these nodes to be a *singleton* GHT with its own coordinate space. The GHT consists of one node, located at the origin and oriented in the direction of the camera's view. The known baseline between the node's image sensors allows a single camera to determine the scale of features in absolute terms.

Of course, a singleton GHT is a degenerate case. All inserts are stored at the single node, and no feature matches will be discovered.

3.2.2 GHT Maintenance

In order to support feature matching, a GHT should contain the features observed by each of its constituent nodes. Although the constituent nodes of a given GHT have already agreed upon a coordinate system, new feature matches among nodes of the GHT may allow the nodes to eliminate errors that might otherwise build up through pairwise matching. More importantly, we will soon see that these inserts are critical to allow merges with adjacent GHTs.

3.2.3 Merging GHTs

Adjacent GHTs are GHTs that contain nodes within radio range of one another. We call the set of nodes that are within radio range of a GHT, but are located within another GHT, the *neighbor set*. A neighbor node may constitute an entire GHT, as in the case of singletons, or simply a single member of a multi-node GHT.

Neighbor exchange Nodes from the neighbor set attempt to find matches between the adjacent GHTs by inserting features from their home GHT into the neighboring GHT. These features may have been directly observed by the neighbor, or they may have be stored at the neighbor by another member of the neighbor's GHT. In extreme cases, the neighbor may actively query its GHT to find additional features to share with the adjacent GHT. As an optimization, the proxy may respond immediately without inserting the feature if it contains a local feature match.

Proxy responses A neighbor's coordinate system is independent of the GHT into which it will insert. Therefore, insertions are passed through a proxy node inside the adjacent GHT. The proxy node performs the insertion, and forwards responses back to the neighbor. After the neighbor receives a response the neighbor may trigger a merge of the two GHTs

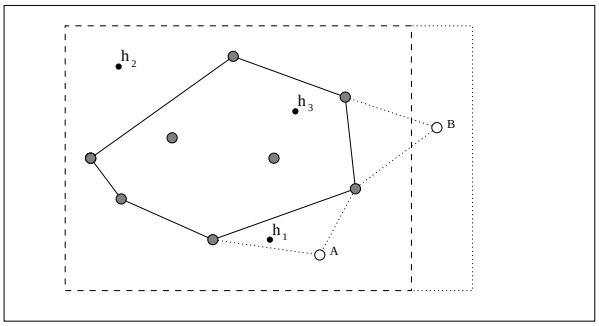


Figure 3.2: The group of gray nodes is a GHT, using the dashed box as its loose bounding box. To store an item, X, it is hashed repeatedly (h_1, h_2, h_3) until a locations is found within the perimeter. If node A joins the GHT, the loose bounding box need not be changed, therefore only those items, like X, that were placed in their current location after after skipping a hash location that is in the new perimeter. If node B joins the GHT, the loose bounding box must be expanded, requiring all items to be rehashed.

by broadcasting the new coordinate system to both GHTs. This decision might be triggered only after a threshold of matches has been met.

Consistent hashing GHTs were proposed for sensornets of static extent. As such, the range of the geographic hash function is predetermined by the geographic range of the sensornet. In a dynamic GHT, the size of the sensornet changes, and so the range must vary as well. The range should not exceed the true size of the sensornet by too much, or data items will be concentrated at the edges. The range should not be too small, or data items will be unduly concentrated in a few nodes.

To solve these problems, an appropriate range should be chosen for any GHT. One such range is the convex hull of the nodes in the GHT, though hashing to this irregular shape is not straightforward. In addition, because the topology of a dynamic GHT changes with time, it is important to develop a *consistent* [20] hash function that leaves most data items in the same location in response to small topology changes.

We advocate a two-phased hashing strategy, illustrated in Figure 3.2, that uses a loose bounding box, a family of hash functions, and knowledge of the true boundary of the GHT. To determine the location for a data item, it is hashed into the loose bounding box using the first member of the hash-family. Using a local polygon inclusion test, if the location is *also*

inside the true boundary of the GHT, the item is routed to the node closest to the hashed location. If not, successive members of the hash-family are used until an agreed upon limit is reached. If the limit is reached, the item is stored at the node closest to the first hashed location. A looser bounding box leads to addition computational effort to determine the appropriate hashed location for storage, but will lead to fewer complete rehashings which must occur when the bounding box is changed.

3.3 Evaluations

We conduct NS [35] simulations using a simple implementation of a GHT using code from GPSR [21]. Experiments are run in a 250m square with 100 randomly placed and oriented cameras, each with a radio range of 40m. Features are randomly placed in or near the 250m square where they may be detected by cameras if the camera is within 125m, and oriented properly. Cameras are assumed to have a 30° viewing angle. The number of features is varied to measure the effect of feature density.

3.3.1 Convergence

The first metric by which to measure a calibration technique is its ability to find matches and allow for the convergence of nodes into a shared coordinate space. We compare Lighthouse against three other strategies. The first two strategies are simple short range advertisement schemes. In the 1-hop scheme, each camera broadcasts the set of features it observes to all cameras within radio range. In the 2-hop scheme, features are rebroadcast by any camera that hears them from the direct observer.

Figure 3.3 shows that Lighthouse improves upon the performance of each of these schemes, allowing the 100 cameras to converge into approximately half the number of independent coordinate systems as the 2-hop scheme. The final scheme is an impractical flooding protocol that propagates all features to all reachable nodes. It shows the absolute minimum number of isolated groups that exist when all cameras are aware of the features

of all other cameras. For example, when only 100 features are detected throughout the network there are approximately 14 sets of camera that share no features in common.

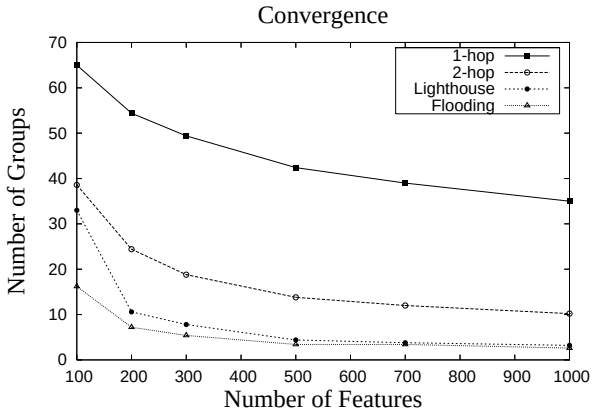


Figure 3.3: Lighthouse is compared to simpler 1-hop and 2-hop neighbor schemes, as well as a perfect matching scheme that floods all features to all nodes. As the feature density increases, all schemes are able to reduce the number of independent GHTs by finding shared features. At all densities Lighthouse performs significantly better than the short-range schemes. At reasonable feature densities, Lighthouse approximates complete flooding.

In should be noted that the 2-hop and flooding schemes are not viable schemes for simultaneous calibration and GHT construction. Both schemes assume that two cameras may merge into a single GHT, even if the nodes between them cannot. In such a case, the merged nodes would not be able to use their coordinate system for the geographic forwarding required to implement a GHT.

3.3.2 Scalability

In the last section we examined Lighthouse's ability to find matches, we now consider the cost of doing so. Figure 3.4 shows the amount of bandwidth used to disseminate features for matching. The graph underestimate the cost, in absolute terms, because our simulation uses very compact representations of features (integers). In reality, features are likely to be considerably larger, but the relative effect should be similar in each case.

Thinking about the costs asymptotically, 1-hop emits a message from each node, and receives features from d adjacent node. In 2-Hop, each node emits a feature, and the d

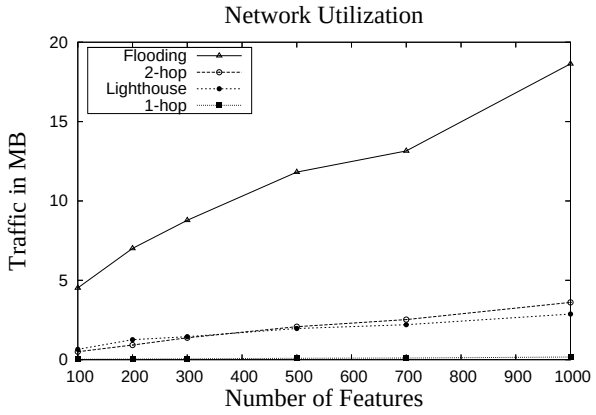


Figure 3.4: As feature density increases, more features are exchanged and more bandwidth is consumed. Lighthouse is able to grow as slowly as 2-hop, despite its ability to find wide-spread matches.

nodes that hear it re-emit it. The $O(d^2)$ nodes within two hops share their features. In Lighthouse, each node must store its feature in the GHT. A single GHT insert requires $O(\sqrt{n})$ transmissions to cross the sensornet. Flooding requires that each node emit its feature and that the sensor field flood it ($O(n)$).

Chapter 4

Distributed Event Processing

In this section, we describe the concept of *complex events* and describe how complex events are detected in the Distributed Constraint Processing (DCP) framework. Complex events are formed in a hierarchical way from simpler events with constraints. DCP detects events in a decentralized manner, avoiding global collection trees, and balancing the computational and network load across participating nodes. We build our constraint matching on top of geographic hash tables because they are a natural fit for our needs: they use geographic, rather than node-based addressing and they provide a matching mechanism that is scalable and fault-tolerant. We extend GHTs to provide a local matching service.

4.1 Complex and Primitive Events

Events are defined as occurrences of interest in a system. A person in a room, high temperature in an area, or the theft of a book might all be events in different applications. However, unlike a single sensor observation, a "book theft" event requires many observations and computation over those events. Accordingly, we divide events into two categories: primitive events and complex events.

Raw sensor readings are primitive events. A primitive event consists of the reading itself annotated with metadata, such as a time and location.

Complex Events are derived from simpler events. They are produced by *Event Processors* or *Event Detectors*, rather than individual sensors. An event processor creates complex events when it observes the appropriate constituent events. For example, a fire detector may require the observation of high temperature and smoke sensor readings. Further, the detector computes a fire event only if high temperature and smoke events occur in close proximity, in time and space.

In DCP, there are multiple event processors for every event type. Event processors of the same type are distributed throughout the network using Regional GHTs to facilitate distributed event aggregations. The efficient evaluation of constraints to produce complex events will be discussed in detail in Section 4.

4.2 Event Specification Language

Our event specification language borrows event operators from active database research where event operators were used in specifying triggers in database systems. We have also incorporated windowing constructs from stream processing and complex event processing research.

4.2.1 Sensor Specification

All events are derived in some way from raw sensor readings. The output of each sensor is declared in order to reference their raw readings in derived events. The sensor specification conforms to the following template:

sensor *name*
schema *attr_list*
attr_list → *attr* | *attr_list, attr*
attr → *attr_type name*
attr_type → *double* | *int* | *string*

Figure 4.1: A sensor is given a name, and any number of named and typed attributes. These attributes are referenced to create *events*.

Here are two example sensor specifications:

sensor *temperature* ***schema*** *double temp*

sensor *barometer* ***schema*** *double pressure*

We also assume that a pseudo-sensor named *node* exists in all sensor platforms. *Node* provides the spatial and temporal context information used during the construction of both complex and primitive event. It has the following specification:

sensor *node*
schema *string node_id,*
double[2] loc,
double time

Figure 4.2: Node specification template

node_id is the unique id of the device the sensor is located on. *loc* specifies the platform's location and the *time* attribute is used to access the platform clock. We assume that sensors are sufficiently synchronized, in time and space, to use these values in calculating constraint matches.

4.2.2 Base Event Schema

Complex and primitive events are both represented as attribute collections. All event of the same type have the same set of attributes which is called the event schema. The schema for each event type is specified in the event type declaration.

Some attributes, such as timestamp and location, are required in both complex and primitive events. They constitute the *base event schema*. The base schema includes the attributes *event_id*, *loc*, *start_time*, *end_time* and *node_id*. The base event schema facilitates the use of standardized event operators that evaluate common spatio-temporal relationships.

event_id is the identifier that identifies an instance of an event type. This identifier can be made unique by generating a fresh identifier for each complex event instantiation, or it can be created based on a subset of the attributes of an event instance. In the latter case, logically duplicate event instances will have the same identifier and may be suppressed during later processing. The *loc* attribute stores the location assigned to the event instance.

start_time and *end_time* represent the occurrence interval of the event. Finally, *node_id* identifies the node that generates the event instance.

4.2.3 Primitive Event Declaration

Primitive event declarations specify the transformation of sensor readings into primitive events. A primitive event can be regarded as a sensor reading annotated with metadata information. Primitive event declarations are made using the template in Figure 4.3.

primitive *name*
on *sensor_list*
schema *base_schema, attribute_list*

Figure 4.3: A primitive event is created by combining attributes from one or more sensors. The pseudosensor "node" is often used to provide the time and location required by the base event schema.

The *name* symbol stands for the name assigned to the primitive event type such as *person_detected*, or *barometer_reading*. *Sensor_list* contains the sensors the primitive event is defined upon. It may contain multiple sensors, but they must be located on the same node. Sensor fusion across nodes is described by complex events. Finally *schema* specifies the attributes of this primitive event type and the way they are assigned values. Here we provide an example primitive event specification for a temperature reading event common in many sensor network scenarios.

primitive *temp*
on *temperature, node*
schema *event_id as hash(node.node_id,*
node.node_time),
loc as node.loc,
start_time as node.time,
end_time as node.time,
temp as temperature.temp

Figure 4.4: The *temp* primitive event consists of the temperature reading from the sensor named *temperature* along with time and location information to populate the base event scheme from the *node* pseudosensor.

4.2.4 Complex Event Declaration

Complex events are combinations of simpler events, each of which may be primitive or complex. For most applications, users are interested in specifying complex events which impose spatial, temporal or attribute-based constraints on their subevents. We take a SQL-like approach to complex event specification and extend it with spatial/temporal constructs such as time windows to support these constraints. Our complex event specification template is given in Figure 4.5.

complex *name*
on *source_list*
schema *base_schema, attribute_list*
where *constraint_list*

Figure 4.5: Complex event declaration template

Every complex event type is assigned a unique name with the *name* attribute. The *source_list* is used to specify the subevents of a complex event type. The source list may also contain the *node* pseudo-sensor. As in primitive event specifications, *schema* specifies the attributes of the complex event type and also defines the transformation from subevents and their attributes into the attributes of the complex event. The *constraint_list* in the *where* clause specifies a logical expression on the subevents that must be fulfilled to construct the complex event. Constraints can be defined over subevent attributes, and can specify temporal or spatial patterns over subevents. We provide event operators for easy specification of constraints over subevents. Existential constraints are also available through subqueries. These features are described in Section 4.2.5.

As a simple example of a complex event consider the high temperature event. We define the high temperature complex event using the previously defined *temp* primitive event as follows:

```
complex hitemp
     on temp T, node
 schema event_id as hash(node.node_id,
                         node.node_time),
        loc as T.loc,
        start_time as T.start_time,
        end_time as T.end_.time,
        temp as T.temp
  where T.temp > 70
```

Figure 4.6: A hitemp event is constructed from a single subevent when a temp event, T, meets the constraint: T.temp $>$ 70.

4.2.5 Constraint Specification

Temporal, spatial, attribute-based and existential constraints can be specified in the where clause of a complex event specification. Each constraint returns a boolean result. For easy specification of event constraints we provide event operators, as introduced in the event languages developed in active database research. We have borrowed the event operators *and*, *or*, and *sequence* from existing work in that area [6, 13, 36]. All event operators are n-ary operators. The last argument of each event operator is the time window argument, w, which specifies the maximum time between any two subevents of the complex event. Subevents which are separated by more than w time units cannot be part of the same complex event instance. When an event operator produces output on a given set of subevents we say the corresponding event constraint is satisfied.

We also provide the SQL construct *exists (subquery)* for the specification of existential constraints. The result of the *exists* clause is true if the subquery returns any result. An example event specification to detect unattended luggage is given in Figure 4.7.

The unattended bag complex event is an example for a security monitoring scenario. We assume that there is a detector for bags already implemented and we can access a bag detection event through BagDetector. The event specification is made such that a bag is considered unattended when no person is detected within 5 meters of the bag for 60 seconds.

```
complex unattended_bag
     on BagDetector B, node
 schema event_id as hash(node.node_id,
                node.node_time, B.bagid),
        loc as B.loc,
        start_time as B.start_time,
        end_time as B.end_.time,
        bagid as B.bagid
  where not exists ( select * from person_detected P
        where and(P,B;60) and distance(P.loc, B.loc) < 5 )
```

Figure 4.7: An *unattended_bag* event when a bag is detected, but no person is detected within 5 meters for one minute. The base event schema is populated from the base schema in the *BagDetector* event.

4.3 Example Application

Consider an example object tracking application using calibrated stereo cameras. Stereo cameras can localize the 3D positions of the objects in their frustums, and can identify different objects using techniques, such as histogram comparison [5]. Such a camera network can be used to monitor behavior of people and to detect abnormal activities in an area. Here, we present an example scenario where the monitored event is a *person chasing another person*. We use our event specification language to declare the events involved in the application.

In order to detect complex events, we break them down into simpler, lower-level events and repeat this process until all events are primitive. For our example chase scenario, this process is illustrated in Figure 4.8. We can think of the *people_chasing* complex event as two people running close to each other (*e.g.* 10 meters) for a certain amount of time (*e.g.* 5 seconds). Below is the specification for the *people_chasing* complex event based on this idea.

people_chasing complex event is defined using the *running_group* complex event. *running_group* complex event detects two people running in close proximity. Its specification is given below.

The *running_group* complex event depends on the complex event *running_person*. The

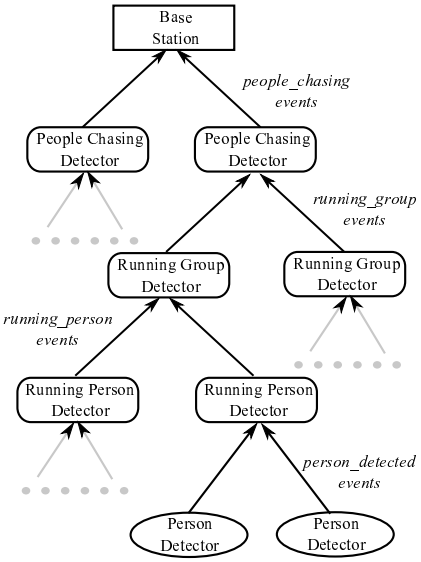


Figure 4.8: The process to detect the *people_chasing* event is decomposed into the following steps: 1. Detect a person using the person detector on the stereo camera sensor; output a *person_detected* event; 2. Detect a running person by calculating the person's moving speed using two consecutive *person_detected* events of the same person; output a *running_person* event; 3. Detect people running together by calculating the distance between two different running people; 4. Detect people chasing each other by looking for two people keeping running closely for a period of time, examining continuous *running_group* events with same person IDs; send *people_chasing* events back to the base station.

```
complex people_chasing
     on running_group as G1, running_group as G2,
        node
 schema event_id as hash(node.node_id, node.time,
        G1.person1_id, G1.person2_id),
        loc as avg(G1.loc, G2.loc),
        start_time as G1.start_time,
        end_time as G2.end_time,
        node_id as node.node_id,
        person1_id as G1.person1_id,
        person2_id as G1.person2_id,
  where seq(G1, G2; SRC_PERIOD_RG) and
        G1.person2_id = G2.person2_id and
        G1.person1_id = G2.person1_id and
        distance(G1.loc, G2.loc) <= CHASING_DIST
```

Figure 4.9: People-Chasing event specification

```
complex running_group
     on running_person as R1, running_person as R2,
        node
 schema event_id as hash(node.node_id, node.time,
        R1.person_id, R2.person_id),
        loc as avg(R1.loc, R2.loc),
        start_time as min(R1.start_time, R2.start_time)
        end_time as max(R1.end_time, R2.end_time),
        node_id as node.node_id,
        person1_id as R1.person_id,
        person2_id as R2.person_id,
  where and(R1, R2; SRC_PERIOD_PR) and
        R1.person_id != R2.person_id and
        distance(R1.loc, R2.loc) <= GROUP_DIST
```

Figure 4.10: Running-Group event specification

running_person complex event, which is used to find a running person, can be detected by computing a person's moving speed and comparing it to a threshold speed. This involves the comparison of two *person_detected* events of a person with different location and timestamps. The specification of the *running_person* complex event is given below.

```
complex running_person
     on person_detected as P1, person_detected as P2,
        node
 schema event_id as hash(node.node_id, node.time,
        P1.person_id),
        loc as P2.loc,
        start_time as P1.start_time,
        end_time as P2.end_time,
        node_id as node.node_id,
        person_id as P1.person_id,
        speed as distance(P1.loc, P2.loc)
        /(P2.end_time-P1.end_time)
  where seq(P1, P2; SRC_PERIOD_PD) and
        P1.person_id = P2.person_id and
        distance(P1.loc, P2.loc) <= RUNNING_DIST and
        distance(P1.loc, P2.loc)
        /(P2.end_time-P1.end_time)
        > SOURCE_PERIOD_PD*MAX_SPEED
```

Figure 4.11: Running-Person event specification

These specifications are naturally expressed with both spatial and temporal constraints that limit the distance and interval between subevents. People can only run so fast, so a spatio-temporal constraint prevents spurious matches from distant, unrelated events. Furthermore, these constraints allow DCP to operate efficiently, disseminating subevents only far enough to meet other relevant events. Without such constraints, a global event detection process would have to occur which would reduce the performance of the system.

Finally, the *person_detected* events can be generated by the *person_detector* on each sensor node, which constantly analyzes the stereo images taken by the stereo camera.

```
 sensor  person_detector
 schema  int person_id,
         double[2] loc
primitive person_detected
       on person_detector as PD, node
  schema event_id as hash(node.node_id, node.time,
              PD.person_id),
         loc as PD.loc,
         start_time as node.time,
         end_time as node.time,
         node_id as node.node_id,
         person_id as PD.person_id
```

Figure 4.12: Person-Detected event specification

Although complex events are decomposed in a top-down manner, DCP uses a proactive approach for event processing. Events are constantly generated by lower-level event processors and pushed into higher-level event processors. Whenever an event processor produces an event, it looks up the system configuration to find high-level event processors that operate on this type of subevent, then sends the event to their location. In such a way, events of all complexities can be detected with low delay.

In the following sections, we will use this example application to illustrate how our constraint processing framework works and evaluate its performance with simulation (See section 4.10).

4.4 Regional GHTs

In applications that detect events in a spatial area covered with wireless sensors, we expect that most events contain regional or temporal constraints because they are triggered by related phenomena, perhaps detected by various nearby sensor types. In our example application, people chasing may only be considered a suspicious behavior when it happens in a certain high security area, and people will only be considered to be chasing if they are running in close proximity. DCP leverages these constraints to obtain significant performance improvements without compromising correctness.

Data-Centric Storage [39] introduced the Geographic Hash Table (GHT) for wireless sensor networks. GHTs hash keys into geographic coordinates within the network topology, and store key-value pairs at the sensor node geographically closest to the hashed location. The canonical form of the GHT hash function is *coordinates = hash(key)*.

To preserve spatial locality of the events, we extend the canonical GHT hash function to create *Regional GHTs* which take `keys` and `regions` into account during hashing. The `region` defines the boundary of a geographic area. The extended regional hash function returns coordinates within the specified region. The regional hash function is *coordinates = hash_r(key, region)*.

Figure 4.13 shows the difference between a normal GHT and Regional GHT. Note that the event is stored much closer to its original location in a Regional GHT. In a Regional GHT, lookups must specify a matching region to find a particular event.

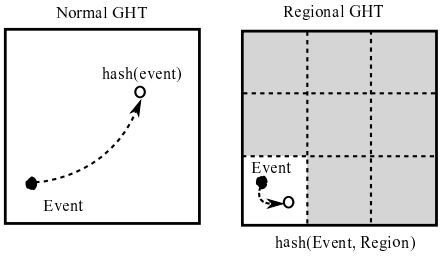


Figure 4.13: Normal GHT hashes events to global coordinates. Regional GHT hashes events to coordinates within a given region which is the lower left tile.

4.5 Hierarchical Event Processing

To use Regional GHT for spatially constrained event detection, we first divide the sensor network field into a grid of tiles. The size of the tiles can be determined by the constraints expressed in event composition specifications, the resolution of sensor readings, the density of the event detectors, or simply picked arbitrarily. When an event occurs, it is stored in the tile covering the event location.

When processing a regional query for a particular area, the query will be sent to *all* tiles overlapping the queried area. Each of these sub-queries will use the Regional GHT to find the location where the interested data would be stored in each tile. A Regional GHT avoids the need to store events at arbitrary locations in the (potentially large) sensor field, though lookups may need to explore a few tiles if the queried area is large, or falls on a tile border.

Moving beyond support for pull-based queries, we can extend Regional GHTs to detect complex events, as specified by the language of Section 4.2.4. Here, events are not only stored at the location they are created, they are also pushed to a rendezvous point determined by the hash of the event type of any complex event specification for which they may be a part. Event processors at that location attempt to construct complex events that meet the event specification.

Sensor nodes constantly produce primitive events with metadata such as timestamp and location information. When a lower-level event is detected, it is sent to all higher-level event processors that need the lower-level event as input. Due to the locality preserving effect of the Regional GHT, the lower-level events only need to be sent to the higher-level event processors in tiles that contain the lower-level event's location (with occasional additions, described in Section 4.7).

As higher-level events are computed, these events may be sent to the locations of even higher level event processors. When low-level events are combined into a complex event, redundant data is removed, and only the attributes attached to the new event are pushed

to higher-level processors, usually at a lower rate than the lower-level events.

Hierarchical event processing is performed efficiently from bottom up. At each level, events are hashed and distributed evenly within the tiles due to the advantage of GHTs, and can be directly accessed by ad-hoc queries. A Regional GHT is basically a spatial index making spatial queries efficient.

In the application of Section 4.3, there is a Person Detector on each stereo camera node, so the *person_detected* primitive events are stored at the nodes where they are detected, and propagated to *running_person* detectors. The *running_person* and *running_group* also propagated to the processors for the specifications in which they are referenced. Finally, *people_chasing* events are sent to a base station for human attention. Figure 4.14 shows how the events are processed in a hierarchical order. Note the *people_chasing* events are stored at the same node where the dependent *running_group* events are produced. This is an optimization decision explained in Section 4.9.

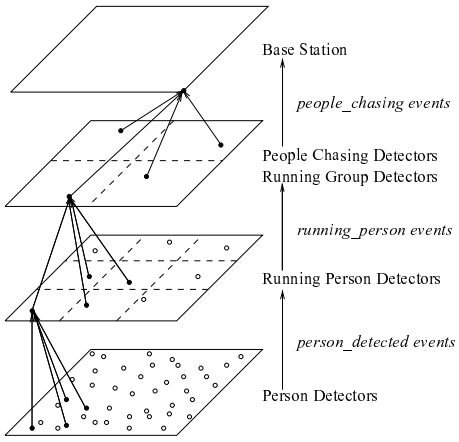


Figure 4.14: The hierarchical event processing for people chasing detection. Each event detector accepts multiple lower-level events as input and produces higher-level events. Events are pushed from bottom up to the base station.

In DCP, the original GHT's `put()` function is used for sending lower-level events to higher-level event processors, rather than storing directly. So the `key` in the `put()` function is key of high-level events, while the `value` is the lower-level event. However, the `get()` function still works the same way, returning the event data associated with the `key`. This is

because we use the proactive approach to propagate events. Lower-level events are pushed to higher level event processors using the `put()` function, rather than the event processors fetching the lower-level events using the `get()` function. Instead, the `get()` function is used only to perform a regional query. After a higher-level event processor receives lower-level events as input, it may store the lower-level events locally for the purpose of time-related event aggregation, but these lower-level events are not returned from ad-hoc query executions.

4.6 Temporal Rehashing

To further balance the transmission and computation load in the network, *Temporal Rehashing* periodically changes the hash location of a given key to eliminate hot spots in the network. With *Temporal Rehashing*, the locations of the complex event processors will be periodically changed, altering the nodes which receive and store the events. Therefore, *Temporal Rehashing* load balances the bandwidth, CPU, and power usage among nodes. The form of the hash function for Regional GHT with Temporal Rehashing is *hash_rt(key, region, time)*.

`time` is the timestamp of the event. Just as DCP divides the sensor region into regularly spaced grids, time is divided into periods of known length. Two events with times in the same period (and equal keys and regions) will be hashed together. If the period differs, the events will be hashed independently, though the returned coordinates will still fall within the same geographic region.

4.7 Interest Area and Interest Interval

In space-related event aggregations, higher-level event processors often express constraints between their lower-level events, rather than absolute constraints. “Find to two people with 3 meters of one another.” rather than, “Find any people in the auditorium.” This implies that lower-level events may require forwarding to tiles besides the ones they are located in,

so that they may be matched and high-level events can be computed.

Taking the *running_group* event for example, when two people are running near an edge shared by two tiles, they may be running close to each other but on different sides of the edge. If the *running_person* events are only sent to tiles containing their locations, this *running_group* event will not be detected. To detect the *running_group* event, the Running Group Detectors in both tiles should be able to observe both of the two *running_person* events. We introduce the notion of *Interest Area*, which represents the area around a lower-level event's location that may impact a higher-level event processor. The size of the Interest Area is determined by the spatial constraints that the higher-level events place on the lower-level events. Figure 4.15 shows how Interest Area causes events to be sent to multiple tiles.

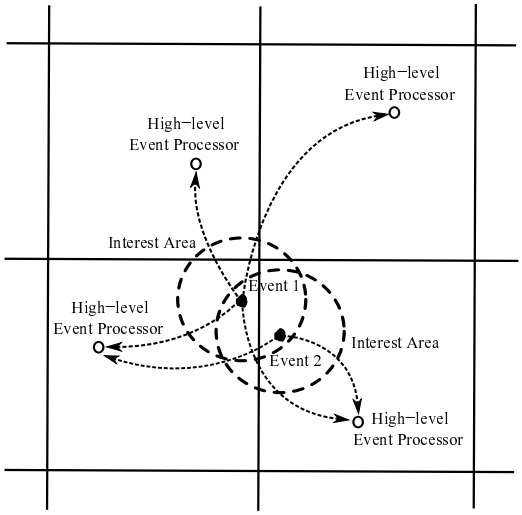


Figure 4.15: Interest Area causes events to be sent to multiple higher-level event tiles to allow space-related event aggregations. Event 1's Interest Area overlaps with 4 tiles, so it will be sent into all 4 tiles. Event 2's Interest Area only overlaps with 2 tiles, so it will be sent into 2 tiles.

For the same reason, in time-related event aggregations, when temporal rehashing is used, events must be sent to the hash locations of different time periods to allow time-related aggregations. Taking the *running_person* event for example, when a person starts running right before the time of rehashing, and stops running running immediately after

the time of rehashing, the two *person_detected* events happen before and after the rehashing time need to be sent to the hash locations in both time periods. Analogously to the Interest Area, we introduce the notion of *Interest Interval* which is the time interval around an event time that may affect higher-level event processors. The length of the Interest Interval is determined by the temporal constraints the higher-level events place on the lower-level events. Figure 4.16 shows how Interest Intervals cause events to be sent to the hashed locations for multiple time periods.

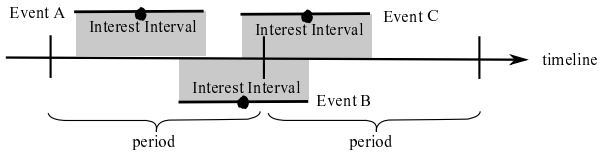


Figure 4.16: Interest Interval causes events to be sent to event processors of multiple time periods to allow time-related event aggregations. The Interest Intervals for Event B and C overlap with two time periods, so they will be sent to hash locations for both periods. Event A's Interest Interval only overlaps with its own time period, so it will only be sent to the hash location of its own time period.

Interest Area and Interest Interval are used to guarantee that no events are missed because of the usage of Regional GHT and Temporal Rehashing. Whenever a higher-level event has spatial or temporal constraints on its dependent lower-level events, an Interest Area or Interest Interval will be applied to the lower-level events. When a higher-level event depends on more than one type of lower-level events, each type of lower-level events can have different Interest Area sizes and Interest Interval lengths. The effects of Interest Area and Interest Interval may be compounded. For instance, if Event 1 in Figure 4.15 and Event B in Figure 4.16 are the same event, it will be sent to $4 * 2 = 8$ different hashed locations.

We now show how to map the event specification to the size of Interest Area and the length of Interest Interval, taking the *running_person* event as an example. Referring to the specification of the *running_person* event in Section 4.3, if the *MAX_SPEED* a person can run equals to 10 m/s, and *SRC_PERIOD_PD* = 0.5s, the Interest Area for the dependent *person_detected* events will be a circle around the person's location with a radius of *SOURCE_PERIOD_PD* * *MAX_SPEED* = 0.5*10 = 5 meters, which means the locations of two

consecutive *person_detected* events that can trigger a *running_person* event can be at most 5 meters apart. Here we assume all the stereo camera sensors are synchronized in time. When a person is running closer than 5 meters to the edge of a tile, this person will also be reported to the Running Person Detectors in the other tiles within 5 meters range. Therefore, when the person runs into an adjacent tile, he/she will be immediately detected running by the Running Person Detector in the that tile. The interest interval can be easily picked as two times the period of the *person_detected* events being pushed to the Running Person Detectors, which is 2*$SOURCE\_PERIOD\_PD$ = 2*0.5 = 1 second.

In queries that match disparate events, such as "find a blue ball within 10m of a red ball," the sum of the Interest Areas for each event must be 10m. Any appropriate combination may be selected, with the expected rarity of each event and the reuse of each event in other queries playing a role in selecting an appropriate trade-off.

4.8 Event Implementation

In order to realize the Hierarchical Event Processing and perform complex event detection in practice, the base implementation of *Event* has the following important fields: *event_id*, *event_type*, and *target_type*. *event_id* is the event identifier. Events are identified by the event name, or a system-wide unique identifier. *event_type* indicates whether this event is primitive or complex. *target_type* tells how the event processors are located in the network. Its value can be `self`, `ght` or `base`. When *target_type* is `self`, there is an event processor on each node, processing the lower-level events generated on the current node. When *target_type* is `ght`, the network field is divided into tiles. The field *tile_size* indicates the size of the tiles. There is an event processor in each tile with its location computed by the Regional GHT. If Temporal Rehashing is used for this event, there will also be a *rehash_period* field. When *target_type* is `base`, the event processor is on a base station, all lower-level events are sent to the base station. There are additional *target_id* and *target_loc* fields to provide the node_id and geographic coordinates of the base station. This target_type is used to simulate

a global query issued from a base station for data collection purpose.

Each event has a list of *source_event_ids*, which are all the lower-level events that make up this event. The subevents can have different spatio-temporal constraints, so each lower-level event can have a differently sized *interest_area* and different length of *interest_interval*.

We assume that for any specific application, all events and their dependencies are specified *a priori*. The DCP framework leverages this information and forms a hierarchically connected event processing map as discussed in Section 4.5.

Each event processor has a *process*() function. This function is performed whenever a lower-level event is received by the event processor. This function aggregates lower-level events into higher-level events, using local storage to temporarily store lower-level events for temporal aggregation. When a new complex event is produced, it is forwarded to the higher-level event processors that are dependent on this event type.

4.9 Optimization

If the characteristics of the queries in the application is known *a priori*, such as the distribution of the query regions, the events being queried, and the frequency of the queries, the tile size for Regional GHTs can be optimized to minimize the network utilization.

Supposing an Interest Area of radius R for the input events, there exists an optimal tile size with side length L which minimizes (on average) the total distance D that an event must be transmitted to reach all relevant event processors. When L is small, the event location has a higher chance of being near a tile edge and the event will be sent into multiple tiles which increases D. When L is large, the random location of the event processor may be far away from the event location, which also increases D. Our simulation shows the experimental relation between D/R and L/R, as shown in Section 4.10.2.

Another optimization can be done when the source events of a complex event only come from the same node. In this case, the complex event processor can be located at the exact node where the source events are produced. This optimization eliminates the unnecessary

event delivery. For example, the People Chasing Detectors detect *people_chasing* events by comparing two *running_group* events with same person ids, so the *people_chasing* events can be stored at the same node where the dependent *running_group* events are produced, as shown in Figure 4.14. This optimization requires prior knowledge of the queries in order to choose a hash function that hashes the two queries together.

4.10 Evaluations

We show the the advantages of in-network processing allow distributed constraint processing to produce *more* efficient sensor networks while simultaneously decreasing their complexity. We show how DCP compares to a centralizing algorithm by examining the load distribution and total bandwidth consumed during event collection. We separately evaluate the effectiveness of temporal rehashing by showing how DCP performs without temporal rehashing.

4.10.1 Experimental Setup

We conduct experiments with the NS2 [35] network simulator. Experiments are run in a 300m by 300m square with 200 randomly placed and oriented stereo cameras, each with an 802.11 network interface of 40m range. Several people are moving in the square using a random way-point model with speeds between 0 and 7m/s and no pause time. Our application seeks to find the runners, which we define to be those persons moving faster than 5m/s. One person is considered to be the *guard*, constantly chasing the closest running person. The *guard* starts idle and looks for anyone else that is running. If there is at least one person running, the *guard* immediately starts chasing (at 10 m/s) the closest runner. When the *guard* catches the running person, it matches the speed of the runner, so they are running close together. After the person being chased changes to a speed lower than 5m/s, the *guard* becomes idle and looks for another running person to chase.

Objects can be seen by a camera if the objects are within 40m of the camera, and the camera is orientated in the proper direction. Cameras are assumed to have a 90 degree

viewing angle and take pictures twice per second. There is a Person Detector on each camera node, producing a *person_detected* event whenever the simulated camera sees a person. Running Person Detectors are placed in a grid of 100m by 100m squares with a rehashing period of 30s. Running Person Detectors receive all *person_detected* events with a 2.5m radius Interest Area. Running Group Detectors and People Chasing Detectors are placed in a grid of 150m by 150m squares with rehashing period of 50s. The 10s Interest Interval assumes a *people_chasing* event is detected when two persons are running close to each other for more than 5s. *running_person* events are sent to People Chasing Detectors every 5s and their Interest Area is a circle with 10m radius. Whenever a *people_chasing* event is detected, it is sent to the base-station located at (0,0). Each simulation runs for 1000s. GHT uses the GPSR [21] routing protocol to forward packets to destination locations. We turn off the GPSR's perimeter mode which is used to bypass holes in the network. In our experiments, we chose a dense node deployment to allow better camera coverage, so there are unlikely to be any holes in the topology.

4.10.2 Tile Size Selection

We first analyze the effect of varied tile sizes on the performance of Regional GHTs. A large tile to interest area ratio (L/R) requires every GHT store to travel further, while a small ratio requires multiple stores due to Interest Area overlap with nearby tile edges. We run the simulation for different ratios for the Running Person Detectors with $R = 2.5$m. Figure 4.17 shows the simulation result. Since camera range and radio range are similar, *person_detected* events can almost always be sent to hash location in one hop when $L < 40$. The hop value shown in Figure 4.17 is discrete. Each hop covers at most 40m in distance, the maximum radio range, which is about $16R$. When L is larger than $15R$, the average event delivery distance in hops increases slowly. This is due to the effect of discrete hops. We expect that the a faster increase will be shown when L is large compared to the hop distance, not R.

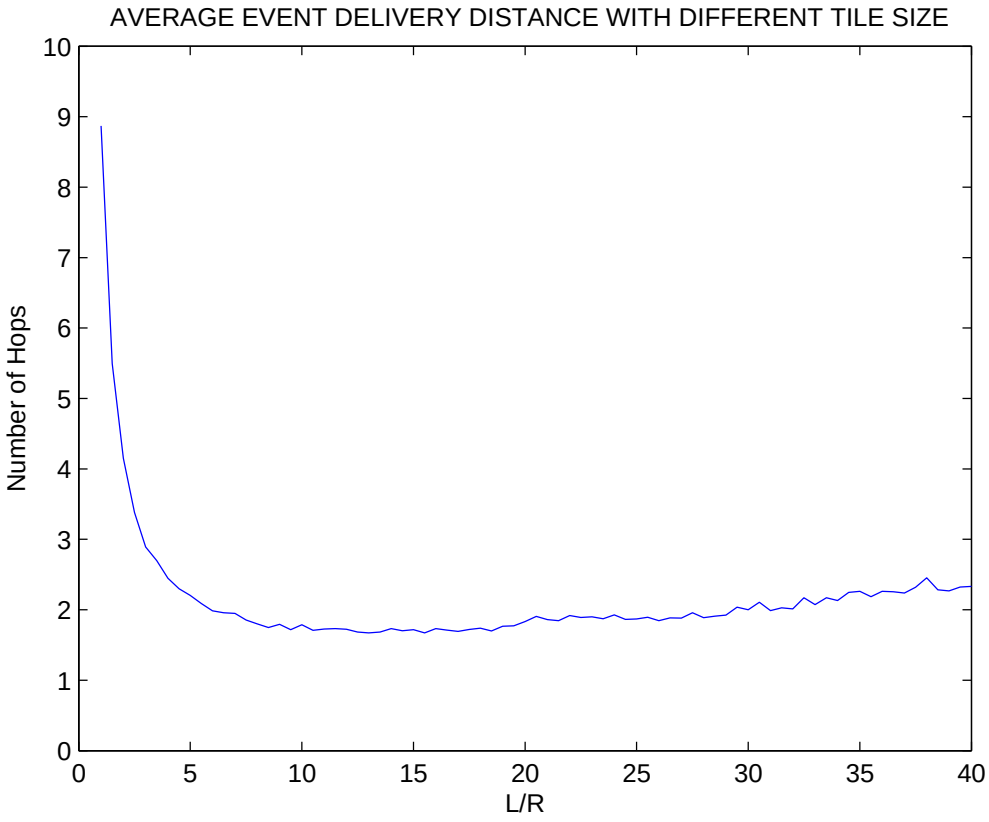


Figure 4.17: Simulated relation between the average delivery distance (counted in hops) of the *person_detected* events and tile side length L. When the radius of Interest Area R is fixed, the average event delivery distance reaches its minimum of about 1.7 hops when L is approximately $15R$.

4.10.3 Bandwidth Distribution

DCP distributes complex event processors throughout the network using the Regional GHT. We test the network traffic with three different detection techniques: Centralized Processing (No DCP), DCP without Temporal Rehashing, and FULL DCP (with Temporal Rehashing). All three experiments detect the *people_chasing* events at the base-station. Figure 4.18 shows the experimental results. The centralized algorithm, sending all primitive events back to base-station without in-network aggregation, creates a serious hot spot near the base-station. When DCP is used, the network traffic is evenly distributed. The network traffic in the center of the sensor field is more than the traffic on the edge of the field. This

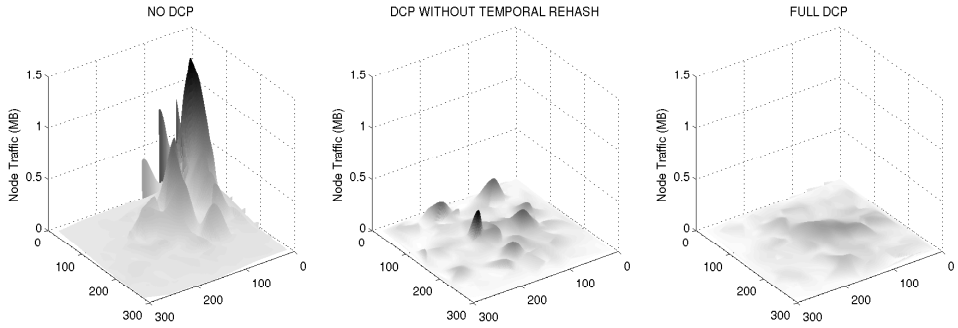


Figure 4.18: Bandwidth distribution comparison of Centralized Processing (left), DCP (right), and DCP without Temporal Rehashing (middle). Centralized algorithm sends all primitive events back to base-station. Both DCPs send *people_chasing* events back to base-station. Figures show the node traffic distribution created by all event packets.

is because the moving objects tend to move near the center where more events are created. Without Temporal Rehashing, traffic still tends to build up around several locations where the event processors are placed. By using temporal rehashing, DCP further balances the network traffic.

It is worth noting that abstractions that allow neighborhoods or regions of nodes to be defined would not allow purely local aggregation. A technique similar to our Interest Area approach would be required to find matches that span regions. This difficulty is a core reason we avoid the intermediate notion of region and proceed directly to data-centric event constraints.

4.10.4 Bandwidth Usage

Not only does DCP balance the network traffic, it also reduces the number of radio transmissions because events are usually sent to close destinations and require fewer hops. Figure 4.19 shows that DCP cuts the total network traffic by about 65%. Equally important, the traffic of the busiest node is much lower under DCP and grows slowly with additional moving objects, prolonging network lifetimes. The adoption of Temporal Rehashing further reduces the requirements on the most heavily loaded node by sharing the work of event processing over time.

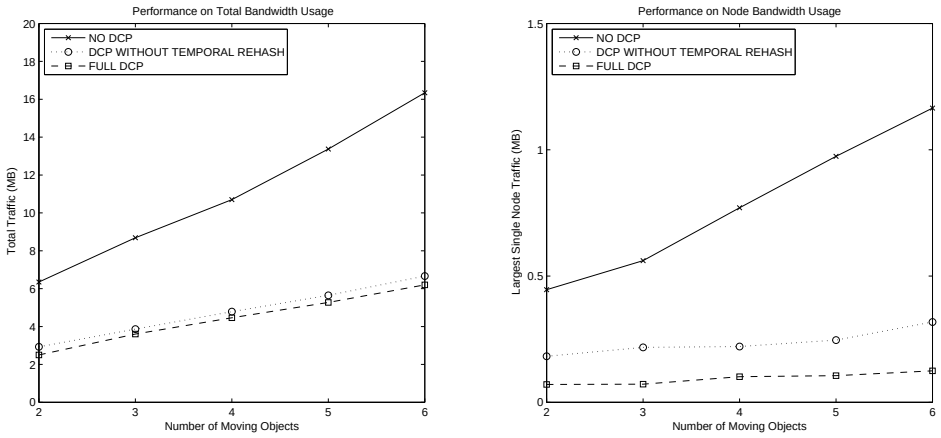


Figure 4.19: The bandwidth usage comparison of centralized algorithm, DCP without temporal rehashing, and DCP. The number of moving objects is increased from 2 to 6 (including the *guard* object). Left image shows the total traffic in the network. Right image shows the traffic of the busiest node. DCP uses much less bandwidth in both cases.

4.10.5 Node Failure

The fault-tolerant aspect of GHTs is discussed in detail in [39]. Here we show the high-level impact of this fault-tolerance. We show how the ability of the sensornet to detect high-level events is affected by the loss of camera and radio nodes. Our example object tracking application depends on radio connectivity, but also on camera coverage to detect the low-level events in the first place. As cameras and radios are lost, detection suffers.

We run the simulations with fewer nodes and report the number of complex *people_chasing* events detected. Dead nodes are unable to generate *person_detected* events nor communicate through radio. We examine the low-level *person_detected* events detected by the live nodes to calculate the ideal number of *people_chasing* events that could possibly be detected in each simulation. We show the actual number of high-level events detected as a measure of performance. We also repeat the simulation with nodes that function as radio nodes, but lack cameras. This separates the influence of network connectivity and sensor availability. Figure 4.20 shows that at densities over 70% of our baseline, there is little effect on the aggregation abilities of DCP. In all cases, about 5-8% of high-level events are missed, perhaps due to network partition or congestive losses. Below 70%, radio coverage becomes

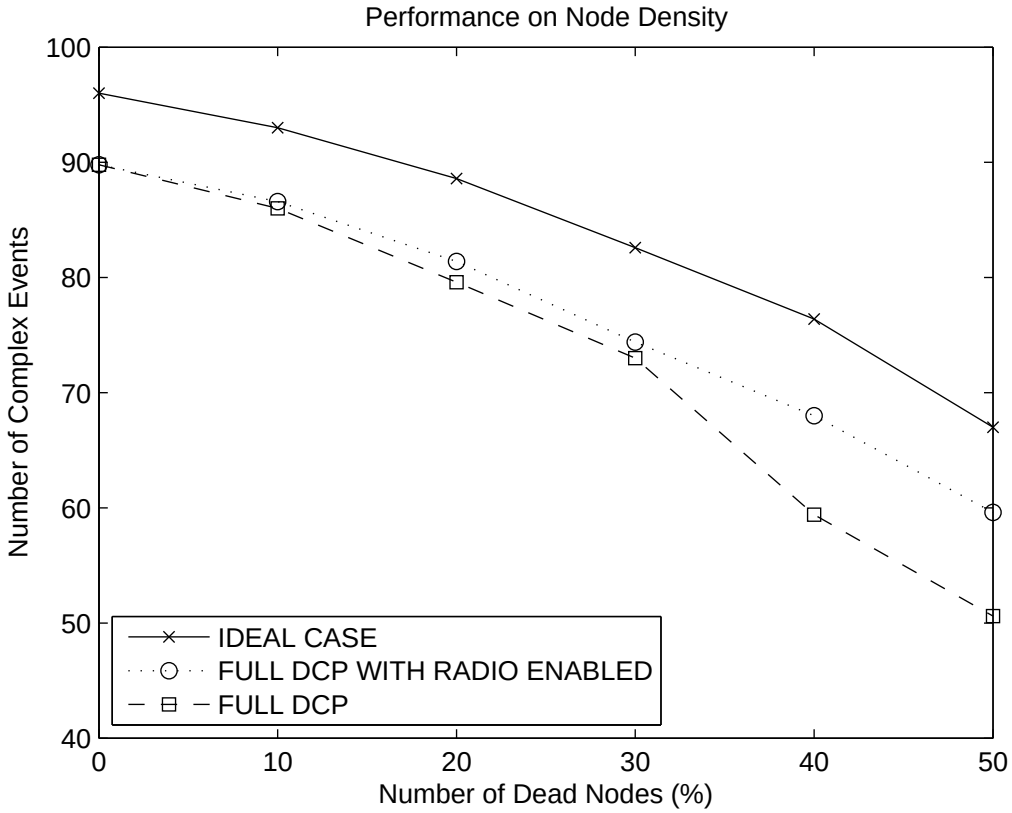


Figure 4.20: The effect of node density on DCP effectiveness. Simulations are run with different node densities, from 200 cameras in a 300m by 300m field, down to 100 cameras. With fewer cameras, fewer high level events are detected due to lack of camera coverage, and decreased radio connectivity. The top line show how many high-level events *could* be detected if all low-level events were aggregated. The middle line shows how many high-level would be detected if some percentage of cameras, but not radios, are turned off. The bottom line shows the effect of complete outages (camera and radio). Results are the average of five simulations.

a problem. If the "dead" cameras continue to function as radio nodes, DCP performs well, still finding about 90% of potential complex events. However, if "dead" nodes have neither camera nor radio, the success rate of DCP detecting complex events degrades to about 75%.

Chapter 5

Probabilistic Object Tracking

Cross-camera object matching is an important application in visual sensor networks. It allows moving objects to be tracked over a larger space than that be covered by any single camera. Each camera view covers a fraction of the whole monitored space. The camera views can be either isolated or overlapped. The goal of object tracking is to reidentify objects as they move between camera views. When objects move from one place to another, they will be captured by different cameras. Object matching is accomplished by identifying equivalent objects detected in different cameras, based on object specific features.

5.1 Single Camera Object Tracking

When an object appears in a camera view, it usually appears in multiple consecutive image frames within a time window. Tracking objects within a single camera view is relatively easier and more straightforward than cross camera tracking. Objects detected in consecutive image frames in a single camera share similar illumination conditions, view angles and view distances. The similarity in object appearances makes the object features more reliable in object matching. Besides, objects usually move at known speed and won't immediately change their locations. Objects in consecutive image frames can be easily tracked by their relative locations and moving directions. In this thesis, we assume that single camera object

tracking is solved and can yield reliable object matching results. Instead, our work focuses on cross camera object tracking where large object matching error rates may occur and studies how to improve the matching performance using probabilistic methods.

5.2 Feature Matching

In computer vision, features are parts of images that the vision problem or application is interested, and are often associated with abstract image information called feature descriptors. There are many different types of features that have been used in various vision algorithms. Corners, for example, are image points where two edges intersect, and are represented by the pixels in the local neighbourhood. Color histograms, on the other hand, are properties of blobs of image pixels, and are usually in the form of multi-dimensional histograms. Feature detection is the process of finding these application specific features in images and computing their descriptors.

To match objects using any object feature such as corners, color histograms, contours, gradients, and so on, common steps are performed in the following order. First, each camera isolates the foreground using background subtraction, resulting in blobs of image pixels. Next, feature descriptors are computed upon the image blobs using feature generators. Finally, descriptors are compared using feature matchers to determine if they belong to the same objects. Acting individually, a feature matcher might compute the distance or similarity between two descriptors, and then compare the match result to a preset threshold value to determine if two objects match.

Desirable features must be repeatable. Similar features should be detected in different images of the same scene. Repeatable features are important to object tracking applications. Features from the same objects can not vary too much in order to be matched correctly. However, this is not always true in cross camera matching where illumination conditions, view angles and view distances may be quite different among cameras. This causes many features to work much less well in cross camera matching than in single cameras.

An object tracking system based on any individual feature may be incorrect. If the feature matcher determines that two different objects are the same but they are actually not, it is a false-positive match. On the other hand, if the feature matcher thinks the same object is two different objects, it is a false-negative match. These errors come from several sources. First, background subtraction can be imperfect. The foreground image blobs of the objects can be missing pixels, or contain pixels from the background or from other overlapping objects. These errors introduce noise when the foreground image blobs are used to compute the object feature descriptors. Second, image blobs are 2D projections of 3D objects. Image blobs of the same objects may vary considerably when the objects are seen by the cameras from different angles and distances, or under different illumination conditions. Third, each object feature has its own strengths and weaknesses that cause them to work well in some scenarios and poorly in others. For example, color histograms are usually a good object feature for object matching. However, in dimly lit environments, the colors on the objects will be less vivid and color histogram matching will be less effective. Further, if a team of uniformed individuals walk by, color histograms will not be a strong indicator of object similarity. In these scenarios, alternative object features may be more accurate for object matching.

The common way to compare two object features descriptors is to compute their similarity or difference value. Each feature type has its own way to measure feature similarity. Histogram based features, for example, often compute the euclidean distance or the intersection distance as the difference value [44]. Similarity is the complementary value of the feature difference. For difference value normalized between [0,1] where 0 means no difference and 1 means completely different, similarity value can be defined as $1 - difference$. Given a similarity value between two object features, a feature matcher compares the similarity to a predefined threshold. If the similarity is larger than the threshold, the matcher votes "same" as the object matching result, otherwise "different".

Figure 5.1 shows an example object matching test using the RGB color histograms. The matching result is the similarity of the RGB color histograms of two testing object

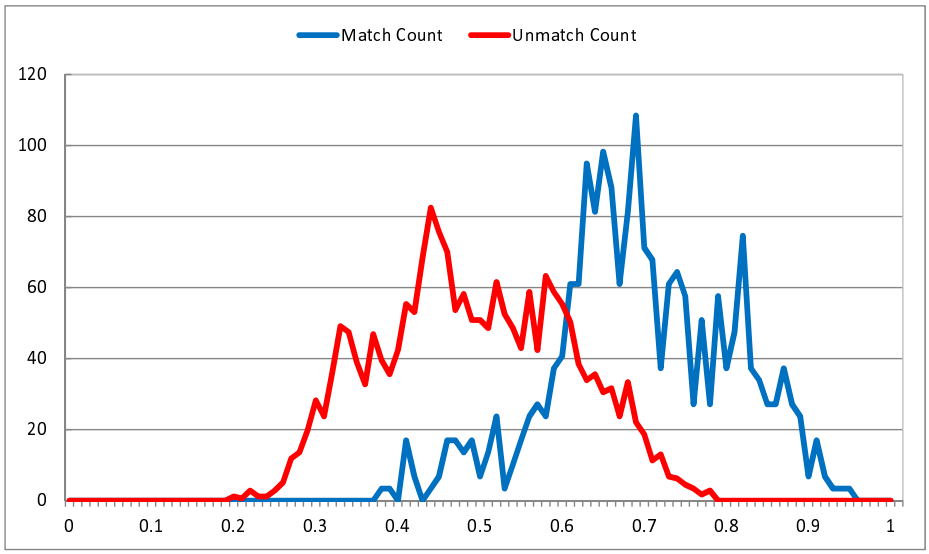


Figure 5.1: Normalized distribution of a RGB color histogram matcher based on 4k test samples. The horizontal axis shows the similarity matching results of two testing image blobs using RGB color histograms, scaled into range [0,1]. The similarity range is divided into 100 bins. In each bin, the number of matches and unmatches are counted separately. The overlapping area shows where the matcher may give incorrect votes.

blobs, scaled into range [0,1]. The dark line is the matching result distribution for true matches. The gray line shows the similarity distribution when comparing different objects. This object feature matcher achieves maximum matching accuracy using a threshold value 0.60. Using this threshold, the matcher votes "same" when similarity values are above 0.60, otherwise "different". However, while matching votes are always correct when similarity values are below 0.38 or above 0.80, similarity values between 0.38 and 0.80 can come from both true matches and false matches and will inevitably lead to wrong votes.

5.3 Multiple Feature Fusion

Since any object feature can match objects incorrectly and its matching accuracy can vary greatly in different scenarios, using a single object feature for cross camera object tracking is suboptimal. Instead, using multiple object features together can overcome the hard times when some features fail to give correct matches. The fundamental problem becomes how to fuse matching informations from several features when there is disagreement.

5.3.1 Majority Vote

One intuitive way to combine multiple feature matchings is to ask each feature matcher for a binary vote on whether two blobs match. The votes for "same" match and "different" match are counted, then the majority vote is picked as the final match decision. We call this fusion method the *Simple Majority Vote* (SMV). This approach suffers because the binary matching votes of each matcher do not consider how reliable each individual algorithm's votes are.

The reliability of the feature algorithm can impact the fusion in several ways. First, each feature algorithm may have different matching accuracy across different camera pairs. For example, if two cameras share similar illumination conditions, RGB color histograms may work well. But for a pair of cameras, one in a bright place and the other one in a dark room, RGB color histograms will perform poorly. Treating matching votes from different feature algorithm indiscriminately will lower the fusion accuracy towards the less accurate features. If we assign less weights to the less accurate features in the function, the accuracy of the fusion function will be closer to the more accurate features. This method is often called the *Weighted Majority Vote* (WMV). One way to derive the weights is to use the historical matching performance values as the weights of each feature.

Second, each individual matching vote from a feature can vary in its reliability too. As in the previous RGB color histogram matcher example shown in Figure 5.1, when the computed match result is above the threshold 0.6, the matcher votes for a match and is usually correct, but the closer the match result is to the value 0.6, the probability is higher that the match vote is incorrect. Intuitively, if a feature matcher is uncertain about its votes, its votes should be given less weight in the final decision other than simply using majority vote which treat each vote the same.

5.3.2 Matching with Confidence

An improvement on the above method might employ confidence scores generated by the feature matchers. Then larger weights can be assigned to feature matching votes that report

higher confidence scores in the final fusion function. In the simplest case, a feature matcher like the RGB matcher discussed above, might report low confidence when the similarity metric is near 0.6.

Going further, matchers might also be supplied with extra information about the current environment or recent object appearances. The added information would allow them to measure how confident they are about their matching votes. For example, we might provide illumination information to color histogram matcher to help it decide whether the environment favors a strong weighting of its results.

Unfortunately explicit confidence scores can be difficult to implement because their accuracy is highly dependent on the feature designer's understanding of the feature's strengths and weaknesses as well as a clear and complete description of the characteristics of optimal environmental conditions for the matcher. Worse, all feature algorithms would need to agree upon a unified measure of their confidence scores so that they can be fused "fairly". Consensus on such a detail is unlikely to be reached easily, which would limit the use of independently developed feature matchers.

One way to remove the complexity of measuring confidence internally in each feature algorithm is to use the Weighted Majority Algorithm(WMA) [25]. WMA assumes no prior knowledge about the accuracy of the algorithms in the voter pool. Instead, the confidence scores of the algorithms are computed by comparing their historical performances. WMA puts a weight on each feature to express its accuracy, based on the feature's agreement with known ground truth. WMA does not require confidence information from the feature algorithms and is easy to implement. However, putting one weight on each algorithm will not work well as scenarios change. If the environment or objects change, a single weight won't adjust to cope with the feature matchers' performance variations. In online tracking applications, step-by-step ground truth is not available, so weights cannot be adjusted.

In online scenarios, a super majority vote could be used in place of ground truth, allowing for dynamic adjustment of weights. For example, weight suppression might be applied to matchers that disagree with a conclusion that 80% of matchers agree on. We have studied

using online WMA for object matching but found its performance is very sensitive to its internal parameters such as the weight suppression factor, the super majority threshold, and the minimum weights which are used to prevent the weights from being decreased to 0. With careful tuning, WMA can work well in certain test scenarios while performing quite poorly in others. We believe that online WMA is too fragile to be used in our object tracking applications and need to find a more robust approach to utilize empirical confidences.

5.3.3 Matching with Probability Distribution

This thesis presents a new technique to fuse multiple feature matching results using historical match probability distributions. We build match probability tables that correlate historical match probabilities for various reported similarity levels from each feature matching algorithm. Then, for online matching, we fuse the values from the match probability tables to make a final match decision. This technique doesn't require explicit knowledge about when an object matcher is confident about its votes. Nor does it require that the match similarities scale linearly, or even monotonically with match likelihood.

Instead, our fusion technique derives match probabilities for each matching algorithm using empirical samples, not fixed formulas, so feature matchers don't need to know about confidences, or even how their match similarities correspond to match probabilities — they require no threshold, for example. Moreover, the fusion process provides a unified confidence measure defined at an arbitrarily fine scale without explicit knowledge of the environment. With this technique, new feature matchers can be added into the fusion function with ease.

Incorporating reliable confidence scores for match votes into the the fusion function can boost the object matching performance in cross-camera object tracking. Computing confidence scores with the feature matchers' empirical performance is more robust than asking designers of feature matchers to compute their confidence values. The derived confidence values are from real-life matching samples, not from the feature designers' guess work. The Weighted Majority Algorithm (WMA) implements confidence by suppressing weights when the algorithm gives wrong predictions. However, in WMA, a single weight is used for each

feature matcher in all possible situations derived during training and is not responsive to the algorithm's performance variation under different circumstances. Or, if weights are adjusted dynamically, the lack of ground truth at runtime may lead to incorrectly decreasing the weights of algorithms despite their correct choices.

Our probabilistic object tracking system consists of two phases. In the offline training phase, we collect ground truth data and run the feature matchers to build match probability distributions. These distributions indicate, for each reported similarity score, how often the two compared image blobs were actually from the same object. In the runtime phase, we use the reported values only to extract probabilities from the historical tables. We fuse these derived probabilities rather than the scores reported directly from the matchers.

Building Match Probability Tables

Before we start matching objects, we need to build the match probability distributions for all feature matchers. A match probability distribution is a table containing pairs of similarity scores and their corresponding match probabilities reflecting historical results when the given score was reported by that matcher.

To build the match probability distribution of a feature matcher, we first collected a training dataset of image blobs and label each blob with an object id. Image blobs of the same object are assigned the same object id. These object ids are used as the ground truth data during training. When a feature matcher compares two object feature descriptors, a matching result r between 0 and 1 is computed between these two descriptors. We divide the matching result range into N bins with equal width. For each bin, we keep two counters C_t, the number of matches, and C_f, the number of non-matches (based on ground truth).

Let's assume the matching result r lies in the *ith* bin. Without requiring the matcher to vote by comparing the matching result r to some threshold, we simply increment C_{t_i} or C_{f_i} depending on whether it is an actual match in the training dataset. Our training set consists of an equal number of matches and unmatched tests to build up two arrays of counters C_{t_i} and C_{f_i}, $i = 1, ..., N$. We call these two arrays the match probability table of

this feature matcher.

Taking RGB color histogram as an example of object features, Figure 5.2 shows the match probability distribution of the same RGB color histogram matcher shown in Figure 5.1. The bins in which the values are neither 100% or 0%, indicate similarity scores for which the matcher cannot be completed "trusted." Historically, when the RGB matcher reports 0.6, about 40% of the samples where actually matches, while scores above 0.78 were only given to true matches.

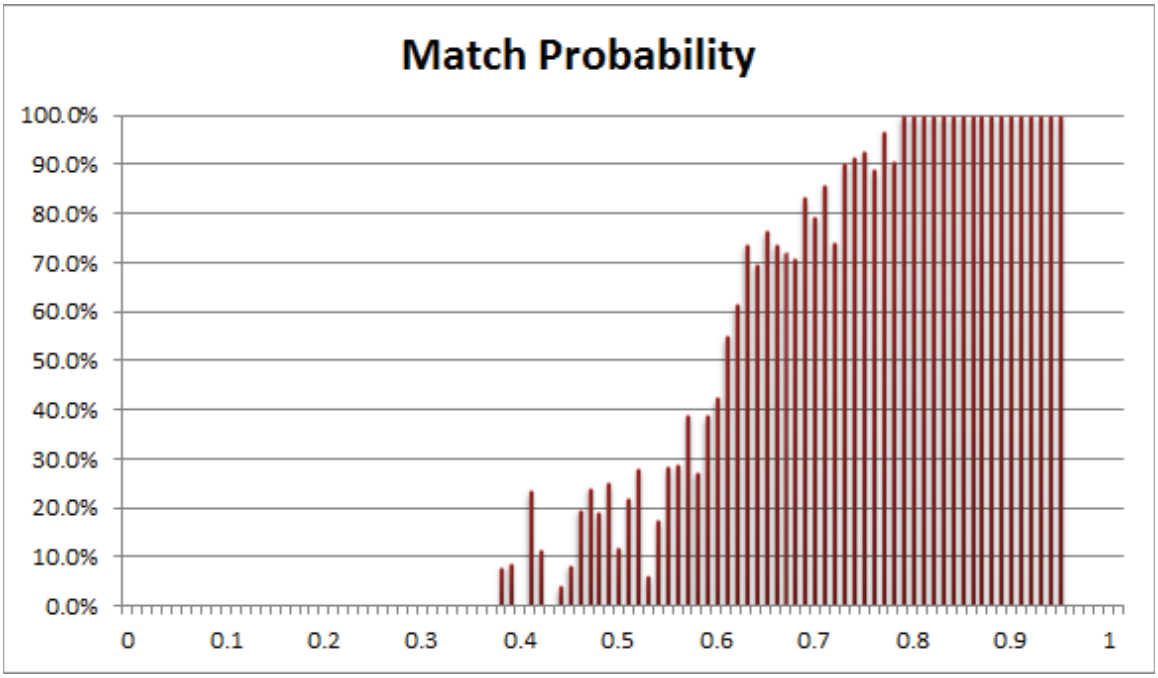


Figure 5.2: Match probability of a RGB color histogram matcher. Scores above 0.78 were only given to true matches. Scores below 0.37 always indicated a non-match. In between, the table captures the likelihood of a match for any particular score.

Matching with Probabilities

Using match probability tables, feature matchers are not required to cast a decisive vote on an object comparison. Instead, each matcher only contributes its own probability estimate based on their feature matching result. With a large training dataset covering many real-life scenarios, the probability values produced by the match probability tables indicate the real

likelihood of a match.

In a runtime object tracking application, we use M different object features. Once an object is detected and its image blob is extracted by background subtraction, all M features are computed for the object. To compute cross-camera matches, each feature matcher computes the matching result r for two object feature descriptors. The fusion process consults the match probability table to locate the bin corresponding to the matching result r, and computes the probability value p,

$$p = \frac{C_t}{C_t + C_f}$$

In Figure 5.2 the values of bars also show the match probability values produced by the match probability table of the RGB color histogram. When the match probability is close to 100%, it indicates a probable match, while close to 0 means a match is unlikely based on the matcher's historical performance. When the match probability is 50%, the matcher is completely uncertain whether it's a match or not. In special cases where $C_t = C_f = 0$ in a bin, the training dataset doesn't produce any sample that lies in the bin. In these cases, we can either set $p = 50\%$ or down-sample the match probability tables to coarser distributions. We might also set $p = 50\%$ when $C_t + C_f$ is too low and doesn't represent a statistically valid probability.

With match probabilities $p_i, i = 1, ..., M$ from all features, we compute the final match probability as the average of all p_i,

$$P = \frac{\sum_{i=1}^{M} p_i}{M}$$

If the final match probability P is above 0.5, we say it's a match.

There are other ways of combining probabilities. For example, if the features are independent, we can also fuse the probabilities logarithmically [3]. However, we don't expect independencies among features and therefore we use linear average instead. Machine learning techniques, such as neural networks, can also be used on these probability values from

each feature to capture the importance of and relations among these features. In our experience, machine learning algorithms can sometimes yield better matching performance comparing to linear averaging, but the improvement is limited and is always at the expense of more computational resources. In the thesis, we argue that using probabilities instead of binary votes can boost the matching performance, even with simple combining techniques. We prove this argument in the next section by using simple linear averaging to fuse the probabilities.

5.4 Fusion Performance Study

In order to evaluate our fusion technique in cross-camera object tracking, we set up a camera network in an indoor environment with 9 D-Link DCS-900 Internet cameras [8]. Figure 5.3 shows the floor plan of our testing environment and the locations and orientations of the cameras. These cameras take VGA sized (640x480) pictures at about 5 frame per second. Sample images from these cameras are shown in Figure 5.4. Image frames in Motion JPEG format are streamed to data collection computer servers through network connections. The cameras, labelled with unique IDs coming from the last digit of their IP addresses, are shown in Figure 5.3. We gathered various scenarios in which six individuals of varying height, sex, and clothing walked around the area covered by the camera network. A dataset of about 10 minutes of images is collected from all cameras. We ran background subtraction on all image data and collected about 4,200 foreground image blobs. Each image blob was manually labelled with the corresponding person's name as the IDs of the objects. The whole dataset gives us about 7.6 million different blob pairs , each containing two blobs from two different cameras.

We have implemented several groups of object features based on different aspects of object appearances. The first group consists of several color-based features. We select color histograms in four different color spaces, the RGB, HSV, Intensity, and RG Chromaticity [32] which works well in unevenly lit environment. We added another color-based

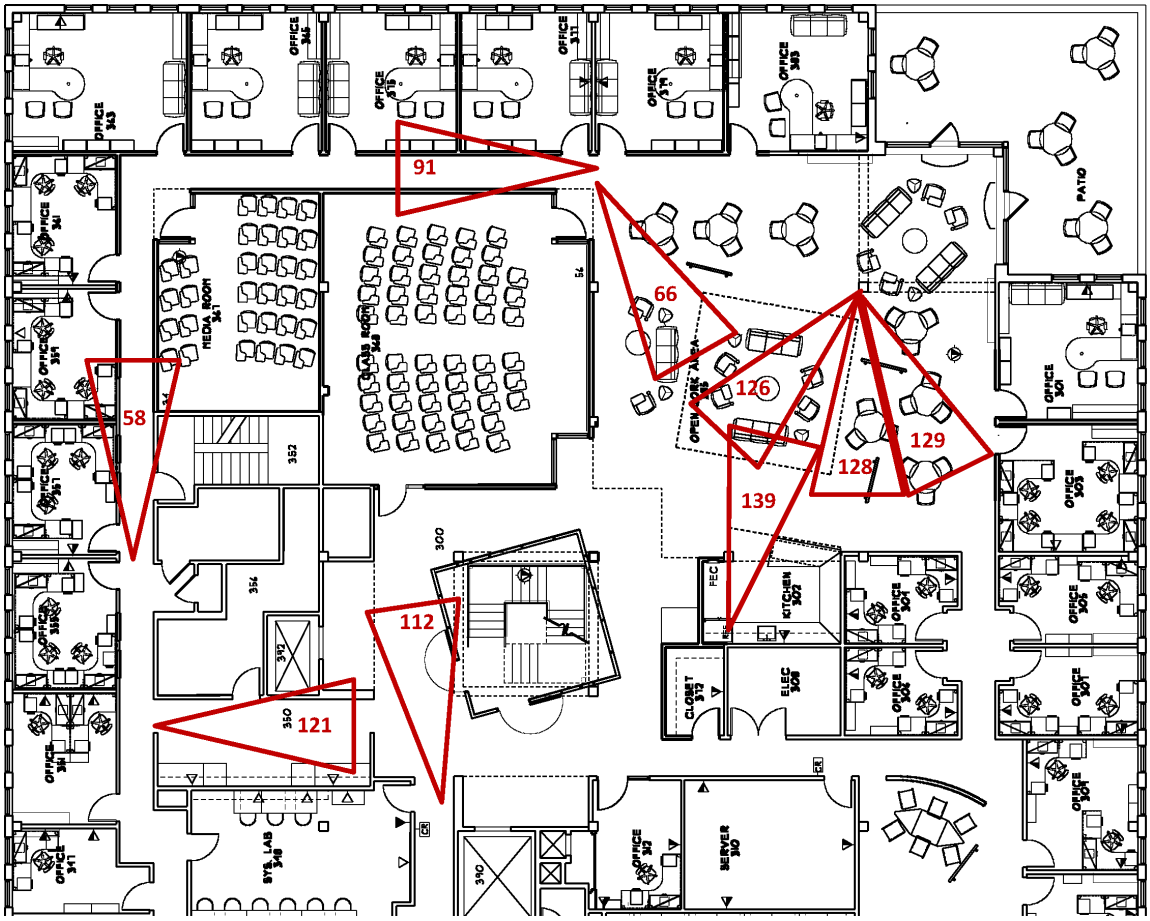


Figure 5.3: Indoor camera network using D-Link cameras and the floor plan. The cameras are labelled with their unique IDs.

feature, Auto Correlogram [18], which tolerates object appearance changes due to changes in viewing position and camera zooms. The last color-based feature is called Vertical Gradient which divides the image blob from top to bottom into several slices with equal height, and then computes the average intensity of all pixels in each slice to build a vertical histogram. The second group consists of shape-based features. In this group we created a simple feature which computes the height-width ratio of the objects, and another histogram feature, this time based on the contours of the objects. A contour histogram is built using the slopes of all edges on the contour of the objects. The third group contains gradient-based features. This group includes Harris corners [15] and SIFT features [26]. In our implementation of

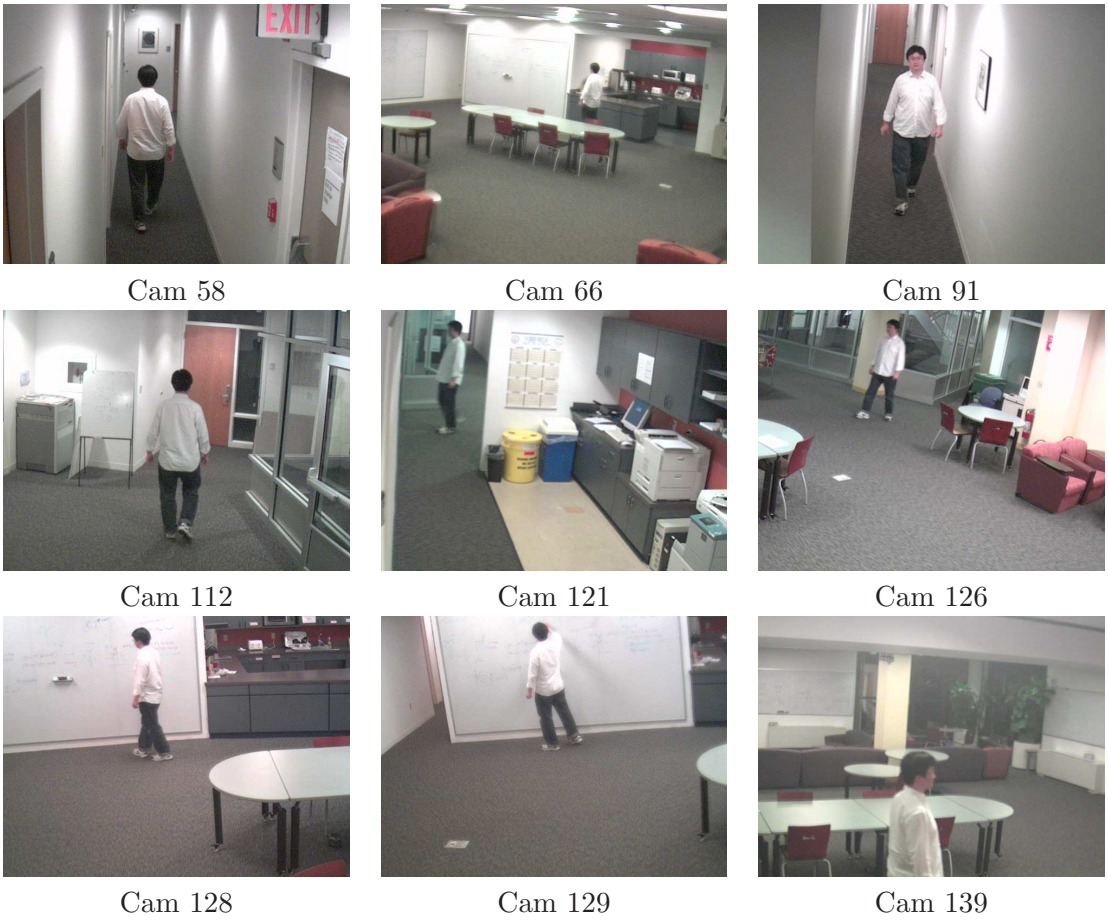


Figure 5.4: Sample images of D-Link cameras. The cameras are set up in a way that the object appearances in each camera view are quite different in terms of view angle and view distance, except Cam 128 and 129 which have similar views.

these feature algorithms, we did not optimize the feature generators and feature matchers to achieve best matching performance because we wanted to investigate whether our proposed fusion technique would work on reasonable, if not perfectly tuned, algorithms. Table 5.1 shows the list of features and fusion algorithms used in our tests and their name abbreviation that are used later.

We implemented a feature detector and a feature matcher for each feature algorithm. In feature matching, the similarity scores of HSV, Intensity, RGB, RG Chromaticity, and Contour histograms are derived from the intersection distance. Vertical Gradient uses euclidean distance. These choices come from our experience and give decent matching

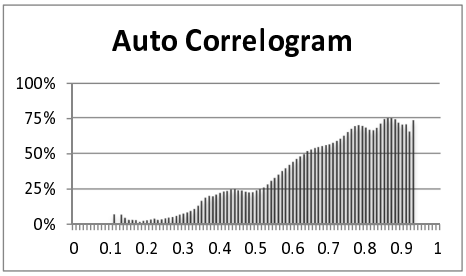
Category	Feature Name	Abbreviation
Object Features	Auto Correlogram	AC
	Contour Histogram	CH
	HSV Histogram	HSV
	Height/Width Ratio	HWR
	Harris Corner	HC
	Intensity Histogram	INT
	RGB Histogram	RGB
	RG Chromaticity	RGC
	SIFT	SIFT
	Vertical Gradient	VG
Fusion Algorithms	Simple Majority Vote	SMV
	Weighted Majority Vote	WMV
	Weighted Majority Algorithm	WMA
	Probabilistic Matching	PROB

Table 5.1: Name abbreviations of all features and fusion algorithms

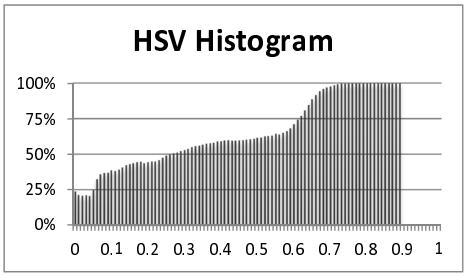
performance on our dataset. Auto Correlogram uses L_1 distance as recommended in [18]. Harris Corner and SIFT features contain a list of detected feature points. They compute the similarity score as the ratio between the number of the matched points and the size of the shorter point list of two features. Height/Width Ratio simply divides the smaller ratio over the larger ratio to get a score within range [0,1].

To show the overall match probability distributions of each feature algorithm, we compute descriptors of all image blobs for all feature algorithms, then compute the match results of all cross-cameras image blob pairs. We implemented features matchers in ways that all matching results are similarity scores in range [0,1]. These similarity scores are used to build the match probability distributions over 100 bins. Figure 5.5 and 5.6 shows the match probability distribution of all feature algorithms in our experiments. They also show the required storage and computation resources for each feature algorithm which we will discuss later in Section 5.4.5.

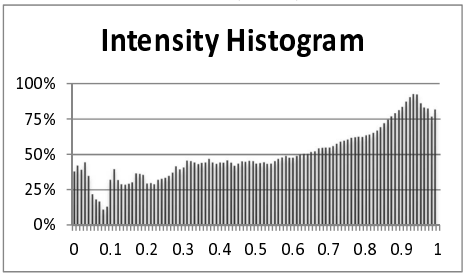
A few properties are striking. The Vertical Gradient may be the most well-behaved feature. Starting at a similarity score of about 0.2, there is a very nearly linear increase in match probability as the score rises to 0.9. If the similarity score were scaled to stretch the 0.2–0.9 range over the range 0.0–1.0 it would approximate a match probability. On



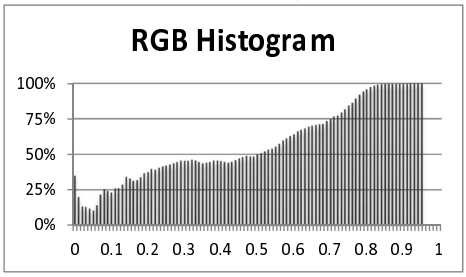
Avg Size (byte): 24,576
Avg Gen Time (usec): 171,579
Avg Match Time (usec): 304



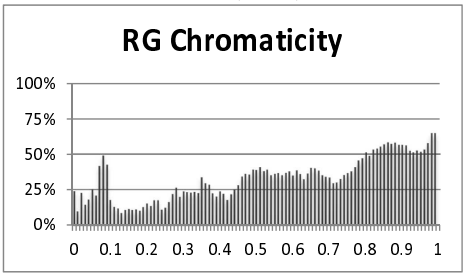
Avg Size (byte): 4,096
Avg Gen Time (usec): 8,410
Avg Match Time (usec): 11



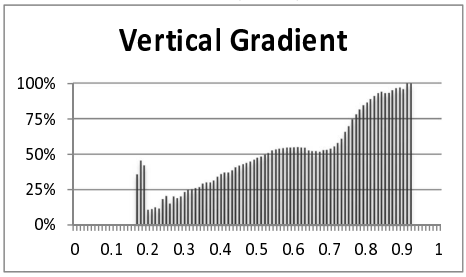
Avg Size (byte): 64
Avg Gen Time (usec): 7,730
Avg Match Time (usec): <1



Avg Size (byte): 4,096
Avg Gen Time (usec): 7,979
Avg Match Time (usec): 11



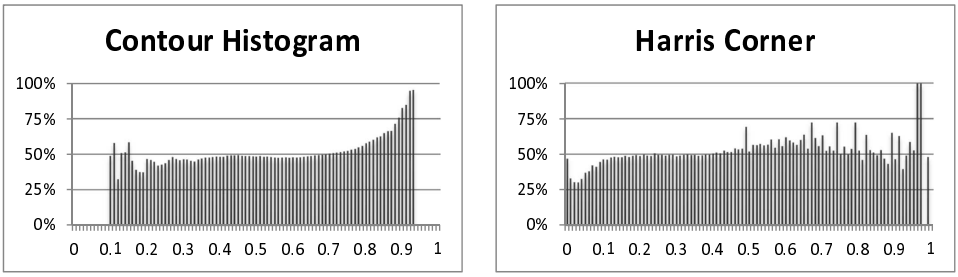
Avg Size (byte): 512
Avg Gen Time (usec): 8,369
Avg Match Time (usec): 1



Avg Size (byte): 64
Avg Gen Time (usec): 1,652
Avg Match Time (usec): <1

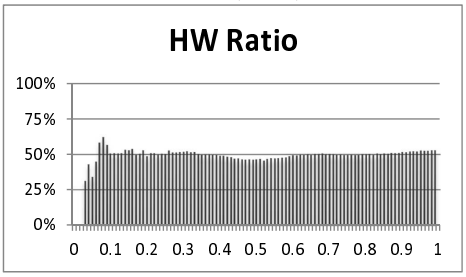
Figure 5.5: Match probability distributions of color-based feature algorithms . Under each graph, the average feature size, average feature generation time, and average match time for two blobs are shown.

the other hand, all features exhibit an interesting anomaly that only our fusion technique handles well. Similarity scores close to the the lower range do *not* seem to indicate that the a match is extremely unlikely, which is what one might conclude after examining the higher and middle range. We speculate that these very "poor" scores often occurred between a "normal" blob, and another blob from the same object that was partially obscured by the

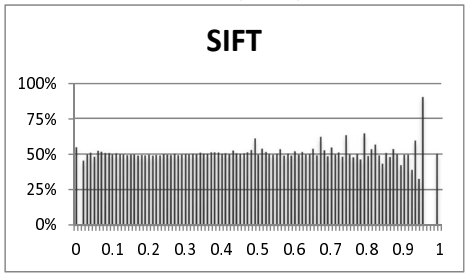


Avg Size (byte): 144
Avg Gen Time (usec): 101,314
Avg Match Time (usec): 1

Avg Size (byte): 2,160
Avg Gen Time (usec): 2,831
Avg Match Time (usec): 10,877



Avg Size (byte): 8
Avg Gen Time (usec): 1
Avg Match Time (usec): <1



Avg Size (byte): 49,584
Avg Gen Time (usec): 43,131
Avg Match Time (usec): 6,168

Figure 5.6: Match probability distributions of non-color-based feature algorithms . Under each graph, the average feature size, average feature generation time, and average match time for two blobs are shown.

edge of a frame. These very poor scores are not as indicative of a mismatch as they might seem. Finally, we can see that the SIFT feature performs poorly, with almost no trend toward definite matches with higher scores. This is likely because our blobs are quite small for SIFT features, and because SIFT features are better suited to images like buildings and graphics, and less suited to the fairly simple shapes and gradients of humans and their clothing. SIFT feature is also sensitive to view angles so it won't work well with cameras with large view angle variations.

5.4.1 Matching across all cameras

To evaluate the matching performance of the probabilistic object tracking framework, we ran a matching test across all 9 cameras. The complete dataset is divided into a training dataset and a test dataset. The training dataset size is 9 times the size of the test dataset.

We never duplicate test data from the training process. We compute the matching accuracy of the probabilistic matching method and the accuracy of using individual algorithms. The individual algorithms vote using thresholds derived to minimize errors over the training data.

We add three fusion algorithms to compare to our fusion technique. The first one is the simple majority vote (SMV). The second one is the weighted majority vote (WMV) using the feature algorithms' empirical accuracy values from the training dataset as the weights. The last one is the Weighted Majority Algorithm(WMA). As we mentioned in Section 5.3.2, WMA is sensitive to its internal parameters and suffers from the lack of online ground truth data. Instead of starting with equal weights and letting the weights change over time during the tests, we run the WMA with the training dataset and average the weights from all data samples to be the WMA's initial weights for testing. During the tests, we do not change the weights because we assume no online ground truth. We ran the WMA weights training for many rounds with different combination of suppression rate and minimum weights and calculate the accuracy with test data. Then we pick the best accuracy among all rounds to represent the WMA's performance.

Table 5.2 shows the accuracy of all feature algorithms and the comparison results of all fusion algorithms. As we can see, the SIFT feature performs worst as we expected, giving the lowest accuracy. All color-based features work better than others. All fusion algorithms have the matching performance close to the RGB Histogram which works best among all feature algorithms. The probabilistic matching method performs better than all comparison fusion algorithms, and also outperforms the best single feature algorithm, with a 1.1% absolute boost.

From Table 5.2, we can see that all individual feature algorithms work poorly, with the best accuracy of 60.9%. This is reasonable because we test over blob pairs across all cameras. In our setup, each camera has unique illumination conditions, view angles, and view distances to moving objects. Mixing blob pairs from all cameras undoubtedly won't favor any feature algorithm in our test.

SIFT	50.4%
Height/Width Ratio	50.9%
Contour Histogram	52.1%
Harris Corner	52.8%
Vertical Gradient	57.6%
Intensity Histogram	58.1%
HSV Histogram	58.7%
RG Chromaticity	59.5%
Auto Correlogram	60.5%
RGB Histogram	60.9%
SMV	60.4%
WMV	60.6%
WMA	60.7%
Probabilistic Matching	62.0%

Table 5.2: Matching accuracy of all feature algorithms and fusion algorithms across all cameras. The result is based on 7.6 million cross-camera blob pairs. 9/10 of the blob pairs are used as the training dataset, while the rest 1/10 constitute the test dataset. The accuracy values are sorted separately in ascending order. The accuracy of all fusion algorithms are close to the best feature algorithm, but only the Probabilistic Matching method outperforms the best feature algorithm.

5.4.2 Matching between camera pairs

Feature algorithms will work better if they are trained and tested separately for each individual pair of cameras, because using data only from the same camera pair will eliminate lots of environmental variations that decrease the matching performance. At the same time, we expect the fusion algorithms should achieve better performance boost since only votes from same pairs of cameras are aggregated.

We performed another test to isolate this effect. This time the training and test dataset are divided in sub-groups so that data samples in each group are from the same pair of cameras. The results are shown in Table 5.3. The first column is the IDs of the pair of cameras under test. The second column shows the algorithm in the all features that has the best matching accuracy. The 3th to 6th columns show the matching accuracy of the 4 fusion algorithms, SMV, WMV, WMA and Probabilistic Matching respectively. Since we are averaging results from multiple test runs, it is not fair to use different parameter settings for WMA in each round. So we set up the WMA to use a fixed suppression_rate of 0.95,

and the minimum_weights is set to 0.01. These parameters come from our experiences and are likely to produce good matching performance. The last two columns are the absolute and relative performance boost of the probabilistic matching to the best single algorithm. The absolute performance boost is the absolute difference of the matching accuracy. The relative performance boost is the value of the absolute boost divided by the the error rate (1 - accuracy) of the best single algorithm. A positive boost value means our probabilistic fusion method works better than the comparison best single algorithm. Figure 5.7 shows the matching accuracy in ascending order by the best feature algorithms.

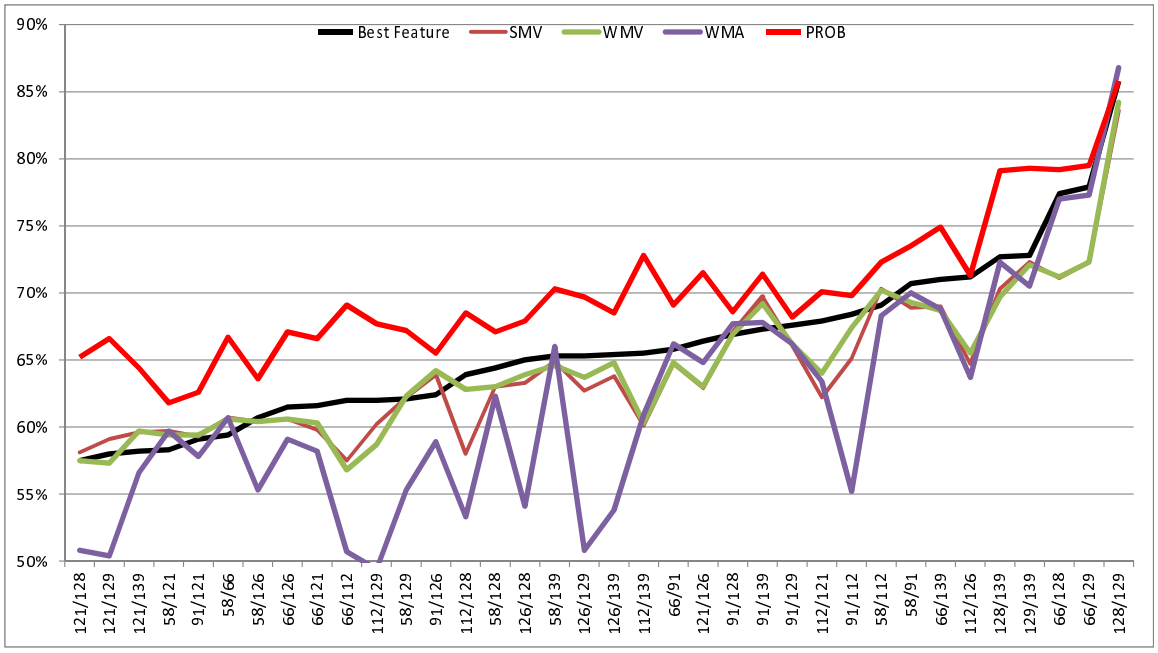


Figure 5.7: Sorted matching accuracy for matching between camera pairs. The graph is sorted in ascending order by the accuracy of the best feature algorithm for each camera pair. The probabilistic matching method (red lines) almost alway outperforms the best feature algorithms (black lines) and the other comparison fusion algorithms.

As we expected, the best feature algorithms from all camera pairs are all color-based algorithms. However, the best feature algorithm of each camera pair is different, as shown in Figure 5.8. In most cases, SMV and WMV's performance are close to the best feature algorithms, sometimes a little better, other times a little worse. WMA's performance varies a lot. In several cases, it works very well (e.g. Cam Pair 128/129), but often its accuracy is

Cam Pair	Best Ft/Acc	SMV	WMV	WMA	PROB	Abs Boost	Rel Boost
58/66	AC/59.4%	60.7%	60.6%	60.7%	66.7%	7.3%	17.9%
58/91	HSV/70.7%	68.9%	69.3%	70.0%	73.5%	2.8%	9.7%
58/112	RGC/69.1%	70.3%	70.2%	68.3%	72.3%	3.2%	10.5%
58/121	RGC/58.3%	59.7%	59.4%	59.7%	61.8%	3.5%	8.5%
58/126	VG/60.7%	60.4%	60.4%	55.3%	63.6%	2.9%	7.4%
58/128	VG/64.4%	63.0%	63.0%	62.3%	67.1%	2.7%	7.6%
58/129	VG/62.1%	62.2%	62.3%	55.3%	67.2%	5.0%	13.3%
58/139	HSV/65.3%	64.9%	64.6%	66.0%	70.3%	5.0%	14.5%
66/91	RGB/65.8%	64.7%	64.8%	66.2%	69.1%	3.3%	9.6%
66/112	RGC/62.0%	57.5%	56.8%	50.7%	69.1%	7.1%	18.7%
66/121	RGC/61.6%	59.8%	60.3%	58.2%	66.6%	5.0%	13.1%
66/126	RGC/61.5%	60.6%	60.6%	59.1%	67.1%	5.5%	14.4%
66/128	RGB/77.4%	71.1%	71.2%	77.0%	79.2%	1.8%	7.8%
66/129	RGB/77.9%	72.3%	72.3%	77.3%	79.5%	1.6%	7.3%
66/139	HSV/71.0%	69.0%	68.7%	68.8%	74.9%	3.9%	13.4%
91/112	RGC/68.4%	65.1%	67.4%	55.2%	69.8%	1.4%	4.3%
91/121	RGB/59.1%	59.3%	59.4%	57.8%	62.6%	3.5%	8.6%
91/126	VG/62.4%	63.9%	64.2%	58.9%	65.5%	3.1%	8.2%
91/128	VG/66.9%	67.0%	66.9%	67.7%	68.6%	1.7%	5.0%
91/129	VG/67.6%	66.1%	66.2%	66.2%	68.2%	0.6%	1.9%
91/139	RGB/67.3%	69.7%	69.2%	67.8%	71.4%	4.1%	12.6%
112/121	RGC/67.9%	62.2%	64.0%	63.4%	70.1%	2.1%	6.6%
112/126	RGC/71.2%	64.7%	65.5%	63.7%	71.3%	0.1%	0.3%
112/128	AC/63.9%	58.0%	62.8%	53.3%	68.5%	4.6%	12.7%
112/129	RGC/62.0%	60.2%	58.7%	49.4%	67.7%	5.7%	15.1%
112/139	RGB/65.5%	60.1%	60.3%	60.9%	72.8%	7.3%	21.0%
121/126	RGB/66.4%	62.9%	63.0%	64.8%	71.5%	5.1%	15.2%
121/128	RGB/57.5%	58.1%	57.5%	50.8%	65.2%	7.6%	18.0%
121/129	RGB/58.0%	59.1%	57.3%	50.4%	66.6%	8.6%	20.5%
121/139	RGB/58.2%	59.6%	59.7%	56.6%	64.4%	6.2%	14.9%
126/128	RGC/65.0%	63.3%	63.9%	54.1%	67.9%	2.9%	8.1%
126/129	RGC/65.3%	62.7%	63.7%	50.8%	69.7%	4.4%	12.8%
126/139	RGB/65.4%	63.8%	64.8%	53.8%	68.5%	3.1%	8.8%
128/129	RGB/85.8%	83.6%	84.2%	86.8%	85.8%	0.0%	0.0%
128/139	RGB/72.7%	70.3%	69.7%	72.3%	79.1%	6.4%	23.5%
129/139	RGB/72.8%	72.3%	72.1%	70.5%	79.3%	6.4%	23.6%

Table 5.3: Camera-pairwise matching accuracy of all feature algorithms. For each pair of cameras, 9/10 of all blob pairs are used as training dataset, while the rest 1/10 constitute the test dataset. We show the best feature algorithm and its accuracy, as well as the accuracy of 4 fusion algorithms. The Abs/Rel Boost are the performance boost of the probabilistic matching method over the best feature algorithm.

closer to 50%–random chance. Without online groundtruth, fixed weights from the average weights of training data are oftentimes not suitable for the test data. For all camera pairs under test, our probabilistic matching method works better than the two comparison fusion algorithms SMV and WMV. It also always works better than the best single algorithm except one camera pair 128/129. The absolute performance boost reaches 8.6%. The relative performance boost reaches 23.6%. The average absolute performance boost is 4.0%. And the average relative performance boost is 11.5%. Both are better than matching across all cameras in the previous experiment. In the only test case where our probabilistic matching method works no better than the best single algorithm, the two cameras 128 and 129 share very similar views with same view distance and illumination conditions, as shown in Figure 5.4. And in this test case, the best single algorithm, the RGB color histogram works very well with matching accuracy more than 85%. These results suggest that our probabilistic matching algorithm are usually beneficial in boosting object matching performance, especially with camera pairs that have large variation in camera views where no single feature algorithm achieves very good results. In these cases, our probabilistic matching method, will increase the cross camera object matching accuracy.

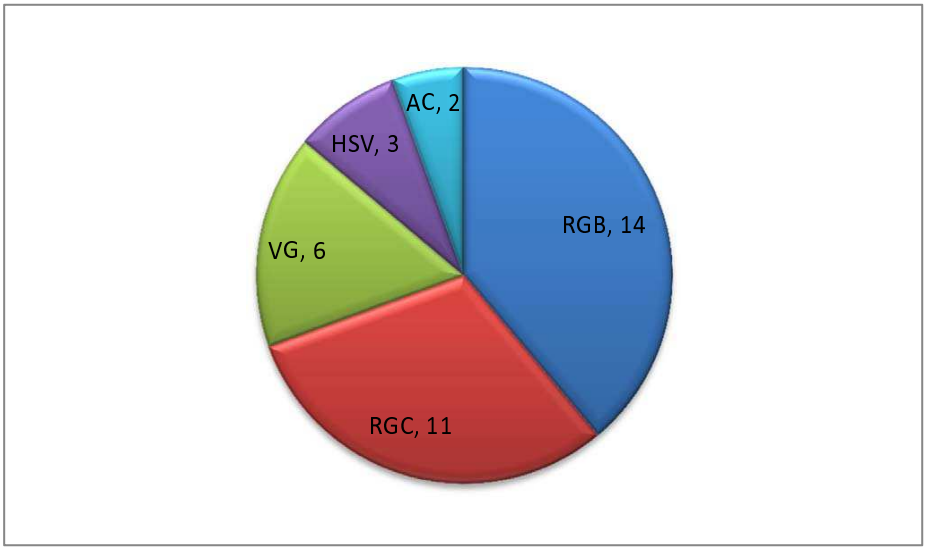


Figure 5.8: Best features in matching between all 36 camera pairs. There is no single feature algorithm works best all the time. RGB color histogram works well in general, but over half of time it is outperformed by other features.

Table 5.4 shows the average absolute and relative performance boost of the probabilistic matching method over 5 runs with randomly generated training and test datasets. In all camera pairs, our probabilistic method always boosts the matching performance up to 9.6% absolute and 22.9% relative.

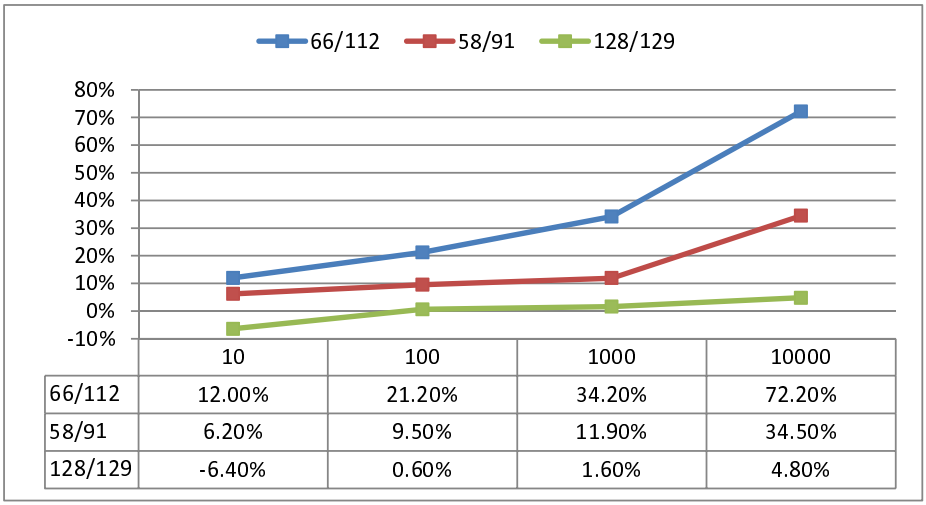
	66	91	112	121	126	128	129	139
58	7.1%/ 17.6%	2.8%/ 9.5%	3.3%/ 10.4%	3.0%/ 7.2%	2.9%/ 7.5%	2.3%/ 6.6%	4.7%/ 12.8%	5.3%/ 15.2%
66		3.2%/ 9.5%	8.0%/ 21.2%	4.9%/ 13.1%	5.7%/ 14.8%	1.8%/ 8.0%	1.6%/ 7.5%	3.9%/ 13.9%
91			1.2%/ 3.7%	3.6%/ 8.7%	3.2%/ 8.6%	1.6%/ 4.8%	0.4%/ 1.2%	4.4%/ 13.5%
112				3.6%/ 11.2%	1.4%/ 4.6%	5.5%/ 15.0%	7.6%/ 19.7%	7.6%/ 22.2%
121					5.4%/ 16.0%	7.3%/ 17.3%	9.6%/ 22.8%	6.1%/ 15.0%
126						2.8%/ 7.9%	4.1%/ 11.6%	3.4%/ 9.8%
128							0.1%/ 0.6%	5.6%/ 21.1%
129								6.2%/ 22.9%

Table 5.4: Average absolute/relative matching accuracy boost of the probabilistic matching algorithm. Data are average of 5 test runs with different randomly generated training/test datasets. In all tests, the probabilistic fusion method is able to boost object matching performance.

5.4.3 Number of Bins

The probability distributions are counters over a set of bins with equal width. In the previous experiments, the default number of bins is set to 100. In this section, we study the effect of changing the number of bins. We select 3 different camera pairs from the previous experiments that our fusion algorithm works very well (66/112), average (58/91), and not so well (128/129) based on the relative accuracy boost value. Cameras 66 and 112 see very different view angle/distance and object moving pattern. Cameras 58 and 91 capture the front and back of moving persons respectively. Cameras 128 and 129 have overlapping views

and similar view angles and distance. The results are shown in Figure 5.9.



	10	100	1000	10000
66/112	12.00%	21.20%	34.20%	72.20%
58/91	6.20%	9.50%	11.90%	34.50%
128/129	-6.40%	0.60%	1.60%	4.80%

Figure 5.9: Average relative matching accuracy boost of the probabilistic matching algorithm with different number of bins. Data are average of 5 test runs with different randomly generated training/test datasets. The results show a trend of increasing performance boost with larger number of bins.

From the test results, we see a trend that the probabilistic matching method works better with larger number of bins. Even with camera pairs that share similar views, increasing number of bins still achieves better matching performance. However, larger number of bins means more data to store the probability matching tables. With bin count $= n$, the total number of storage require for each feature algorithm is $(2 * n + 2) * sizeof(int)$. It requires about 800K bytes storage to store 10 probability match tables with bin count of 10000 (with 4 bytes integers). So the number of bins are bounded by the available capacity of on-board storage on each camera node minus the amount needed for feature computation and matchings. Also, depending on the number of training data, the bin count should be small enough so that each bin has enough number of matching and unmatching samples. Otherwise, if the bin count is too large comparing to the size of training dataset, the counter values of lots of bins will be very small or even zero, creating an overfitting problem. The training dataset has to include groundtruth data which usually requires manual object identification and will not be arbitrarily large. In real-world applications, a suitable bin

count will have to take into account both available storage capacity and training data.

5.4.4 Object Matching

As we mentioned earlier, tracking objects in a single camera is relatively easy. Due to temporal adjacency, objects in consecutive image frames can be tracked with very high accuracy. When tracking objects across cameras, if the object descriptors contain information from more than one frame, the object matching can achieve better accuracy by leveraging the extra information in descriptor comparison.

In all previous experiments, each object contains only one image blob. We run another test to measure the matching accuracy improvement of using objects containing multiple images blobs. We traverse the whole dataset. In each camera, we combine every n blobs in consecutive image frames into a single object representation. Then we divide all objects into training and test datasets and run the cross-camera object matching experiments. When matching two objects, individual feature algorithms compute the matching results of all $n*n$ blob pairs between two objects. All $n*n$ matching results are compared to the feature algorithm's threshold to generate a "yes" or "no" vote on whether the two blobs match. Then the majority vote of the $n*n$ blob matching votes are used as the final vote for object matching. The probabilistic matching method, on the other hand, averages the derived matching probabilities from all $n*n$ blob pairs. In this experiment, we use half of the objects in training, and the other half in testing. Table 5.10 show the experiment results when $n = 1, 3, 5$ respectively.

From the table we can see that when the objects contain more frames, the matching accuracy of the best feature increases. At the same time, the probabilistic matching algorithm continues to improve performance. The experiment confirms that the high tracking accuracy in single camera can be utilized to improve cross-camera objects tracking.

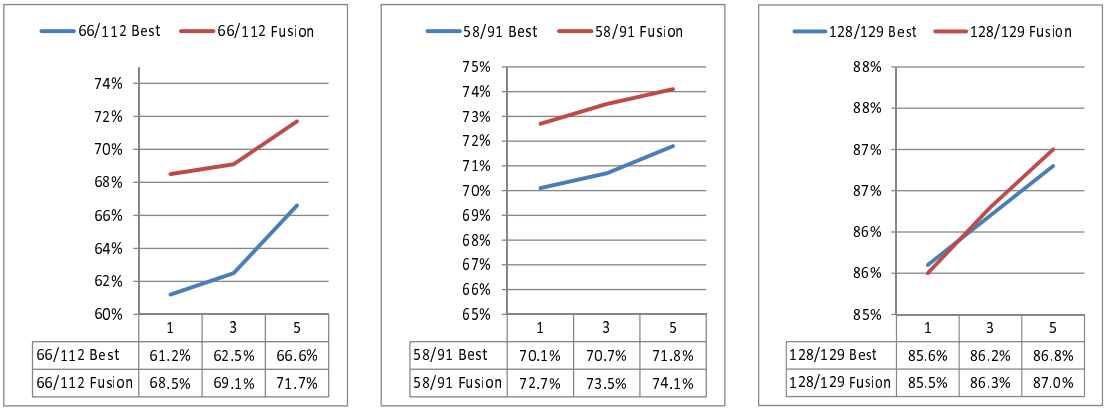


Figure 5.10: Object matching accuracy boost of the probabilistic matching algorithm with different frame counts. Here are average results of 5 runs from 3 different camera pairs. When the objects contain more frames, the matching accuracy of the best feature, as well as the probabilistic fusion algorithm, both increase.

5.4.5 Resource Usage

In the object matching experiments, the features from independent frames are compared independently. However, if object features are provided with efficient aggregation methods, features can be combined to save the bandwidth needed to transmit the object features. For example, most histogram based features can be averaged. Interest points based features, e.g. Harris Corner and SIFT, can reduce the total number of interest points by only keeping the points that repeat most often. Interest points with high repeatability are often called "stable" features.

In a visual sensor network, object tracking requires both feature descriptor detection and feature matching. Both processes consume the computational power of the camera nodes as well as storage and network resources. For all features we have used in our experiments, we list the average feature size, feature generation time and feature matching time in Figure 5.5 and 5.6.

Color based and gradient based features usually have to traverse all pixels in the image blob. The higher resolution and frame rate the images are, the more cpu cycles are needed to generate the feature descriptors. Also, depending on the algorithm complexity, each feature requires different amount of time to generate the features for an image blob. In

our experiments, the Auto Correlograms, Contour Histograms and SIFT features are the most expensive to generate. These features either involve complex algorithms or require additional processing of neighboring pixels when traversing the pixels in an image blob.

Histogram based features have a fixed descriptor size and the matching often requires little effort. However, the size of multi-dimensional histograms, such as HSV and RGB histograms, can become very large when the bin count of the histogram increases. On the other hand, interest points based features have descriptors with varied sizes, and the matching computation is quadratic in the size of the interest point list. In our experiments, SIFT features are generally larger in feature size than the other features. This is because the SIFT features of each image blob contains a list of interest points, and each point is associated with a descriptor of 128 doubles. The average count of the SIFT feature key points for our test image blobs from the D-Link cameras is about 50. If higher resolution cameras are used, more SIFT features can be detected in an image blob, and we will need more storage space for longer list of SIFT features. Usually, we get more Harris Corners than SIFT features from an image blob, therefore, although individual Harris Corners occupy less storage space than SIFT features, it takes the longest time to match two image blobs with Harris Corners due to their large list size.

Finding the proper size of feature descriptors is crucial to limiting the resource usage within the system constraints while achieving the best tracking performance. Table 5.5 shows the storage requirements of HSV histogram and Vertical Gradients. Their matching accuracy between camera 58 and 91 is also listed.

The HSV histogram is 3-dimensional, so the descriptor size increase in cubic. The D-Link cameras used in our experiments do not have vivid color so large bin counts create overfitting. The best matching accuracy is achieved when bin count is 4. The Vertical Gradient is a single dimension histogram and its descriptor size increases linearly to the bin count. Based on the relatively small image blob size in our experiments, VG achieves its top matching performance with bin size set to 8.

A simple fusion test on HSV and VG reveals that the fusion works best with HSV2 and

Feature	Bin Size	Bytes per descriptor	Matching accuracy
HSV	2	32	71.9%
	4	256	73.9%
	8	2,048	70.3%
	16	16,384	71.3%
VG	2	8	63.4%
	4	16	63.7%
	8	32	64.7%
	16	64	64.6%

Table 5.5: Storage size and matching accuracy of HSV histogram and Vertical Gradient. The matching accuracy results are the average of 5 test runs on camera pair 58/91. Features with different internal parameter settings require different storage space. It is not always better in matching accuracy to have more expensive settings.

VG8, or HSV4 and VG16, as shown in Table 5.6. Considering the storage size of the two features, it is preferable to use HSV2 and VG8 in practice.

	VG2	VG4	VG8	VG16
HSV2	73.4%	73.3%	**73.8%**	73.7%
HSV4	73.5%	73.4%	73.5%	**73.8%**
HSV8	70.8%	70.3%	71.1%	71.3%
HSV16	71.4%	71.1%	71.5%	71.8%

Table 5.6: Probabilistic fusion accuracy of HSV histogram and Vertical Gradient with varying bin size. The matching accuracy results are the average of 5 test runs on camera pair 58/91. Fusions of features and their variants yield different matching performance.

Chapter 6

Related Work

6.1 Camera Calibration

In computer vision, camera calibration is the process to find the camera parameters that affect the production of camera images. The camera model has two types of parameters. The intrinsic parameters describe the cameras internal attributes such as focal length, image transformation, principle points, etc. The extrinsic parameters denote the coordinate system transformation from the world 3D coordinates to the camera coordinates. There parameters are represented by the camera matrix. In this thesis, we study camera calibration of distributed camera networks to find the extrinsic parameters of all cameras so that cameras are able to compute the relative coordinate transformation between each. These cross camera transformations allow cameras to be precisely localized.

There have been many published works dealing with the calibration of single and multi-camera systems. Svoboda et al. published their work on multi-camera self-calibration [46]. The system computes both intrinsic and extrinsic parameters for an arbitrary number (>3) of cameras with no prior knowledge. The user is required to to capture a relatively large number of images with a laser pointer in the dark(or other easily detectable object) as input data. Devarajan and Radke presented a more distributed method of camera calibration in their paper [7]. Although in their method the calibration takes place in a distributed

fashion, hand labelling of the camera "vision graph" and feature point correspondences is required. Similar to Devarajan and Radke, Mantzel et al. proposed a method to perform camera localization in a distributed fashion, given known feature correspondences [31]. They introduced the notion of a camera "microcluster", or a set of cameras with large set of overlapping visible feature points.

6.2 Distributed Event Processing

This thesis presents a sensornet programming model, including a declarative language to express events with spatio-temporal constraints and an efficient event detection framework to provide runtime support for the programming model.

Many approaches have been proposed to provide programming abstraction and communication models for sensor networks. Our work differentiates itself from existing approaches in its focus on removing any sensor-oriented aspects of the programming abstractions.

Our programming model resembles the database-based approaches, such as TinyDB [30] and Cougar [52], which express sensor data of interest in a network-independent way using SQL-style queries. Comparatively, our programming model is built in a similar way of those in the active database area [6, 13, 36] and is tailored to allow the expression of events with complex spatio-temporal constraints.

EnviroSuite [28] is an environmentally immersive programming framework which uses object-based model to abstract interactions between physical objects and the runtime environment. Our DCP framework is event-based and focuses on expressing hierarchical constraints. Hood [50], Abstract Regions [49], and Regiment [34] present sensor programming models based on groups of nodes defined by their physical proximity or network topology. One may view these groups as sets of nodes that follow constraints that may be laid out in our declarative language. We believe that event constraints represent a similar level of abstraction with the added benefit of removing the need to consider nodes at all when specifying application behaviour. Event constraints may express "a red ball within 10 meters

of a blue ball," while regions based on node membership cannot, if the node may detect objects at a distance.

The DCP framework decouples event location from node location by extending GHTs [40] as the address mechanism. GHTs were introduced to provide Data-Centric Storage (DCS) [39] for wireless sensor networks. In DCS, events are hashed to geographic locations by event names and stored at the closest node to the hashed location. GHT uses GPSR [21] to route packets to the destination locations. DCP extends GHTs to *Regional GHTs* which preserve spatial locality in events and allow local operation despite large-scale network partitions.

6.3 Object Tracking, Decision Fusion and Machine Learning

Object tracking is one of the most common applications in visual sensor networks. Cross camera object tracking is often realized by the process of object reidentification. Many previous literatures on object reidentification [19, 10, 2, 47, 43, 24, 27, 33] concentrate on optimizing certain object features to the targeting application scenarios, or finding a better algorithm to utilize the object features.

Our approach for object tracking doesn't require object features that always work well, nor focus on a single feature type. Instead, we accept the fact that in real world applications, any object feature may fail at times. However, by combining multiple object features, our fusion technique allows the performance of object reidentification to be better than or retained to the best working features.

Guo et al. [14] address the cross-camera vehicles matching problem with multiple features, such as lines, points and regions. In their experiments, each object contains a collections of image patches. They use the weighted sum of the correlation scores from all images patches to compute the match score. Sun et al. [45] uses multidetector fusion for vehicle reidentification. Their fusion technique is called the linear opinion pool [9] which is quite similar to the weighted sum method.

In our fusion framework, the fusion results are probabilities that two objects match. The

linear opinion pool or log-opinion pool [3] are traditional fusion approaches for probabilities as well as distances. Log-opinion pool are often used when the combining independent sources. Other fusion techniques such as voting [51] or ranking [17] are used for classifier fusion. Object matching can be treated as a classification problem if the matching happens between an object sample and a collection of objects. In this case, AdaBoost [11] is often used to boost the performance of weak classifiers. In this thesis, we don't assume the object tracking is limited to objects that have been seen by the camera network, and always expect that the object matching can happen with new objects. Therefore, we don't treat the object matching as a classification problem and take the probabilistic approach to boost the matching performance.

Machine learning techniques are often used to learn complex patterns from empirical data and make intelligent decisions on new test cases. In cross camera object tracking, the similarity values from multiple features algorithms can be fed into learners to build decision models. We have used Waffles [12], an open-source machine learning toolkit, to run experiments on our datasets. By using different machine learning algorithms, such as decision tree, neural networks, etc, we get mixed results depending on different internal parameter settings. Similar to our experience with the online weighted majority algorithm, it is important to provide a good selection of internal parameters to achieve good matching performance. We believe our probabilistic method and machine learning techniques are complementary in practice. For example, multiple machine learning techniques can be used at the same time to predict the overall similarities, then their matching probabilities and be used in our probabilistic matching method to boost the matching performance again. On the other hand, instead of averaging the matching probabilities from multiple feature algorithms, the matching probabilities can be used as the input of machine learning algorithms to build decision models to learn the interesting relations among these feature algorithms. As we mentioned in Section 5.3.3, applying machine learning techniques on the probabilities will sometimes help us gain a little more boost on the matching performance but at the cost of more computational resources.

Chapter 7

Conclusions

We believe visual sensor network is an exciting research field with real-world applications and research challenges. Visual sensor network is an interdisciplinary research area involving embedded system, image processing, network communication and distributed systems. In this thesis, we designed and implemented several techniques to support complex vision tasks in visual sensor networks. These tasks include camera calibration, complex event detection and cross-camera object tracking.

First, we have provided an distributed solution, called Lighthouse, to the problem of sensor localization and multi-camera calibration. Lighthouse builds GHTs incrementally, avoiding the need for localization infrastructure or special hardware.

Second, programming sensornets is well recognized as a hard problem, and data-centric techniques have emerged as a way of taming the associated complexity. In this thesis, we have described an area in which existing sensornet programming paradigms have not yet embraced a data-centric approach. We have filled that gap with a distributed constraint processing engine for constructing complex events from subevents. Details of the sensor network are abstracted away so that constraints may be expressed directly between events, rather than through an intermediate abstraction based on the node location or attributes. We believe this separation is particularly important for future sensornets that will integrate more node that sense events at a distant. Despite the increased level of abstraction, our

approach to distributed constraint processing is efficient, scalable, and fault-tolerant because it uses local resources to process local events.

Last, we presented a probabilistic object matching framework to fuse multiple feature matching results using historical match probability distributions of the feature algorithms. In cross-camera object tracking, we cannot expect any single object feature to work well all the time for object matching. Multiple object features based on different aspects of object appearance can be employed together to achieve more consistent matching performance. Fusion of multiple features can benefit from the confidence scores of the matching results from feature matchers. Confidence scores are more reliable and easier to implement if computed from empirical data. Our framework frees feature algorithms from making explicit binary matching votes and matching confidences. Our evaluation of the probabilistic object matching framework shows it outperforms the comparison fusion techniques and is able to tolerate poorly designed feature algorithms to almost always maintain a better accuracy than the best feature algorithm in the pool.

The above techniques have architectural, system-level and algorithmic components, spanning from signal processing, communication protocols to decision fusion. We believe that this thesis work will provide a good basis for interesting future research in visual sensor networks.

Bibliography

[1] P. T. Baker and Y. Aloimonos. Calibration of a multicamera network. In *Proceedings of Omnivis 2003: Workshop on Omnidirectional Vision and Camera Networks*, Madison, Wisconsin, June 2003.

[2] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(4), 2002.

[3] J.A. Benediktsson and P.H. Swain. Consensus theoretic classification methods. *Systems, Man and Cybernetics, IEEE Transactions on*, 22(4), jul/aug 1992.

[4] Michael Bramberger, Andreas Doblander, Arnold Maier, Bernhard Rinner, and Helmut Schwabach. Distributed embedded smart cameras for surveillance applications. *Computer*, 39(2), 2006.

[5] Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, and Steven A. Shafer. Easyliving: Technologies for intelligent environments. In *HUC*, 2000.

[6] Sharma Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Data Knowledge Engineering*, 14(1), 1994.

[7] Dhanya Devarajan and Richard J. Radke. Distributed metric calibration of large camera networks. In *in Proc. 1st Workshop on Broadband Advanced Sensor Networks*, 2004.

[8] D-Link DCS-900 internet camera. http://www.dlink.com/products/?pid=DCS-900.

[9] K R Farrell, R P Ramachandran, and R J Mammone. An analysis of data fusion methods for speaker verification. *Proceedings of the 1998 IEEE International Conference on Acoustics Speech and Signal Processing*, 2, 1998.

[10] Andras Ferencz, Erik G. Learned-Miller, and Jitendra Malik. Learning hyper-features for visual identification. In *NIPS*, 2004.

[11] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting, 1997.

[12] Michael S. Gashler. Waffles: A machine learning toolkit. *Journal of Machine Learning Research*, MLOSS 12, July 2011.

[13] Stella Gatziu and Klaus R. Dittrich. Detecting composite events in active database systems using petri nets. In *RIDE-ADS*, 1994.

[14] Yanlin Guo, Steven C. Hsu, Harpreet S. Sawhney, Rakesh Kumar, and Ying Shan. Robust object matching for persistent tracking with heterogeneous features. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(5), 2007.

[15] C. Harris and M. Stephens. A Combined Corner and Edge Detection. In *Proceedings of The Fourth Alvey Vision Conference*, 1988.

[16] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[17] Tin Kam Ho, Jonathan J. Hull, and Sargur N. Srihari. Decision combination in multiple classifier systems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(1), 1994.

[18] Jing Huang, Ravi Kumar, Mandar Mitra, Wei-Jing Zhu, and Ramin Zabih. Image indexing using color correlograms. In *CVPR*, 1997.

[19] Anil K. Jain, Yu Zhong, and Sridhar Lakshmanan. Object matching using deformable templates. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(3), 1996.

[20] D. R. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web.

[21] Brad Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*. ACM Press, 2000.

[22] Arezou Keshavarz, Ali Maleki Tabar, and Hamid Aghajan. Distributed vision-based reasoning for smart home care. In *Proc. Workshop on Distributed Smart Cameras (DSC 2006)*, 2006.

[23] A. J. Lacey, N. Pinitkarn, and N. A. Thacker. An Evaluation of the Performance of RANSAC Algorithms for Stereo Camera Calibration. In *Proceedings of The Eleventh British Machine Vision Conference*, September 2000.

[24] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Semi-local affine parts for object recognition. In *In BMVC*, 2004.

[25] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm, 1992.

[26] David Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2), 2004.

[27] David G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, 1999.

[28] Liqian Luo, Tarek F. Abdelzaher, Tian He, and John A. Stankovic. Envirosuite: An environmentally immersive programming framework for sensor networks. *ACM Trans. Embedded Comput. Syst.*, 5(3), 2006.

[29] Y. Ma, S. Soatto, J. Kosecká, and S. S. Sastry. *An Invitation to 3-D Vision: From Images to Geometric Modeling*. Springer-Verlag, 2004.

[30] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. In *OSDI*, 2002.

[31] William Mantzel, Richard Baraniuk, and Hyeokho Choi. Distributed Camera Network Localization. In *Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, November 2004.

[32] Birgitta Martinkauppi, Maricor Soriano, and Matti Pietikäinen. Comparison of skin color detection and tracking methods under varying illumination. *J. Electronic Imaging*, 14(4), 2005.

[33] Jiri Matas, Ondrej Chum, Martin Urban, and Tomás Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image Vision Comput.*, 22(10), 2004.

[34] Ryan Newton, Greg Morrisett, and Matt Welsh. The regiment macroprogramming system. In *IPSN*, 2007.

[35] The Network Simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[36] N.W. Paton and O. Diaz. Active Database Systems. *ACM Computing Surveys*, 1(31):63–103, 1999.

[37] Nissanka Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system.

[38] Mohammad Rahimi, Rick Baer, Obimdinachi I. Iroezi, Juan C. Garcia, Jay Warrior, Deborah Estrin, and Mani Srivastava. Cyclops: in situ image sensing and interpretation in wireless sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, New York, NY, USA, 2005. ACM.

[39] Sylvia Ratnasamy, Brad Karp, Scott Shenker, Deborah Estrin, Ramesh Govindan, Li Yin, and Fang Yu. Data-centric storage in sensornets with ght, a geographic hash table. *Mob.Netw.Appl.*, 8(4), 2003.

[40] Sylvia Ratnasamy, Brad Karp, Li Yin, Fang Yu, Deborah Estrin, Ramesh Govindan, and Scott Shenker. GHT: a geographic hash table for data-centric storage. In *WSNA*

'02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications. ACM Press, 2002.

[41] Anthony Rowe, Adam Goode, Dhiraj Goel, and Illah Nourbakhsh. Cmucam3: An open programmable embedded vision sensor. *Tech. Rep. CMU-RI-TR-07-13, Carnegie Mellon Robotics Institute*, 2007.

[42] Gyula Simon, Miklós Maróti, Ákos Lédeczi, György Balogh, Branislav Kusy, András Nádas, Gábor Pap, János Sallai, and Ken Frampton. Sensor network-based counter-sniper system. In *Proc. SenSys*, 2004.

[43] Josef Sivic, Frederik Schaffalitzky, and Andrew Zisserman. Object level grouping for video shots. *International Journal of Computer Vision*, 67(2), 2006.

[44] John R. Smith and Shih-Fu Chang. Tools and techniques for color image retrieval. In *Storage and Retrieval for Image and Video Databases (SPIE)*, 1996.

[45] Carlos C. Sun, Glenn S. Arr, Ravi P. Ramachandran, and Stephen G. Ritchie. Vehicle reidentification using multidetector fusion. *IEEE Transactions on Intelligent Transportation Systems*, 5(3), 2004.

[46] D. Svoboda, T. Martinec and T. Pajdla. A convenient multi-camera self-calibration for virtual environments. *PRESENCE: Teleoperators and Virtual Environments*, 14(4), August 2005.

[47] Shimon Ullman, Erez Sali, and Michel Vidal-Naquet. A fragment-based approach to object representation and classification. In *IWVF*, 2001.

[48] M. Valera and S.A. Velastin. Intelligent distributed surveillance systems: a review. *IEE Proceedings - Vision, Image and Signal Processing*, 152(2), April 2005.

[49] Matt Welsh and Geoff Mainland. Programming sensor networks using abstract regions. In *NSDI*, 2004.

[50] Kamin Whitehouse, Cory Sharp, David E. Culler, and Eric A. Brewer. Hood: A neighborhood abstraction for sensor networks. In *MobiSys*, 2004.

[51] L. Xu, A. Krzyzak, and C.Y. Suen. Methods of combining multiple classifiers and their applications to handwriting recognition. *Systems, Man and Cybernetics, IEEE Transactions on*, 22(3), may/jun 1992.

[52] Yong Yao and Johannes Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31(3), 2002.