Distributed Calibration of Smart Cameras

John Jannotti
Department of Computer Science
Brown University
jj@cs.brown.edu

Jie Mao
Department of Computer Science
Brown University
jmao@cs.brown.edu

Abstract

Localization in sensornets determines the location of sensor nodes, and allows applications to make geographically sensitive queries. Smart camera networks must not only be localized, but *calibrated*. Calibration goes beyond localization to include orientation and position information that is sufficiently fine-grained to allow fusion between overlapping camera views.

This paper introduces Lighthouse, a distributed calibration system that allows wireless smart camera networks to obtain a unified coordinate system without manual configuration or additional hardware. The proposed 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.

1 Introduction

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*. The cameras of 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.

Calibration requires precise positions and orientations, beyond the limits of existing localization techniques. Even differential GPS or Cricket [9], each with accuracy in the centimeter range, would be unable to determine the orientation of a small camera sensor. Even small errors in orientation may result in large absolute errors when estimating the position of distant objects.

This paper presents *Lighthouse*, a distributed calibration technique that allows smart camera networks to obtain a unified coordinate system without specialized hardware. Only the cameras of the network are used. Further, Lighthouse avoids centralizing all features in order to save bandwidth and power. Lighthouse cameras obtain features locally and then use distributed matching techniques to find correspondences. These correspondences allow nodes to agree on a common reference frame.

1.1 Traditional Approach

Multi-camera geometric calibration is an active research topic [1, 10]—current solutions are centralized, usually requiring factorization of very large matrices [2, 7]. 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.

1.2 Our approach

Our approach is to find pairwise correspondences between the features observed by a camera and the features observed by another camera or group of calibrated cameras. The fundamental challenges are finding robust features that may be matched across cameras and doing so in a distributed fashion.

Lighthouse solves the distributed matching problem using local search and Geographic Hash Tables (GHTs). A Lighthouse camera begins its search for feature matches by considering the features of geographic neighbors. However, feature matches may occur across large distances. Two cameras may observe the same distant skyline, for example (See Figure 1). Geographic hash tables

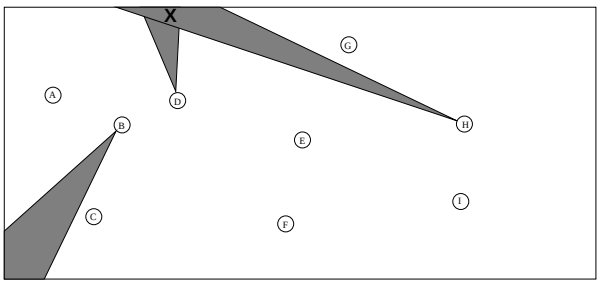

Figure 1: A small misorientation of camera H would lead to large errors in estimating the position of X, but if D and H are aware that they both observe X, they may use that information to orient precisely with respect to each other. Complicating this matching problem, cameras B and D do not observe similar features despite their proximity, while distant cameras D and H have overlapping views.

allow distant nodes to insert features into a shared table. When features "collide", the observers are notified and compute a relative transform.

GHTs presuppose the use of geographic forwarding, such as GPSR [4], which requires a shared coordinate system. Yet until correspondences are found, the nodes do not share a coordinate system. Lighthouse includes bootstrapping techniques that allow adjacent GHTs to share features without geographic routing between them. When correspondences are found, the adjacent GHTs are merged into a single, large GHT.

2 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.

2.1 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 [5] 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 describes how features can be detected without wholesale feature exchange.

2.2 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 [6] 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 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 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 2 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.

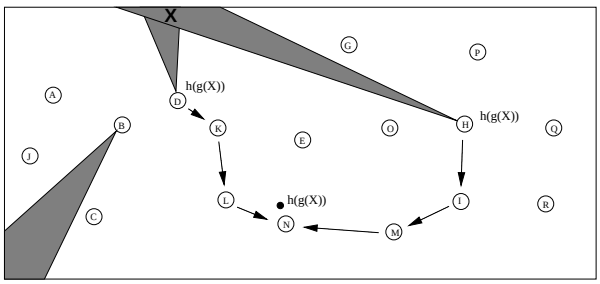

Figure 2: The feature X is observed by the two separate camera nodes. The feature is categorized through a geometric hash function, g(), and then a storage location is selected with a geographic hash, h(). Each camera routes the feature toward the designated location, where the closest node, N, stores the feature, detects matches, and informs the observers.

3.1 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 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.

3.3 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 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* [3] 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, 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.

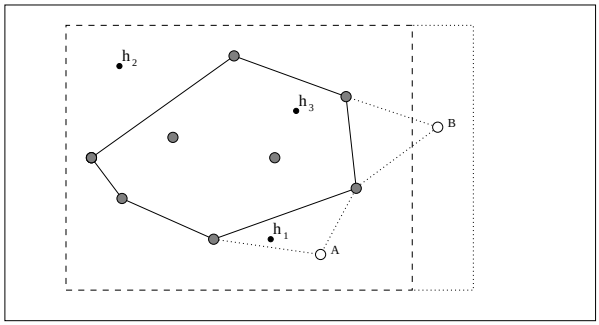

Figure 3: The group of gray nodes is a GHT, using the dashed box as its loose bounding box. To store an item, X, it is hashed repeatedly (h_1, h_2, h_3) until a locations is found within the perimeter. If node A joins the GHT, the loose bounding box need not be changed, therefore only those items, like 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.

4 Evaluation

We conduct NS [8] simulations using a simple implementation of a GHT using code from GPSR [4]. 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.

4.1 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 4 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.

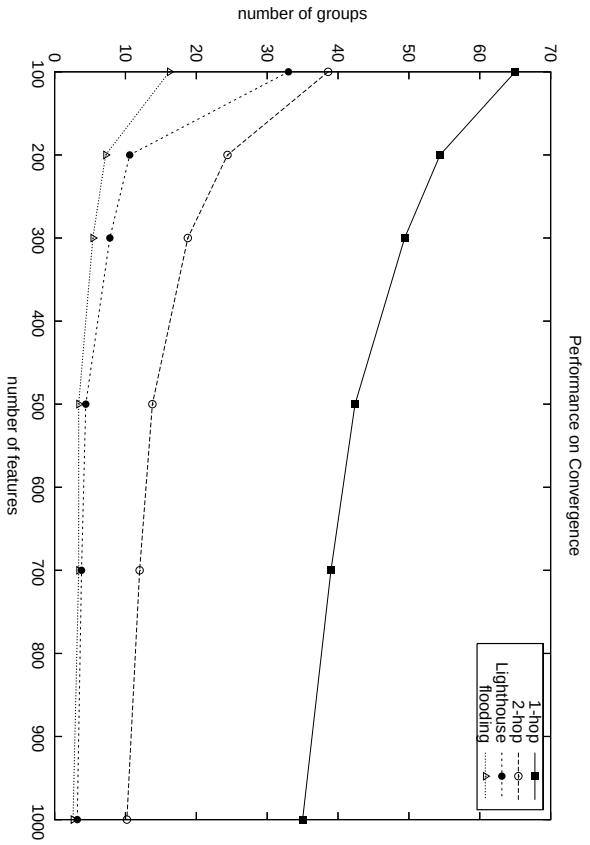

Figure 4: Lighthouse is compared to simpler 1-hop and 2-hop neighbor schemes, as well as a perfect matching scheme that floods all features to all nodes. As the feature density increases, all schemes are able to reduce the number of independent GHTs by finding shared features. At all densities Lighthouse performs significantly better than the short-range schemes. At reasonable feature densities, Lighthouse approximates complete flooding.

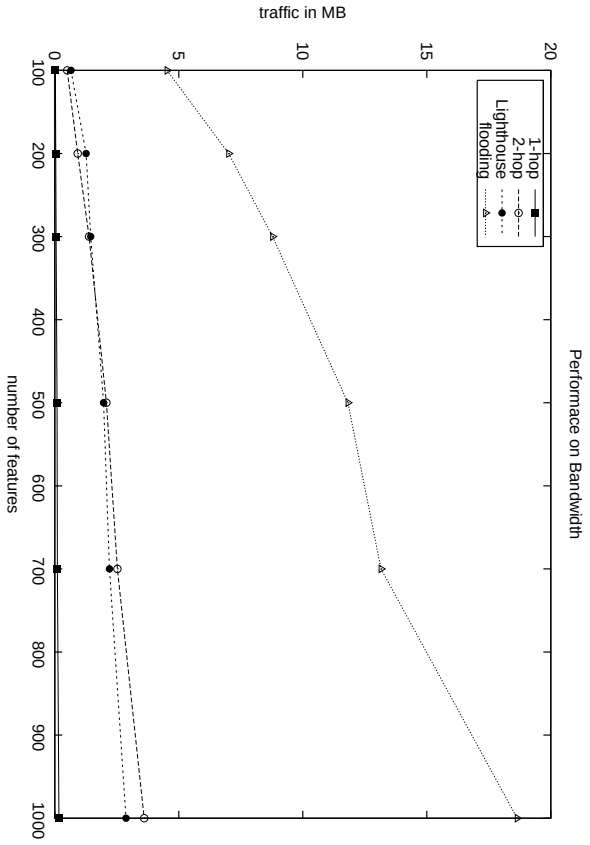

Figure 5: As feature density increases, more features are exchanged and more bandwidth is consumed. Lighthouse is able to grow as slowly as 2-hop, despite its ability to find widespread matches.

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.

4.2 Scalability

In the last section we examined Lighthouse's ability to find matches, we now consider the cost of doing so. Figure 5 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 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 ($O(n)$).

5 Conclusions

Lighthouse is a distributed solution to the problem of sensor localization and multi-camera calibration. Lighthouse builds GHTs incrementally, avoiding the need for localization infrastructure or hardware. We look forward to working with real-world data from our camera pod prototype to develop models to control error in real-world deployments.

References

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

[3] 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. In *Proc. 29th ACM Symposium on Theory of Computing*, May 1997.

[4] Brad Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Proc. ACM/IEEE MobiCom*, August 2000.

[5] 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.

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

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

[8] Ns. `http://www.isi.edu/nsnam/ns/`.

[9] Nissanka Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In *Proc. ACM/IEEE MobiCom*, August 2000.

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