Designing Smart Camera Networks using Smartphone Platforms: A Case Study

Yusuf Simonson, Robert Fowler
Renaissance Computing Institute
Chapel Hill, North Carolina 27517
Email: yusuf@renci.org, rjf@renci.org

Edgar Lobaton, Ron Alterovitz
The Department of Computer Science
The University of North Carolina at Chapel Hill
Chapel Hill, North Carolina 27517
Email: lobaton@cs.unc.edu, ron@cs.unc.edu

***Abstract*—We believe that new smartphone architectures like Android and iOS will play increasingly important roles in sensor network design. Because of this, we wish to investigate the state of development for sensor network-based applications on the Android framework. We introduce an application for Android that allows for the semi-autonomous remote control of Rovio robots. It coordinates with a sensor network of cameras to provide a live stream of camera images and predicted robot locations. Furthermore, it provides functionality for moving the robot to user-selected destinations without the need for manual control. The application acts as a straw man for working with traditional sensor network architectures, and provides important insight on some of the challenges related to sensor network development, especially on the Android platform. We outline these challenges and provide some suggestions on semantics that could alleviate development effort.**

I. Introduction

Sensor networks represent a rapidly growing field. But despite extensive research and development, the design of such systems provides a number of large challenges. As a result, deployment tends to be expensive (sometimes prohibitively so), and often requires significant engineering expertise. This is particularly true for smart camera networks for which there are no standard hardware or software platforms.

An example of such a distributed camera network is shown in Figure 1. Each camera node is capable of communicating with their neighbors in order to aggregate information. Information may also be stored in a computer server which can be used as a large database for tasks such as object recognition. Besides static cameras, there are also mobile agents that can communicate with the smart cameras in order to navigate through the environment. Finally, users can interface with the system through the use of computers or smart phone devices.

Due to the diverse range of environments (surveillance, wildlife monitoring, search and rescue [1]) in which camera networks are used, it is very important to have hardware platforms that are reliable and resilient in hostile settings. Several platforms have been developed, including CITRIC [2] and DSPcam [3], among others. They vary in computational and communication constraints due to limitations in battery life or form factor.

The software end is in many ways even more problematic. At a high level, if we consider the servers involved in collating and processing data received from the sensors as part of the architecture, then we oftentimes have non-trivial distributed programming challenges that require careful coordination and data synchronization. The server-end tends to handle clients in parallel, further compounding coordination issues.

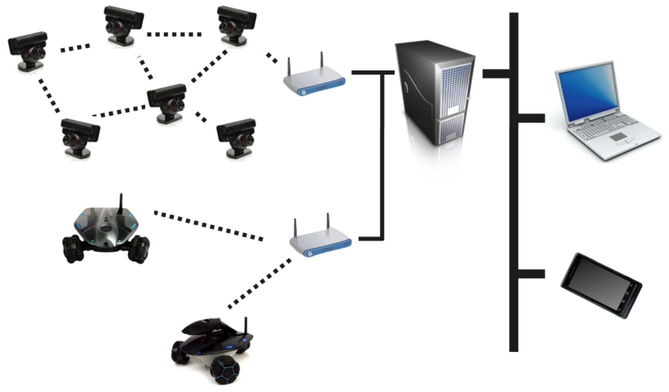
Fig. 1. Camera Network System: Camera nodes are capable of communicating with each other; mobile nodes can be used to perform active tasks such as exploration and navigation; a main server aggregates information from the network and is also used as a database; and the user interfaces with the system through the use of computers or smartphones.

At a local level, sensors that communicate with each other can randomly fail without warning. Sensors are usually resource starved or constrained on networking capabilities. Furthermore, testing the architecture can be expensive and time consuming.

Sensor networks also suffer from what we call the patchwork problem: because the needs of sensors and the servers that process their data are so different, disparate technologies are used, and nodes oftentimes communicate with each other through custom-made protocols that can be bug ridden and expensive to develop.

The rise of powerful mobile phone platforms like Android and iOS introduces an escape from some of these problems:

1) Mobile phone platforms tend to be widely available, as economies of scale drive their prices down.
2) Their hardware and software systems are much more homogeneous in nature than most sensor network architectures as a whole.
3) They have built-in support for a number of sensor needs, such as cameras and cellular radios.

Given these advantages, we expect mobile phones will become common platforms in sensor networks. There is already some active experimentation with the use of mobile phones in sensor networks, such as CitiSense, which proposes the concept of "citizen infrastructure" [4], or the employment of crowd sourcing techniques to gather data points across a large area.

Android and iOS devices are much more powerful than prior generations of embedded systems, and current projections indicate these systems will only become more capable in the future [5]. The drawback of this is that many libraries traditionally used for sensor networks are targeted at more resource constrained environments. Some libraries further require the use of special environments, such as TinyDB's need for TinyOS [6]. As a result, these libraries cannot be used in new mobile phones.

The contribution of this paper is the development of a distributed camera network which incorporates a user interface for Android devices. We outline some of the challenges we had in developing sensor network software for the Android platform, and how these challenges could be mitigated with new libraries and semantics actively being researched by efforts such as those from the *Rapidly deployable, Robust, Real-time Situational Awareness and Response* (R3SAR) research project [36], our effort to simplify sensor network design for mobile platforms.

The rest of the paper is organized as follows: *Methodology* gives an outline of our development process. *System Overview* summarizes the current state of the camera network architecture and its Android interface. *Findings* lists the issues we reached while developing the Android interface. Finally, *Future Work* details the efforts being pursued by R3SAR, and how they might mitigate the issues we had.

II. Related Work

A few papers survey the landscape of sensor networks and provide general criticisms and bodies of knowledge [7], [8], [9].

TinyDB [6] and Tables [7] explore the use of relational algebra as a communication paradigm between sensors and servers.

We focus on Android as a smartphone platform for development, although several others exist, including iOS [10] and Blackberry OS [11], among others. MeeGo is another Linux-based alternative, headed by Intel and Nokia [12].

Camera network has a number of related works, in the areas of vision graphs, simplicial representations, activity topologies and full metric models. These related works are briefly outlined below. A more detailed taxonomy is provided in [13].

Vision graphs ([14], [15], [16]) represent camera coverage as vertices and coverage overlap between cameras as edges. They do not provide geometric information on the nature of the network coverage, e.g. holes in the coverage.

Simplicial representations ([17], [18], [19], [20]) improve upon vision graphs by incorporating geometric information. This work focuses on the detection and recovery of holes in the coverage. Camera network's *CN*-complex [13] builds on simplicial representations.

Activity topologies ([21], [22], [23], [24]) identify specific artifacts shared between several camera views. Contrast this with Vision Graphs where entire camera views are compared to establish overlap.

Full-metric models ([25], [26], [27], [28], [29]) determine all geometric information about camera locations and finds overlap between views so long as no objects are obstructing the camera views. Otherwise, these objects must be located for proper characterization. Full-metric models involve a non-trivial amount of computation and tend not to be robust.

III. Methodology

In developing the application, we used the standard toolchain recommended for Android development in order to emulate a quintessential experience. Rather than experiment with newer JVM-based languages that can target the Android environment, such as Scala [30], we stuck to Java. For the IDE, we used Eclipse.

While developing the application, we took note of any common patterns in issues we met, and compared them with general findings with sensor network development at the Renaissance Computing Institute. These are documented in the *Findings* section.

IV. System Overview

In this section we go over the system that will be discussed throughout the rest of the paper and the type of applications that motivate our work.

A. The Physical Platform

A platform similar to the one described in Fig. 1 is considered. Camera nodes consist of USB cameras attached to computers running the *camera node* server application. The application provides a local GUI, and exposes an API for remote clients to connect to. Both the API and application provide access to the following:

1) Streaming updates of the camera's view.
2) Information on the predicted robot location and orientation with respect to the streaming camera images.
3) The ability to move the robot. It can be moved to another location with respect to the streaming camera images, or to another camera's field of view altogether. This involves traditional motion planning challenges, with the further issue of having to move the robot between the views of cameras that might not have any overlap.

These nodes communicate to each other wirelessly through the TCP/IP protocol.

The Rovio WowWee robot [31] is used as the target artifact. Camera network devices track the robot and send commands over a wireless connection to move it to appropriate locations. The ability to move the robot and see through its onboard webcam is also granted to any user logged into the WowWee web interface.

A main server is setup which plays the role of a gateway. It keeps track of the camera nodes that are active, the number of mobile agents, and the users connected to the system. The users are capable of connecting to the network by first interfacing with the gateway which in turn provides all the information needed (such as IP address and ports) for direct communication with the camera nodes.

This system is setup to enable distributed navigation of the robot across the area covered by the camera network. A user can provide a request to move to a specific location in a given camera view, and the distributed network handles the path planning and local control components for the robot. The user gets updates on the location of the robot in the network, as well as feedback from the robot's camera view. The motivation of this setup is to be able to provide a camera system that can be used for navigation and exploration.

B. User Interface

The Android based application starts with a view of the topology of cameras (Fi. 2, left). The topology is displayed as a graph, where the vertices are cameras and the edges indicate overlap between two camera views. Because the views of the cameras do not all overlap with one another, the graph is incomplete. The Rovio should not move into areas that none of the cameras can see, so the indication of edges are important: they suggest how the robot can move from one camera's view to another. The topology also highlights the vertices for the cameras that can currently see the Rovio.

Users can click any vertex within the topology to zoom to that camera's view. If the Rovio is within the view, the application displays its predicated location and orientation with a red line (Fig. 2, middle). Using touch gesture, a user can order the Rovio to move to another location with a given orientation, even if it is in the view of a camera different than where it currently is. The server handles the task of determining a safe path for the Rovio (where safe means the entire path is visible to one or more cameras) and moving it appropriately.

A third view provides a live stream of the Rovio's camera (Fig. 2, right). Fetching these images involves communicating directly with the Rovio base station via HTTP.

V. Findings

In this section, we discuss the problems we had developing the application.

A. High cost

Overall, the application consists of 735 lines of Java code and 87 lines of XML, not including comments and empty lines. The sloccount tool [32] estimates that the application would take 1.74 developer months to create, (using the COCOMO model [33]) with a total development cost of $19,554. These costs are extraordinarily high, especially given the fact that this application is only one component in the greater sensor network architecture. Development of sensor network software on Android using traditional technologies could prove insurmountably expensive for some.

B. Difficulty in finding domain expertise

It is currently hard to find developers with experience in Android given the relative newness of the platform. This is an unavoidable consequence of working with a new technology, and we anticipate this problem to alleviate itself in the future.

C. Incompleteness of documentation

Android has evolved rapidly since its inception. Many new capabilities have been added, and a few have been removed or deprecated. As a result, unofficial online documentation tends to be outdated and sometimes completely incorrect. Official documentation is better, although there is a large gap between the introductory material and the detailed API documentation. Additionally, methods and fields are often missing JavaDoc descriptions.

Some of this has to do with the complexity of the Android framework. Android's idiomatic development language is Java. In addition to the standard Java packages, Android provides a number of other libraries that could prove useful to a mobile application developer. This means that core Android developers have to document a strikingly large code base.

An important point to note is that the vast majority of these libraries are unnecessary for virtually all sensor network developers. This alone introduces huge opportunities for improving productivity with Android sensor network development.

Overall, the problem with respect to documentation is difficulty in determining the idiomatic way of doing things. For example, there are several ways to update the user interface based on information asynchronously received from the server. Which method to use in different contexts is difficult to discover.

D. Weak third-party libraries

We came across a serious problem while designing the topology view: to our knowledge, there are no good frameworks for drawing graphs in Android. This is especially true for hypergraphs, which is how the server sends the topological data. Some graph libraries do exist for Java, and technically Android supports importing third-party libraries. However, the process proved too troublesome. In the end, we wrote our own drawing routine, which simply arranges the vertices around a circle and changes the hypergraph connectivity to one of an undirected graph. The difficulty in setting up a graph library points to a larger problem: there are few libraries made specifically for Android, and the process of importing third party Java libraries is oftentimes more trouble than it is worth.

E. Development environment shortcomings

We used a standard Eclipse environment, as suggested in the Android documentation. However, many of the developer tools have a feeling of incompleteness to them. Furthermore, the emulator runs the complete environment in a virtual machine. This can give a feeling of sluggishness. The combination of problems with development tools are acute enough to discourage exploratory programming and rapid prototyping.

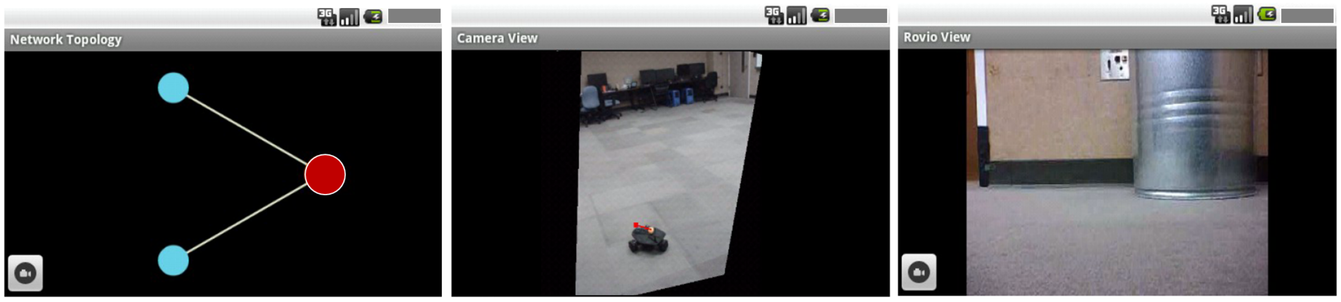

Fig. 2. Graphical Interface through Android Device: Topological view of cameras (left), Camera View with Rovio (middle) and View from Rovio (right).

Debug deployments of Android applications allow developers to watch system logs for important messages and catch exceptions along with the suspended application state when an application crashes. However, Android libraries tend not to run assertion or contractual checks against objects as they are passed around (perhaps for performance reasons). Consequently, exceptions oftentimes occur because of an object that was passed in at an entirely different time. The result is a stack trace that involves nothing but Android code, providing no indication of where the application could be supplying bad input.

F. Dealing with multi-threaded logic

Because of the way that Android is designed, multi-threaded code is an inevitable byproduct of non-trivial GUI applications. The drawbacks of this are all of the problems traditionally associated with multi-threaded code: synchronization, race conditions, deadlocks, *etc.* This holds especially true because Java has weak facilities for concurrency relative to languages like Erlang [34]. Additionally, Android is heavily event-oriented, which could have been greatly facilitated by support in Java for closures (although anonymous classes help). There is an in-progress JSR (Java Specification Request) to add closures [35].

G. The patchwork problem

As mentioned in the introduction, the patchwork problem is the common issue among sensor network architectures wherein disparate technologies are used between sensors and the systems that collate sensor data.

It is important to note that there are three separate components in the *camera network* architecture beyond the Android application:

1) The Rovio server - a web application and service for getting sensor data from the Rovio and sending commands to its actuators. This system is a black box in that it is provided by the company that manufactures Rovio (WowWee) [31] and cannot be modified.
2) The camera server - an application written in C++ that runs on systems connected to one of the topological cameras. It deals with fetching camera pictures and provides high-level commands for, e.g. moving the Rovio from a specific location in one camera's view to a specific location in another's. These commands are available through a TCP/IP based API.
3) The gateway - a server that distributes information to clients about the overall network. This includes connection settings to individual camera servers and data on the overall camera topology as a hypergraph. It also communicates with TCP/IP and is written in C++.

Getting these systems to communicate with one another involves a lot of gratuitous glue code. The Android application needs logic for fetching images via the Rovio server's HTTP API, and logic for communicating with the TCP/IP protocols of the camera server and gateway. The latter proved to be buggy, especially due to race conditions caused by camera server having to communicate with multiple clients. Overall, the patchwork architecture hampered development, and we suspect *camera network* is not unique in this issue.

VI. Future Work

Our intention going forward is to use the aforementioned issues we discovered to inform future research work. What follows is a general outline of the things we are working on in R3SAR [36], an investigation into high-level, high-productivity programming models for sensor networks.

A. A standardized communication framework

First we wish to address the patchwork problem and dealing with multi-threaded logic, which are likely the most formidable challenges even among experienced developers. To some extent, these issues will likely always exist; working with distributed systems is inherently difficult. However proper tooling and abstractions can alleviate some of the complexity. We believe a standardized protocol for allowing disparate technologies to communicate with one another in a distributed setting would be hugely useful. This is what we are most actively researching in R3SAR right now.

The protocol would have to take into account the fact that sensor devices fail frequently, and their communication links are often severed. It must also account for the need of servers to handle connections in parallel. All of this must be done efficiently, because sensor devices are often constrained on networking resources.

Prior attempts at standardizing such communication have often focused on the use of the relational paradigm combined with the implicit determination of where to perform collating computations. Examples include TinyDB [6] and Tables [7]. Because the topology of sensor networks and the nature of optimization opportunities varies so greatly, and because it is so difficult to determine much of this in an automatic fashion, we do not think these solutions are universally applicable. Instead, we suggest that the determination of where collation is done be provided explicitly by the developer. This requires more effort, but it also provides more opportunities for optimization and ensures that the system does not optimize improperly. Furthermore, [37] identifies how the relational paradigm oftentimes does not fit well with sensor data.

We instead suggest that such a framework focus strictly on standardizing communication, and that it allows for the sharing of arbitrarily shaped data. We suspect two modes of communication would be useful. The first would be a streaming fashion. Devices would establish streams with servers, and send data in real-time as it becomes available. Each chunk of data would have a Lamport timestamp associated with it [38], generated by the device. The server (or whatever is upstream) would have a library that exposes this streamed data.

Because collation is often necessary, the library would provide a method that allows an application to specify the oldest timestamped data that could possibly be of interest. This allows the application to look back in time at old data if necessary, while providing the library an opportunity to determine what data is safe to garbage collect. Multiple streams across the system would be possible, including multiple streams between two devices. The streams could be setup and destroyed during the application lifecycle. This is somewhat similar to Intel Concurrent Collections [39], except the framework would focus more strongly on distributed systems.

This communication method is ideal for upstream messaging, but not necessarily for downstream. Thus we are investigating the availability of event-based communication in this framework in addition to streams. Any device can publish events with a given name. Devices can also subscribe to events with specific names. Events are distributed as necessary to the subscribed devices. This method is nice for downstream communication because it presents an inversion of control, so that upstream devices that publish do not concern themselves with who would be interested in certain events.

In this study, such a communication framework would have been hugely useful. A number of bugs were spotted in the custom protocol, especially when multiple clients were communicating with the server. A standard framework could effectively negate the opportunity for large classes of bugs and greatly improve developer productivity. We probably would have architected the system such that the server would stream updates to the Android application, and the Android application would publish events when it requested the robot to move to another location. Logically, this makes sense: the streams would be used for high bandwidth data with frequent, regular updates, and the events would be used for relatively simple data with infrequent, unpredictable updates.

B. A declarative DSL for sensor networks

The communication framework would be useful for developers that need maximum flexibility. Furthermore, most of the issues we found relate to the use of Java and its ecosystem. Java is a very general purpose imperative programming language. A declarative language with domain specific semantics for sensor networks could provide significant productivity gains for teams without strong development expertise and further short-circuit many of the issues listed above by avoiding them altogether.

TinyDB provides one model for a sensor network DSL. It provides a SQL-like interface for querying sensors, and automatically handles the effort of data routing and collation - even in an *ad-hoc* network of sensors. Unfortunately TinyDB is not available for the Android platform, as it is dependent on TinyOS. Also, TinyDB focuses very strongly on power efficiency and performance in *ad-hoc* networks. The former is not nearly as much of a concern with high-powered Android phones, and since Android phones have cellular radios, there is no need to be able to function in *ad-hoc* environment. This might provide some opportunities to create a platform that is similar to, but more flexible than TinyDB.

As mentioned before, we do not believe relational algebra semantics are universally applicable for sensor network architectures. But in the cases in which the model is fitting, such a declarative language could remove or alleviate most of the other issues we found. High costs would likely be reduced since the developer would have to write and maintain significantly less code. It would be less difficult in finding domain expertise because the language would use mostly SQL abstractions, hiding the Android-specific logic. Incompleteness of documentation on the part of Android would not be a concern as developers would be unlikely to have to delve into that information.

The language could additionally build on top of the communication framework to provide a cohesive environment for developers to create sensor network architectures. For simple architectures, developers could use the declarative language. If the language proved too inflexible, they could drop down to the communication framework. Applications could even be composed of some combination of the two abstractions.

VII. Conclusion

We believe smartphones or platforms with similar capabilities will play an increasingly prevalent role in sensor network architectures given their commodity pricing and rich sensing hardware. Thus, we set out to design an application for Android that acts as part of a sensor network to get a feel for what the current challenges are. Using our knowledge from building the application, we provide a number of pain points for Android development, and suggest ideas that could alleviate some issues with sensor network development on smartphone platforms.

Our work is far from complete. Now that we have some insight on sensor network development, we intend to continue work on the suggested ideas. We further intend to upgrade *camera network* to use this model, and report on the results.

Acknowledgments

Some of this research was partly funded by the National Science Foundation grant CPS 0932011 and grant 0937060 to the Computing Research Association for the CIFellows Project.

References

[1] B. Rinner and W. Wolf, "An introduction to distributed smart cameras," in *Proceedings of the IEEE*, vol. 96(10), 2008, pp. 1565–1575.

[2] P. C. et al., "CITRIC: A low-bandwidth wireless camera network platform," in *Third ACM/IEEE International Conference on Distributed Smart Cameras*, 2008.

[3] A. Kandhalu, A. Rowe, and R. Rajkumar, "DSPcam: A camera sensor system for surveillance networks," in *Proc. of the ACM/IEEE Intl. COnf on Distributed Smart Cameras (ICDSC)*, 2009.

[4] "Citisense," http://seelab.ucsd.edu/health/overview.shtml.

[5] J. Hill, M. Horton, R. Kling, and L. Krishnamurthy, "The platforms enabling wireless sensor networks," *Commun. ACM*, vol. 47, pp. 41–46, June 2004. [Online]. Available: http://doi.acm.org/10.1145/990680.990705

[6] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: an acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.*, vol. 30, pp. 122–173, March 2005. [Online]. Available: http://doi.acm.org/10.1145/1061318.1061322

[7] J. Horey, E. Nelson, and A. B. Maccabe, "Tables: A table-based language environment for sensor networks."

[8] R. Sugihara and R. K. Gupta, "Programming models for sensor networks: A survey," *ACM Trans. Sen. Netw.*, vol. 4, pp. 8:1–8:29, April 2008. [Online]. Available: http://doi.acm.org/10.1145/1340771.1340774

[9] L. Mottola and G. P. Picco, "Programming wireless sensor networks: Fundamental concepts and state-of-the-art."

[10] "ios 4," http://www.apple.com/iphone/ios4/.

[11] "Blackberry 6," http://us.blackberry.com/apps-software/blackberry6/.

[12] "Meego," http://meego.com/.

[13] E. Lobaton, R. Vasudevan, R. Bajcsy, and S. Sastry, "A distributed topological camera network representation for tracking applications," *Trans. Img. Proc.*, vol. 19, pp. 2516–2529, October 2010. [Online]. Available: http://dx.doi.org/10.1109/TIP.2010.2052273

[14] Z. Cheng, D. Devarajan, and R. J. Radke, "Determining vision graphs for distributed camera networks using feature digests," *EURASIP J. Appl. Signal Process.*, vol. 2007, pp. 220–220, January 2007. [Online]. Available: http://dx.doi.org/10.1155/2007/57034

[15] D. Marinakis and G. Dudek, "Topology inference for a vision-based sensor network," in *Proceedings of the 2nd Canadian conference on Computer and Robot Vision*, ser. CRV '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 121–128. [Online]. Available: http://dx.doi.org/10.1109/CRV.2005.81

[16] D. Marinakis, P. Giguère, and G. Dudek, "Learning network topology from simple sensor data," in *Proceedings of the 20th conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence*, ser. CAI '07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 417–428. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-72665-4_36

[17] V. De Silva and R. Ghrist, "Coordinate-free coverage in sensor networks with controlled boundaries via homology," *Int. J. Rob. Res.*, vol. 25, pp. 1205–1222, December 2006. [Online]. Available: http://portal.acm.org/citation.cfm?id=1274636.1274646

[18] A. Muhammad and A. Jadbabaie, "Decentralized computation of homology groups in networks by gossip," in *American Control Conference, 2007. ACC '07*, 2007, pp. 3438 –3443.

[19] E. J. Lobaton, P. Ahammad, and S. Sastry, "Algebraic approach to recovering topological information in distributed camera networks," in *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, ser. IPSN '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 193–204. [Online]. Available: http://portal.acm.org/citation.cfm?id=1602165.1602184

[20] E. Lobaton, R. Vasudevan, S. Sastry, and R. Bajcsy, "Robust construction of the camera network complex for topology recovery," in *Distributed Smart Cameras, 2009. ICDSC 2009. Third ACM/IEEE International Conference on*, 302009-sept.2 2009, pp. 1 –8.

[21] D. Makris, T. Ellis, and J. Black, "Bridging the gaps between cameras," in *Proceedings of the 2004 IEEE computer society conference on Computer vision and pattern recognition*, ser. CVPR'04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 205–210. [Online]. Available: http://portal.acm.org/citation.cfm?id=1896300.1896329

[22] A. van den Hengel, A. Dick, and R. Hill, "Activity topology estimation for large networks of cameras," in *Proceedings of the IEEE International Conference on Video and Signal Based Surveillance*, ser. AVSS '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 44–. [Online]. Available: http://dx.doi.org/10.1109/AVSS.2006.17

[23] H. Detmold, A. van den Hengel, A. Dick, A. Cichowski, R. Hill, E. Kocadag, Y. Yarom, K. Falkner, and D. Munro, "Estimating camera overlap in large and growing networks," in *Distributed Smart Cameras, 2008. ICDSC 2008. Second ACM/IEEE International Conference on*, 2008, pp. 1 –10.

[24] R. Hill, A. van den Hengel, A. Dick, A. Cichowski, and H. Detmold, "Empirical evaluation of the exclusion approach to estimating camera overlap," in *Distributed Smart Cameras, 2008. ICDSC 2008. Second ACM/IEEE International Conference on*, 2008, pp. 1 –9.

[25] S. C and K. Tieu, "Automated multi-camera planar tracking correspondence modeling," in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, vol. 1, 2003, pp. I–259 – I–266 vol.1.

[26] L. Lo Presti and M. La Cascia, "Real-time estimation of geometrical transformation between views in distributed smart-cameras systems," in *Distributed Smart Cameras, 2008. ICDSC 2008. Second ACM/IEEE International Conference on*, 2008, pp. 1 –8.

[27] M. Meingast, M. Kushwaha, S. Oh, X. Koutsoukos, A. Ledeczi, and S. Sastry, "Fusion-based localization for a heterogeneous camera network," in *Distributed Smart Cameras, 2008. ICDSC 2008. Second ACM/IEEE International Conference on*, 2008, pp. 1 –8.

[28] A. Rahimi, B. Dunagan, and T. Darrell, "Simultaneous calibration and tracking with a network of non-overlapping sensors," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 1, 2004, pp. I–187 – I–194 Vol.1.

[29] S. Funiak, C. Guestrin, M. Paskin, and R. Sukthankar, "Distributed localization of networked cameras," in *Information Processing in Sensor Networks, 2006. IPSN 2006. The Fifth International Conference on*, 0-0 2006, pp. 34 –42.

[30] "The scala programming language," http://www.scala-lang.org/.

[31] "Wowwee," http://www.wowwee.com/.

[32] D. A. Wheeler, "Sloccount," http://www.dwheeler.com/sloccount/.

[33] C. F. Kemerer, "An empirical validation of software cost estimation models," *Commun. ACM*, vol. 30, pp. 416–429, May 1987. [Online]. Available: http://doi.acm.org/10.1145/22899.22906

[34] "Erlang programming language, official website," http://www.erlang.org/.

[35] B. Goetz, "Jsr 335: Lambda expressions for the javatm programming language," http://jcp.org/en/jsr/detail?id=335.

[36] "R3sar," http://www.renci.org/focus-areas/computing-technology/r3sar.

[37] K. Henricksen and R. Robinson, "A survey of middleware for sensor networks: state-of-the-art and future directions," in *Proceedings of the international workshop on Middleware for sensor networks*, ser. MidSens '06. New York, NY, USA: ACM, 2006, pp. 60–65. [Online]. Available: http://doi.acm.org/10.1145/1176866.1176877

[38] L. Lamport, "Ti clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, pp. 558–565, July 1978. [Online]. Available: http://doi.acm.org/10.1145/359545.359563

[39] "Intel concurrent collections framework," http://software.intel.com/en-us/articles/intel-concurrent-collections-for-cc/.