Design and Implementation of Ubiquitous Smart Cameras

Chang Hong Lin, Wayne Wolf
Princeton University
Princeton, NJ 08544, USA
{chlin,wolf}@ee.princeton.edu

Andrew Dixon, Xenofon Koutsoukos, Janos Sztipanovits
Vanderbilt University
Nashville, TN 37235, USA
{dixonad, Xenofon.Koutsoukos, janos.sztipanovits}@vanderbilt.edu

Abstract

Design aspects and software modelling for ubiquitous real-time camera system are described in this paper. We propose system architecture using a network of inexpensive cameras and perform video processing in-network. In general, ubiquitous systems have to perform spatial and temporal calibration in advance to determine timing and coordination relationship between sensor nodes, and other application specific design considerations, such as system architecture, distributed software, control authority and communication channel. A methodology for transforming well-designed single-node algorithm to distributed system is also proposed. Applications for ubiquitous cameras can be modelled as the composition of system finite state machine, functional services and middleware. A service oriented software architecture is proposed to dynamically reconfigure services when system state changes. We have developed a distributed gesture recognition system with true in-network processing to analyze video in real time. By exchanging data and control messages between neighboring sensors, each node can maintain broader view of the environment with integrated video processing results. Our prototype system is built on Windows machines, and uses webcams as sensors and local network as communication channel.

1. Introduction

In this paper, we describe methods and modelling for ubiquitous real-time camera system. We have developed a distributed multi-camera system for real-time gesture recognition using in-network microprocessors embedded within the sensor nodes. Sending video streams from sensor nodes to centralized servers is not sufficient for real-time application with multiple sensors. Processing video streams in distributed fashion close to or inside the sensor nodes holds the key to realize real-time ubiquitous sensor systems. By exchanging data and control messages between neighboring sensors, each node can maintain a broader view of the environment with integrated video processing results. Service oriented software architecture is also proposed to model sensor system behaviors, and to dynamically reconfigure system functions according to different system states.

Moore's Law has correctly predicted electronics have become cheaper and more powerful over time. Both sensor units and microprocessors can be integrated into a smart camera system to both capture and process video streams in a single package. Smart cameras can perform various real-time video processing, including face, gesture and gait recognition, and object tracking. The use of multiple cameras makes it possible to solve many video applications, such as wild area surveillance, 3-D image reconstruction, and video sensor networks. In order to perform real-time processing, ubiquitous cameras require in-network processing power to handle computational tasks close to where the video is captured. Instead of process only captured video streams, distributed cameras can take advantage of communication with their neighbors to integrate the knowledge of overall system without using centralized servers.

Embedded real-time system should produce not only correct results, but also at the right time. Other than functional correctness, timing, reliability, robustness and power consumption are all important aspects in embedded system design. As sensor systems become much more complicated, modelling and verifying system behaviors become quite a challenge. We propose a service oriented software architecture for ubiquitous sensor systems. A system can be modelled as the combination of system finite state machine, functional services and middleware. The modelling not only eases the understanding and verification, but also provides dynamic reconfiguration during runtime. The next section summarizes related work in ubiquitous camera system. Section 3 introduces design considerations for distributed sensor systems, and section 4 and 5 further describe application independent and specified processing respectively. Service oriented architecture for distributed system is proposed in section 6. An ubiquitous system example is stated in section 7, and section 8 concludes the paper.

2. Related Work

Advances in technology make sensor networks possible, and many researches have been studied in recent years. Although ubiquitous camera system can be considered as a kind of sensor network, distributed systems of cameras post some new challenges. General sensor network contains huge amount of low-cost sensors with limited energy and computing power; while camera system often has fewer nodes with much more energy and resource requirements.

Several architectures and algorithms for real-time camera system have been proposed, and our group were the first to adopt distributed computing [1]. Instead of peer-to-peer computation in our system, most previous systems used centralized server(s) for video processing. Pentland surveyed several real-time video analysis efforts in detail [2]. Besides algorithm development, hardware design is also an important issue for real-time system. Bove's group at the MIT Media Lab proposed a data-flow model for real-time parallel media processing and built tiles of smart sensors [3, 4].

Davis et al. developed a multi-perspective video system for human action analysis and object detection and tracking [5, 6]. Ozer et al. synthesized a 3D model of humans from two cameras approximately perpendicular with each other [7]. The Stanford multi-camera array group uses a dense array of CMOS image sensors to capture multi-thousand frame-per-second videos [8]. Rinner's group in Graz University of Technology developed an embedded camera system for traffic monitoring, and proposed methods for dynamic task allocation among group of cameras [9]. Most of the systems described can achieve real-time performance with centralized processing. However, sending raw video streams to centralized servers is not efficient and practical, especially when the transmission cost between camera nodes is measurable. Using shared memory or buses is unrealistic in real-life applications. Distributed computing in the microprocessors inside or near the sensors and exchange limited processed data would be a better choice.

As embedded camera systems become much more complicated, to model and verify system functions become a tough challenge. Ubiquitous distribution makes the design and verification of communication channels between sensor nodes even more difficult. Several groups have developed tools and environment for software architecture to model distributed systems. The Model-Integrated Computing (MIC) is based on domain-specific models to analyze and test embedded softwares [10]. The *CADENA* framework is an integrated environment for analyzing CORBA Component Model (CCM) based systems [11]. And Holzmann developed *SPIN* model checker to verify distributed software through message passing [12].

In this paper, we study properties for ubiquitous camera system, including both application independent calibration and application specific processing. A software architecture to model distributed sensor system is also presented, and can be further verified using techniques just stated.

3. Ubiquitous Real-Time Camera System

Cameras and microprocessors are now cheap enough so that many smart camera nodes can be integrated into one single system. Though centralized servers simplify many design decisions, it is not realistic for ubiquitous real-time camera systems. Distributed cameras inherently need distributed computing powers to execute various tasks close to the sensors. Captured video streams are processed in network, and sensor nodes communicate with each other to construct an overall view of the entire system.

3.1. Server-Based Multi-Camera System

To analyze video from multiple cameras, we must face the problem of how to integrate the data from multiple cameras. Traditionally, multi-camera systems have relied on centralized server(s): the video streams captured from sensors are sent to one central server (or a cluster of servers) for processing. Server-based processing of image/video data simplifies many design problems. Timing synchronization between distributed sensor nodes is not a problem for centralized system. The data sharing between different nodes can be performed in the hardware and software in the centralized server(s). Software using centralized decision is relatively simpler than software using distributed control for the same video processing task. For applications with physically close cameras and high-performance communication channels, centralized server(s) may be a good design determinant. Stanford's camera array [8] and Ozer's system [7] are two such examples. The camera array uses on-board communication and Ozer's system uses shared-memory. However, sending image/video data to the server(s) has severe penalties for real-time systems with sparsely distributed cameras. Centralized servers require high-performance networks to connect sensors to server(s). These networks consume a significant amount of energy, which may be too high to be supported by battery or other energy sources with limited power. And the transmitted video may also be tampered with or may be disrupted.

3.2. Distributed Multi-Camera System

Ubiquitous camera systems inherently require distributed processing power. As illustrated in Figure 1.(a), the cameras can be set up at arbitrary places, and people or other objects of interest may freely move around the field of view of the cameras. Camera nodes have to exchange information of the captured frames in order to distributed process the video streams in an overall view. In order to minimize the overall communication cost, we would like to perform at least some of the video processing in the processors at or near the camera which captures the images.

Peer-to-peer control algorithms are needed to assign video processing tasks to distributed processors. In order to fulfill real-time requirement, the sensing and computing devices have to be properly selected to fulfill system requirement, and the camera geometry and communication channels have to be determined based on application. The spatial and temporal relationship between sensor nodes has to be calculated in order to compare frames taken around the same period with their neighbors. Single camera algorithm has to be expanded to multiple-camera version, which

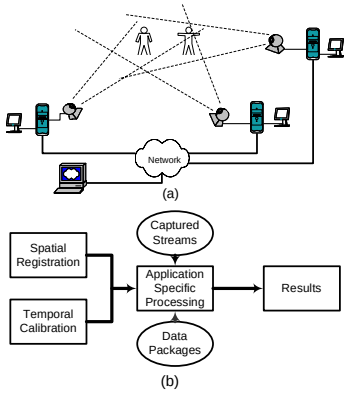

Figure 1. (a) Geometry and (b) functional diagram for ubiquitous camera system.

parallel runs on ubiquitous processors. The control authorities for objects of interest need to be determined and passed among processing elements to assign which unit is responsible for what parts of the overall computation. Communication protocols to transmit control and data information between sensor nodes, distributed scheduling within the communication channels, and system energy minimization are also important issues in ubiquitous camera systems.

Design considerations for ubiquitous camera can be further divided into two categories: application independent and application specific processing. Spatial and temporal calibrations are necessary for initialization for all applications; while other aspects differ from application to application. As shown in Figure 1.(b), a general ubiquitous camera system would collect spatial coordination and synchronize clocks in advance, and update this information on-the-fly during run-time. Then, the distributed processors would take their own captured video and data and control messages from neighboring nodes to generate desired outcome.

4. Application Independent Calibrations

In this section, we describe application independent processing for ubiquitous camera system – spatial and temporal calibrations. Before distributed video processing tasks can operate successfully, sensor nodes have to know the spacial relationship and synchronize their clocks with neighbors.

4.1. Field of View Registration

In multi-camera systems, rather than treating each individual node independently, it is important to establish communication between sensors in order to hand-off processing from one node to another. Knowing overlapping field of views (FOVs) can provide efficient communication scheme between sensor nodes. Cameras can communicate with each other directly without intermediate central control units. In addition, based on the knowledge of when and what to communicate, ubiquitous camera system works better and more efficiently as less communication required. Once the current camera unit knows an object of interest has entered the view of its neighbor, it knows where that object will be once it leaves its own FOV. One possible way to represent the coordination relationship is to find the FOV lines of neighboring cameras inside the views of each camera automatically, as developed by Khah et al. [13]. Once the FOV lines are determined, multi-camera algorithms can then handle hand-off and data transmission between sensor nodes much more easily. Velipasalar and Wolf developed an improved method for determining the FOV lines [14].

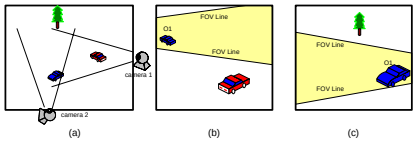

Figure 2. (a) Overall view, (b) view from camera 1 and (c) view from camera 2.

Here we use multiple object tracking as an example. Moving objects are tracked by multiple cameras, and our objective is to provide correspondences between tracks in different camera views, which means to give the same label to the same moving object seen by different nodes. As shown in Figure 2, suppose the object seen in the left upper part view of camera 1 is labeled as *O1*. Once this object passes the FOV line of camera 2, as marked in Figure 2.(b), it becomes visible and starts being tracked by camera 2. As soon as the object passes the FOV line, camera 1 communicates with camera 2 and passes the label of the object. This way, the same object is given the same label throughout the cameras, which means object *O1* is tracked successfully.

4.2. Timing Synchronization

The necessity of timing synchronization was first motivated by Lamport in [15], and many researchers had put efforts on this topic. When dealing with cameras with overlapped field of views, it is meaningless to compare frames from different time point. The comparison is meaningful only when the frames were took around the same time. As stated by Elson and Römer [16], no single synchronization algorithm can work for all the applications. When designing the synchronization for ubiquitous cameras, system requirements, application software, camera architecture, network topology and other characteristic have to be taken into account. One or hybrid synchronization method can then be chosen to fit the application. For example, we can apply Lamport's algorithm on Ethernet based networks [17], and IEEE 802.11 standard's timing synchronization function is also a possible choice on wireless networks [18].

Figure 3.(a) shows the importance of synchronization in ubiquitous systems. It reveals the timing sequence of a system with three cameras within two frames. Camera 1 and 2 are assumed synchronized, and camera 3 is out of synchronization. For most of the applications, the data flow can be divided into three steps: *local processing*, *data transmitting* and *joint processing*. The local processing part takes the captured frame, performs various tasks and prepares data

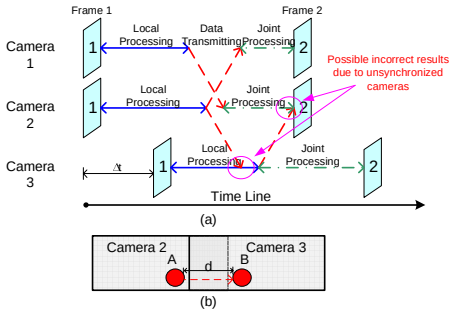

Figure 3. (a) Basic timing sequence for ubiquitous camera system; (b) possible hazard.

for transmission. Messages then being passed to the neighboring nodes. Camera nodes consider both captured image and received data to produce results with overall view in the joint processing phase. Problem may occur when the camera nodes are not synchronized. Camera 2 didn't receive camera 3's packets before frame 2 been shot. Depends on the program, camera 2 might output a wrong result for frame 1 without camera 3's information, or might just drop a frame. The result from camera 3 might not be correct due to the movement of the objects inside the views. Suppose camera 2 and 3's field of views are as shown in Figure 3.(b), and an object moves from position *A* to *B* when the cameras took the shots. Most distributed algorithms would fail to recognize both objects as the same one at first sight.

5. Application Specific Processing

We introduce application specific processing for ubiquitous system. Though there is no universal rule to make design decisions, sensor geometry, image processing software, communication channel, cost and power budget are important aspects. Total system cost should be within budget, including equipments, development, installation and maintenance. Sensors should provide required resolution and sensitivity. Processing Units have to handle computing, control and communication in real-time. FOV of the cameras should cover applications' specification. Underlying network should be able to provide sufficient bandwidth for communication. And power distributed system or batteries should be able to support the camera geometry set-up.

5.1. Migration Methodology

To change a single camera application into ubiquitous camera system, we first determine which parts can be directly inherited, and which parts cannot. The camera nodes have to exchange information with neighboring nodes in order to get an overall view of the captured video streams. The programs are divided into several stages based on the software architecture, and the results of each stage are candidates to be transmitted to neighboring nodes. Each node will exchange data packages and take the captured and processed images, along with the received data to perform following stages. The distributed program run in each node can perform most of the single camera operations with additional multi-camera controls. However, we have to determine the information to be transferred in the earlier stages, and integrate the received data in the later stages.

After the stage to exchange information is determined, we then decide which data and control messages shall be passed to the neighbors and which is processed only in the current node. The data passed depends on the application, performance requirement, communication cost, and other issues. The decision can only be made after taking into account all the system trade-offs. This migration methodology has to fulfill two characteristics: correct and optimized implementation. The mechanism used has to ensure the operation is performed correctly; and to find the optimum scheduling to perform the distributed programs.

5.2. Control Authority Determination

Besides the image data exchanged between the cameras, some applications might exchange control signals for many purposes. Some applications might need the nodes to exchange ownership of objects inside the scene to determine which camera node should perform most of the high-level processing on the certain object. Some camera nodes might want to notify their neighbors that certain objects are moving toward their field of views, or want to send out the characteristics of certain objects to its neighbors to let the neighbors identify them. The control signals could also be used to perform periodical timing resynchronization or spatial calibration. The system may use a separated channel for control signals, or can embed control signals within data packages. Since distributed programs execute concurrently, concurrency and resource sharing are important issues in ubiquitous camera system. A service oriented modelling architecture is proposed to verify control authority.

5.3. Communication and Power Trade-offs

Though we do not expect ubiquitous camera networks to be powered by batteries in the foreseeable future, low-power is still an important issue for realistic distributed system. Power consumption is directly related to installation costs; the more power consumed, the higher-cost power distribution networks and cooling systems would be needed. The power consumption could be divided into two major parts: computation and transmission power. Computation power is mostly determined by the image processing algorithms used; on the other hand, transmission power depends on the packet size, retransmission rate and transmission distance. The larger the packet size and the longer it takes to transfer, the more transmission energy is consumed. This becomes even worse when data transmission dominates the power consumption (eg. wireless network).

Depends on the application and camera geometry, ubiquitous camera systems may use different type of networks

as communication channel to exchange messages. For systems with sensors densely distributed in a small area, wired channels such as local area network (LAN) might be a good choice. On the other hand, for systems with sensors sparsely distributed in a wide area, wireless communication methods such as IEEE 802.11 or satellite are better choices. The data bandwidth between cameras is determined by communication channel, sensor distribution and scheduling algorithm. The channel type determines the maximum total bandwidth; and the scheduling algorithm and the number of nodes would determine the effective bandwidth per link for each camera pairs. Transmission protocols is another important issue in communication. Application layer protocols are required for data and control message exchange, and lower layer protocols are needed when customized middleware is used. Error correction code or encryption can also be added to the designed protocol to provide further fault-tolerance and security in communication channel.

In order to prevent mass packet loss due to network congestion, the traffic level should be kept less than saturation [19]. Suppose the network saturates at network utilization Sat, the network bandwidth should be at least Tr_{peak}/Sat, where Tr_{peak} represents the total peak traffic of the network. Peak traffic is used due to larger data packets often contains more important information, such as *I frames* in *MPEG* stream. The total traffic is determined by the sensor topology, network configuration and peak traffic in each link between camera pairs. For a giving network structure, the bandwidth needed is proportional to the peak packet size. We can then have *40X* saving in bandwidth if abstract models are used instead of compressed video streams.

The total traffic level highly depends on the camera topology and network structure. Suppose we have four camera nodes with geometry shown in Figure 4, and the distance between neighboring cameras is R. Each camera node has to communicate with other cameras with overlapped field of views with peak traffic T. When the cameras are connected in the same wired network, the total traffic is simply the summation of the peak traffics on each link. When wireless network is used, we need to consider both transmission range and scheduling for different topology. Only one node can transmit data at the same time in a 2-hop area, while nodes more than 2 hops away can transmit simultaneously. Careful scheduling methods can reduce the bandwidth requirement by transmitting uncorrelated links simultaneously without congestion. Though camera nodes may transmit packages to neighbors with different distance, the transmission power is proportional to the square of distance to the receiver. Table 1 illustrates the total traffic and transmission power for different network topology in wired and wireless network using direct transfer or multi-hop. Wireless network may require less bandwidth than LAN due to channel sharing. Compare to multi-hop, direct transfer

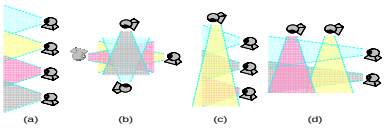

Figure 4. Camera topology examples.

Network	Topology	(a)	(b)	(c)	(d)
LAN	Traffic	$6T$	$12T$	$10T$	$12T$
Direct Transfer	Traffic	$5T$	$12T$	$9T$	$11T$
	Power	$6TR^2$	$16TR^2$	$28.5TR^2$	$31.3TR^2$
Multi-Hop	Traffic	$5T$	$16T$	$13T$	$17T$
	Power	$6TR^2$	$16TR^2$	$15TR^2$	$20TR^2$

Table 1. Total traffic and transmission power for different network geometry.

method needs less system bandwidth with the requirement of more power and tunable transmitters. Camera placement is an important aspect in distributed video processing. It not only determines the algorithm difficulty, but also the bandwidth and power requirement for communication channels.

6. Distributed Software Architecture

In this section, we propose a software architecture for ubiquitous sensor systems. The environment we use is based on Model-Integrated Computing (MIC) [20], which use domain-specific modelling languages to provide a flexible framework for embedded software development.

6.1. Service Oriented Architecture

Traditional embedded systems perform fixed tasks as specified by the software. Even though the system may have multiple threads doing different job, the services provided by the system are fixed once the system boot up. However, as embedded systems become much more complicated, running all possible services becomes impossible. Naively executing all the services will not only increase the operating system scheduling difficulty, but also consume extra energy. Take our gesture recognition system as an example: our prototype system [1] is naive, and would perform the recognition process all the time. However, if we have in priori the knowledge of no objects inside the field of view, we can turn off the whole recognition system safely. We would like the embedded systems to provide complicated services, but perform the services only at the *right* time.

We propose *service oriented architecture* for ubiquitous sensor systems. Each sensor node may provide several services. A service is a software architecture that perform certain tasks, and is enabled according to the system states. For example, gesture recognition is a service, and it is enabled when there're moving objects inside the scene. The software architecture is either enabled or disabled due to the current system state. The required services are specified for each system state, and are dynamically bound during runtime. The service here not only refer to different type of functions, but also to different algorithms for the same function. At a system state, the sensor node may require a series

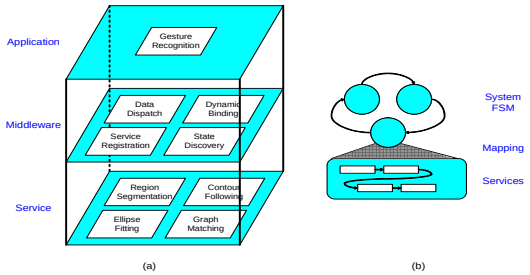

Figure 5. (a) Service oriented architecture example; (b) system modelling phases.

of functional services, and depends on the environment settings, the node may choose different service with the same functionality. For example, a sensor node may require a background substraction service followed by a recognition service, and due to the lighting condition, the sensor may use different background substraction algorithm.

In order to bind required services for a system state, interface and middleware are needed to handle dynamic service binding and message passing. Each service should provide input/output channel information of the functional block, as well as service attributes to the middleware. The middleware can then register the services and dispatch data messages between them. Services with similar functionalities should have the same input and output channels, with possible different service attributes. Using a service requires knowing only its name and interfaces. When system state changes, the middleware would discover the change and dynamically reconfigure the system services.

We can further divide the service oriented architecture into three layers, as illustrated in Figure 5.(a). The first layer is application, which defines the objective of the combined services under it, and can be a service itself for larger applications. Here we use gesture recognition as the objective application example. The second layer is middleware, which maintains the correct services for different system states. The middleware discovers system state change, dynamically binds and registers the required services, and distributes messages between services during runtime. The third layer contains services provided by the application, region segmentation, contour following, ellipse fitting and graph matching are all possible services. Depends on the possible system states, we may have even more services.

6.2. System Modelling

The system modelling consists of three major phases, as shown in Figure 5.(b): in application layer, to construct the system operation finite state machine; in service layer, to define the possible services, either atomic or composite; and in middleware layer, to determine the mapping between the current system state and provided services. There is no unique way to construct the system finite state machine, so the designers may choose any FSM fulfills the application

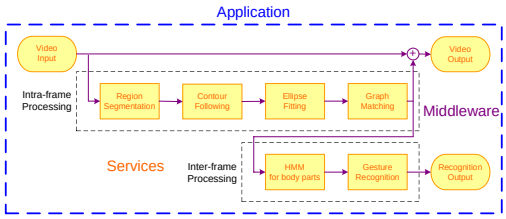

Figure 6. Single camera software architecture of gesture recognition system.

requirement. One possible solution for distributed sensors is target-centric modelling. For each target object enters the scene, system would create a target FSM, change the FSM state as target moves around or be noticed by services, and destroy when target leaves the environment. Each basic functional block can be considered as an atomic service, and many atomic services can be connected to perform more complex behaviors, which is called composite service. Hierarchically, a well-defined application FSM can also serve as an atomic service of higher level applications.

7. Example: Gesture Recognition System

We describe our distributed gesture recognition system as an implementation example of ubiquitous system. This system was first proposed in 2004 [1] and had been revised since.

7.1. Single Camera Algorithm

Video processing algorithms can be broken into several service stages. The software architecture of a single node gesture recognition is illustrated in Figure 6, which consists of intra and inter-frame processing. The intra-frame part performs human body detection and extracts abstract graph representation parameters. It starts with region segmentation to identify foreground objects and skin regions. The system then extracts contours on the boundaries of detected regions, and find abstract ellipse parameters to model various regions. These ellipses are then matched to different human body parts. The inter-frame processes use hidden Markov models (HMMs) to determine movements of the body parts, and use a distance classifier to detect specific gestures. Gesture recognition system is the application in service oriented architecture, and each functional block in Figure 6 serves as a service in this application. The control and data dispatch pathes between the blocks are the middleware, which is *Microsoft DirectX* in our prototype system.

7.2. System Partitioning

In our gesture recognition system, the interfaces of intra-frame services are candidates for data transmission. Since targets may stay around the overlapping region of camera pairs, some body parts might locate outside camera's field of view. Both region segmentation and contour following handle pixel based data, which is too huge to send. The data size for contour points is usually several Kbytes, while

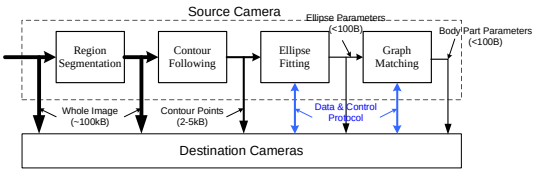

Figure 7. Data transmission candidates for distributed gesture recognition system.

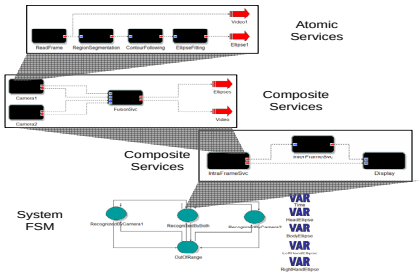

Figure 8. Software architecture modelling for ubiquitous gesture recognition.

ellipse parameters and matched body part with HMMs coefficients often cost less than 100 bytes. When the network bandwidth is large, the contour points would be a good candidate for transmission; while the ellipse parameters and matched body parts would be the choices when the traffic is busy. The data size transmitted from different stages is illustrated in Figure 7. In order to recognize the gestures of a person at the boundaries, body part information has to be passed between overlapping sensor pairs. Since the size of human bodies is bounded, only the parts lie inside or near the overlapping region need to be transferred.

7.3. Software Architecture Modelling

Figure 8 shows the data flow graphs using service oriented architecture. We have two cameras with overlapped field of view, and target person can move around freely inside the scene. The system can be divided into four states: camera 1 only, camera 2 only, overlapped region, and out of range. In different state, system would provide different set of services. When target only locates in one of the cameras, only the camera with target need to perform gesture recognition, and the other can stay in low-power mode and wake up again when system changes to target in overlapped region state. The services provided in each system state are composite services, which in turn are aggregate services. Our basic functional blocks, such as region segmentation, ellipse fitting,... etc., along with control logic blocks serve as the bottom part of the hierarchy, the atomic services.

7.4. Recognition Control

Camera nodes exchange control messages to determine which node should perform gesture recognition for a target. We assume the camera geometry remain stable during the experiment, and the control messages contain timestamps

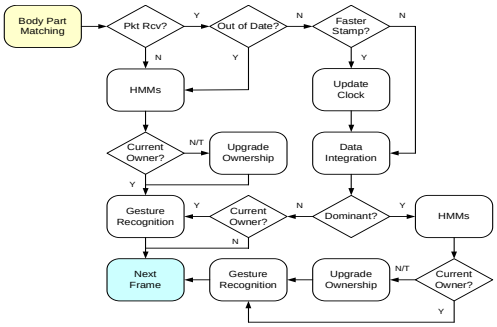

Figure 9. Control flow for ubiquitous gesture recognition.

for synchronization and tokens indicate the ownership of the people in the overlapping area. The token would used to determine which node should perform higher level recognition algorithms. Suppose camera nodes exchange data packages after matching, the system function can be modelled as in Figure 9. After a sensor node finishes matching service, it would wait and check if there is data packet from its neighbors. If no data packet arrived, or the data packet received has an out of date timestamp, the sensor node then perform inter-frame on its captured body parts. If the target is not owned by the node, the node would claim temporary ownership, in case of delayed or lost packets from neighbors. When a node receives messages from its neighbors, it first checks the timestamp of the packet, and update its clock if faster timestamp is received. The sensor nodes would then find matching body parts for certain targets between the captured image and received data, and then the one contains much more pixels would be considered as the dominant body part. The major part of the object of interest and the token received are used to determine the ownership of the object. For example, head can be used as the major part of human body. The new owner of the target would go on perform inter-frame algorithms, while the other camera would discard the whole body parts of that object.

7.5. Gesture Recognition Example

Currently, our system runs on a set of *Windows* machines, and uses webcams to capture video streams. The computers are connected in a local area network, and use user data protocol (UDP) to transmit packets between cameras. Ultimately, we expect our cameras sparsely distributed in an area, and use wireless network to communicate with each other. Figure 10 displays snapshots from our gesture recognition system. The two video streams are taken from cameras with parallel and slightly overlapped field of view. If only single camera is used to recognize the movements of both streams, the left clip would be detected as *hand right* while the right clip as *hand left*. However, multi-camera system would take information from both streams, and determine the entire movement as *open hand*. Our current version of distributed recognition can run at 15.23 frames

Figure 10. Snapshots from distributed gesture recognition system using (a) single-camera and (b) multiple-camera algorithms.

per second on a *Pentium III* 1GHz PC with 128M RAM.

8. Conclusions and Future Work

Advances in VLSI technology makes it possible to perform real-time video processing in embedded systems. Centralized system simplifies design decisions, but is not realistic in real system; instead, peer-to-peer control and computation are necessary. We divide the design aspects into two categories: general and application specific problems. In general, ubiquitous systems have to perform calibration before hand to let camera nodes synchronize and know the coordination relationship. Then, the designers have to look into application specific issues, such as system architecture, video-processing software, control authority, communication channel, and reliability. A service oriented software architecture is introduced for ubiquitous sensors. A system can be modelled as the combination of system finite state machine, services and middleware. Depends on the system state, sensors may dynamically bind different set of services. We propose a distributed gesture recognition system as a design example. Our system use *Windows* machines and webcams to capture video streams. Sensor nodes exchange data and control messages with neighbors to maintain broader view of the environment, and distributed perform gesture recognition. For future work, we would like to work on distributed scheduling, power management and automatic calibration algorithms, as well as apply other applications to our ubiquitous framework, such as tracking, face and gait recognition. Ultimately, we would like to have more cameras in a wider area using wireless ad-hoc network to perform various in-network processing applications.

References

[1] C. Lin, T. Lv, B. Ozer, and W. Wolf, "A peer-to-peer architecture for distributed real-time gesture recognition," in *Int'l Conf. Multimedia and Exhibition.* IEEE, June 2004.

[2] A. Pentland, "Looking at people: Sensing for ubiquitous and wearable computing," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, no. 1, Jan. 2000.

[3] J. Watlington and V. B. Jr., "A system for parallel media processing," *Parallel Computing*, vol. 23, no. 12, Dec. 1997.

[4] V. B. Jr. and J. Mallet, "Collaborative knowledge building by smart sensors," *BT Technology Journal*, vol. 22, no. 4, Oct. 2004.

[5] L. S. Davis, E. Borovikov, R. Cutler, and T. Horprasert, "Multi-perspective analysis of human action," in *Int'l Workshop on Cooperative Distributed Vision.*, 1999.

[6] A. Mittal and L. Davis, "Unified multi-camera detection and tracking using region-matching," in *Workshop on Multi-Object Tracking.* IEEE, July 2001, pp. 3–10.

[7] B. Ozer and W. Wolf, "Video analysis for smart rooms," in *Internet Multimedia Management Systems II.* SPIE, 2001.

[8] B. Wilburn, N. Joshi, V. Vaish, M. Levoy, and M. Horowitz, "High speed video using a dense camera array," in *Computer Vision and Pattern Recognition*. IEEE, July 2004.

[9] M. Bramberger, M. Quaritsch, T. Winkler, B. Rinner, and H. Schwabach, "Integrating multi-camera tracking into a dynamic task allocation system for smart cameras," in *Int'l Conf. Advanced Video and Signal Based Surveillance.* Italy: IEEE, Sept. 2005.

[10] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty, "Model-integrated development of embedded software," *Proceeding of the IEEE*, vol. 91, no. 1, pp. 145–164, Jan. 2003.

[11] J. Hatcliff, X. Deng, M. Dwyer, G. Jung, and V. Ranganath, "Cadena: An integrated development, analysis, and verification environment for component-based systems," in *Int'l Conf. Software Engineering.*, 2003.

[12] G. J. Holzmann, *The Spin model checker*. Boston: Addison Wesley, 2004.

[13] S. Khan, O. Javed, and M. Shah, "Tracking in uncalibrated cameras with overlapping field of view," in *Performance Evaluation of Tracking and Surveillance, with CVPR 2001.* Hawaii: IEEE, Dec. 2001.

[14] S. Velipasalar and W. Wolf, "Recovering field of view lines by using projective invariants," in *Computer Vision and Pattern Recognition*, vol. Int'l Conf. Image Processing. Singapore: IEEE, Oct. 2004.

[15] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM.*, vol. 21, pp. 558–565, July 1978.

[16] J. Elson and K. Romer, "Wireless sensor networks: A new regime for time synchronization," *SIGCOMM Computer Communication Review.*, Jan. 2003.

[17] L. Lamport and P. M. Melliar-Smith, "Synchronizing clocks in the presence of faults," *Journal of the ACM.*, vol. 32, no. 1, pp. 52–78, Jan. 1985.

[18] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*, IEEE Std. 802.11, Nov. 1997.

[19] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. California: Morgan Kaufmann, 2004.

[20] J. Sztipanovits and G. Karsai, "Model-integrated computing," *IEEE Computer Magazine*, vol. 30, no. 4, pp. 110–112, Apr. 1997.