A Fault-Tolerant Distributed Vision System Architecture for Object Tracking in a Smart Room *

Deepak R. Karuppiah, Zhigang Zhu,
Prashant Shenoy, and Edward M. Riseman

Department of Computer Science,
University of Massachusetts,
Amherst, MA - 01003, USA
{deepak|zhu|shenoy|riseman}@cs.umass.edu

Abstract. In recent years, distributed computer vision has gained a lot of attention within the computer vision community for applications such as video surveillance and object tracking. The collective information gathered by multiple cameras that are strategically placed has many advantages. For example, aggregation of information from multiple viewpoints reduces the uncertainty about the scene. Further, there is no single point of failure, thus the system as a whole could continue to perform the task at hand. However, the advantages arising out of such cooperation can be realized only by timely sharing of the information between them. This paper discusses the design of a distributed vision system that enables several heterogeneous sensors with different processing rates to exchange information in a timely manner in order to achieve a common goal, say tracking of multiple human subjects and mobile robots in an indoor smart environment.

In our fault-tolerant distributed vision system, a resource manager manages individual cameras and buffers the time-stamped object candidates received from them. A User Agent with a given task specification approaches the resource manager, first for knowing the available resources (cameras) and later for receiving the object candidates from the resources of its interest. Thus the resource manager acts as a proxy between the user agents and cameras, thereby freeing the cameras to do dedicated feature detection and extraction only. In such a scenario, many failures are possible. For example, one of the cameras may have a hardware failure or it may lose the target, which moved away from its field of view. In this context, important issues such as failure detection and handling, synchronization of data from multiple sensors and sensor reconfiguration by view planning are discussed in the paper. Experimental results with real scene images will be given.

1 Introduction

In the recent years, rapid advances in low cost, high performance computers and sensors have spurred a significant interest in ubiquitous computing. Researchers are now talking about throwing in a lot of different types of sensors in our homes, work places and

* This work is supported by DARPA/ITO Mobile Autonomous Robots S/W (MARS) (Contract Number DOD DABT63-99-1-004) and Software for Distributed Robotics (SDR) (Contract Number DOD DABT63-99-1-0022)

even on people. They hope that the wealth of information from these sensors when processed and inferred carefully would significantly enhance our capacity to interact with the world around us. For instance, today, humans rely largely on their innate sensory and motor mechanisms to understand the environment and react to the various situations arising thereof. Though our intelligence is far superior to today's AI, the memory and number crunching capacity of an average person leaves much to be desired. But with a distributed backbone of processors and sensors augmenting our brain and senses, elaborate information gathering and complex and systematic decision-making could become possible for everyone. A smart environment could assist humans in their daily activities such as teleconferencing, surveillance etc. The smart environment idea is therefore not to replace a human but to augment one's capacity to do things in the environment. An interesting perspective into this area from the machine vision point of view has been provided in [14].

Distributed computer vision forms a vital component in a smart environment due to the rich information gathering capability of vision sensors. The collective information gathered by multiple cameras that are strategically placed has many advantages. For example, aggregation of information from multiple viewpoints reduces the uncertainty about the scene. Further, there is no single point of failure, thus the system as a whole could continue to perform the task at hand. However, the advantages arising out of such cooperation can be realized only by timely sharing of the information between them. The distributed system can then share the information to carry out tasks like inferring context, updating knowledge base, archiving etc. A distributed vision system, in general, should have the following capabilities

- Extraction of useful feature sets from raw sensor data
- Selection and fusion of feature sets from different sensors
- Timely sharing of information among the sensors
- Fault-tolerance and reconfiguration

This paper discusses the design of such a distributed vision system that enables several heterogeneous sensors with different processing rates to exchange information in a timely manner in order to achieve a common goal, say tracking of multiple human subjects as well as mobile robots in an indoor environment, while reacting at run-time to various kinds of failures, including: hardware failure, inadequate sensor geometries, occlusion, and bandwidth limitations. Responding at run-time requires a combination of knowledge regarding the physical sensorimotor device, its use in coordinated sensing operations, and high-level process descriptions.

1.1 Related Work

The proposed work is related to two areas in literature - multi-sensor network and distributed self-adaptive software. Research on multi-sensor network devoted to human tracking and identification can be found in [4], [13], [12], [14] and [15]. An integrated system of active camera network has been proposed in [16] for human tracking and face recognition. In [2], a practical distributed vision system based on dynamic memory has been presented. In our previous work [17], we have presented a panoramic virtual stereo for human tracking and localization in mobile robots. However, most of the current systems emphasize on vision algorithms, which are designed to function in a specific network. Important issues concerning fault-tolerance and sensor reconfiguration in a distributed system of sensors are seldom discussed.

These issues are addressed to some extent in the second area namely distributed self-adaptive software. Much of current software development is based on the notion that one can correctly specify a system a priori. Such a specification must include all input data sets, which is impossible, in general, for embedded sensorimotor applications. Self-adaptive software, however, modifies its behavior based on observed progress toward goals as the system state evolves at run-time [8]. Current research in self-adaptive software draws from two traditions, namely control theoretic and planning. The control theoretic approach to self-adaptive software treats software as a plant with associated controllability and observability issues [7]. Time-critical applications require the ability to act quickly without spending large amounts of time on deliberation. Such reflexive behavior is the domain of the control theoretic tradition. Drawing from the planning community, a generic software infrastructure for adaptive fault-tolerance that allows different levels of availability requirements to be simultaneously supported in a networked environment has been presented in [6]. In [1], a distributed control architecture in which run-time behavior is both pre-analyzed and recovered empirically to inform local scheduling agents that commit resources autonomously subject to process control specifications has been presented.

1.2 Architecture Overview

The proposed distributed vision system has three levels of hierarchy - sensor nodes ($S_1 ... S_N$), resource managers ($RM_1 ... RM_K$), and user agents ($UA_1 ... UA_M$), as shown in Fig. 1. The lowest level consists of individual sensors like omni-directional cameras and pan-tilt-zoom cameras, which perform human and face detection using motion, color and texture cues, on their data streams independently in (near) real-time. Each sensor reports its time-stamped object candidates (bearing, sizes, motion cues) to one or more resource managers at the next level. The communication protocol between the sensor and the resource manager could be either unicast or multicast. A resource manager acts as a proxy by making these object candidates available to the user agents at the topmost level. Thus the resource manager could serve many user agents simultaneously, freeing the sensors to do dedicated feature detection and extraction only. The user agent, in our application, matches the time-stamped object candidates from the most favorable sensors, estimates 3D locations and extracts tracks of moving objects in the environment. There could be other user agents that use the same or different sensor information but with a different task specification as well.

All the components of the system communicate using the Ethernet LAN. Thus Network Time Protocol (NTP) is used to synchronize the local clocks of the nodes after justifying that the synchronization resolution provided by NTP is sufficient for our task. For further details in implementation and applications of NTP, the reader is referred to [10] and [11].

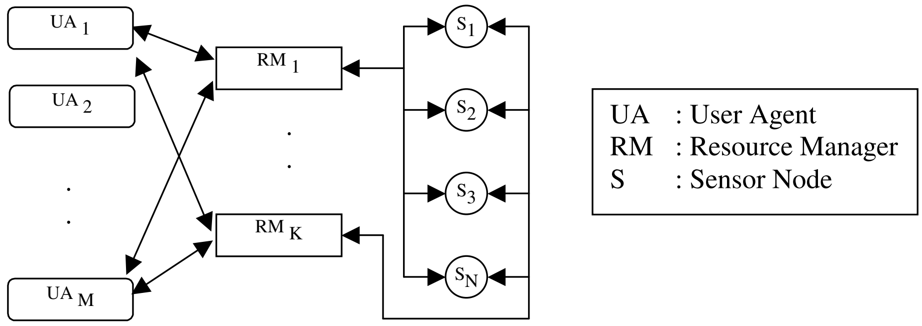


Fig. 1. System Architecture

2 Sensor Nodes

The typical flow of information at a sensor node is shown in Fig. 2. The lowest level is the sensor layer. A sensor node consists of a physical sensor and a processor. In general, the sensor node could also have motor capabilities, for example, a camera that could pan or a camera that is mounted on a robot. The processor could be a desktop computer or a simple embedded processor depending on the computational needs of the sensors. For example, a 68HC11 processor is sufficient for a simple pyro-electric sensor, which detects temperature changes. But a powerful desktop computer is needed for running algorithms for motion detection using vision sensors in real-time. In any case, the nodes should be able to connect to a Local Area Network (LAN). This enables them to communicate with the layer immediately above in hierarchy. Two such vision sensor nodes used in our human tracking system will be discussed in Sec. 6.

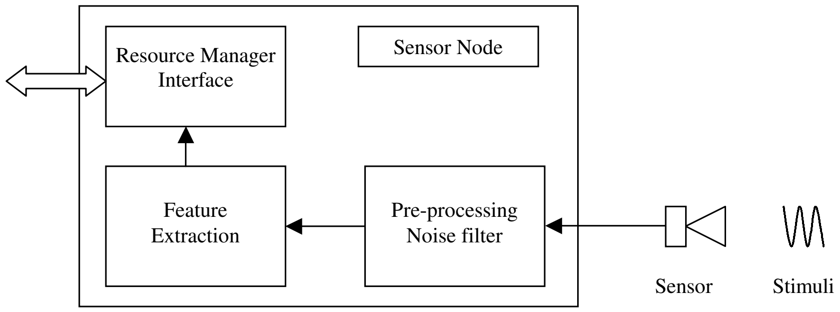


Fig. 2. Sensor Node

At the sensor level, the physical sensor perceives the environment, typically within a certain frequency spectrum of electromagnetic waves. The raw data is then, digitized by the device drivers of the sensor. The digitized data is pre-processed to get rid of random and systemic noise (if the noise models are available). The noise-free data is then used to extract useful features of interest. The features thus extracted are streamed out to the resource manager via the Resource Manager Interface (RM-Interface), shown in Fig. 2. The extracted feature set falls into two classes namely basic features and special features. Typically, basic features are common to all the sensor nodes, require low bandwidth and are streamed by default to the resource manager while the special features are node specific, require high bandwidth and are provided on-demand by user agents. For the task of tracking and localization of human subjects in a smart room, the basic features include the bearing and size of moving subjects while special features include their texture, color and motion cues. The special features can be used to match the objects across two cameras and thereby determine their 3-D location. Object matching using special features is further elaborated in Sec. 6.

When a sensor comes online, its RM-Interface reports to a resource manager, the sensor's unique ID followed by its location and geometry in a global reference frame. On receiving a confirmation from the RM, it activates the sensor's processing loop, which does the motion detection and periodically reports the feature. The RM-Interface is capable of receiving commands from the resource manager. Some commands are general like pausing operation or changing the reporting rate. Others are specific to the resource like motion commands to a PTZ platform or a mobile robot.

3 Resource Manager

A resource manager structure is shown in Fig. 3. The resource manager (RM) is the via-medium between the producers of information (the sensor nodes) and their consumers (the user agents). The resource manager keeps track of the currently available sensors and reports their status to the user agent periodically. The user agent chooses the best sensor subset from the available pool to accomplish its goals. The resource manager, therefore, acts as a proxy between the sensors and user agents. Thus the resource manager could serve many user agents simultaneously, freeing the sensors to do dedicated feature detection and extraction only. This way the sensors are not committed to a single agent, but could be shared among many agents. However, the motor functions of a node cannot be shared because they cause conflicts when more than one agent attempts to perform a motor task on the same sensor node. So, a lock manager manages the motor functions of a node. There is also the facility of maintaining multiple resource managers simultaneously (see Fig. 1). This improves fault-tolerance in the event of failure of a particular resource manager and improves performance in reducing load per resource manager. While using multiple resource managers, the better bandwidth utilization is achieved by employing multicast communication protocol. In such a scenario, a sensor node pushes the data on the wire only once addressing it to the multicast group to which the resource managers belong. The multicast backbone, then, efficiently routes the data to all the members of the group.

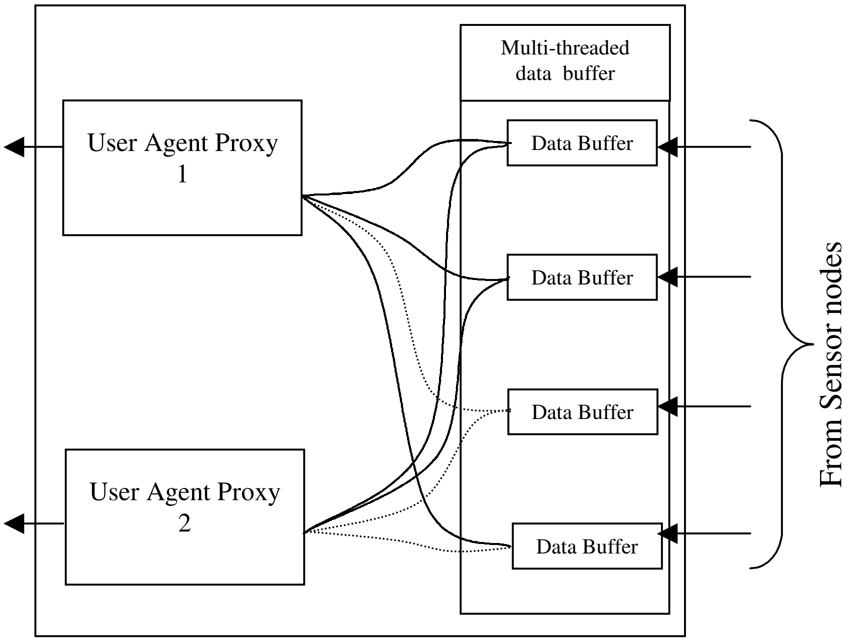


Fig. 3. Resource Manager

3.1 Multi-threaded Data Buffer

In the RM, the Multi-threaded Data Buffer (MDB) collects these time-stamped feature sets from different sensors and buffers them. When the RM-Interface of a sensor begins the registration process by sending its unique sensor ID, the MDB checks for the validity of the ID (i.e. if it is a trusted and known sensor). This simple security check prevents the MDB from being flooded by unscrupulous requests for service. After validation, a thread is spawned to serve that particular type of sensor. The buffering process that is crucial for synchronization of data from different sensors is discussed in Sec. 5.

3.2 User Agent Proxy

User Agent Proxy (UA-proxy) is the end point in the RM, which serves the user agents. Similar to MDB, UA-proxy registers and validates a user agent and a thread is spawned to begin the information exchange with the user agent. UA-proxy provides the user agents with the list of currently available sensors. Depending on a user agent's choice from this list, the UA-proxy delivers the corresponding sensor parameters and their time synchronized feature sets to the user agent. In Fig. 3, the solid lines between the UA-proxies and the data buffers in MDB indicate the respective user agent's choice of sensors.

The states of sensors are pushed to the user agents when one of the following occurs

- A new sensor has registered or,
- A sensor has failed.

Depending on this information, the User Agent modifies its choice set and informs the UA-Proxy which then responds accordingly. The User Agent can also request special features in addition to the default feature set from sensors via the Resource Manager.

4 User Agents

User Agents are processes that achieve a specific goal by using multiple sensorimotor resources. When they go about doing this, the availability of the required resources is not always guaranteed. Even if all the resources are available, the environmental context may prevent them from reaching the goal. This may require some action in the form of redeployment of sensors to handle the new context. User agents are thus adaptive to hardware, software and environmental contexts. A straightforward way to realize a user agent is to identify useful system configurations and assign a behavior to each configuration. Since the combinatorics of this procedure is prohibitive when there are a large number of sensors, an inductive method could be used. In this method, the system could learn from past behavior and identify useful system configurations by itself. An example of user agent used in our system will be given in Sec 7.

5 Time Synchronization Mechanism

The Resource Manager plays a vital role in making the sensors available to the user agents in a timely and fault-tolerant fashion. In this section we discuss how the features from multiple sensors are synchronized in time. A simple way to solve this would be to synchronize all the cameras using an electronic trigger signal. However, this hardware solution is not general enough to handle heterogeneous sensors and mobile sensors, which are an inevitable part of a smart environment. Even in the case of an all camera network, such synchronization could prove to be difficult and expensive, especially when there are many cameras scattered over a wide area. With the objective of providing a simple, inexpensive yet general solution, we propose a software solution exploiting the existing time synchronization protocol (NTP) running in today's computer networks.

Once the computer nodes are synchronized by NTP, the sensors attached to them could work independently and time-stamp their dataset based on their local clocks. These datasets are buffered at the multi-threaded data buffer (MDB) in the resource manager. When a report has to be sent to a UA, its UA-proxy in the resource manager queries the MDB for the current data from the user agent's sensor set. The query is parameterized by the current time. Each sensor's feature set that is closest in time may be returned. This method works when different sensors process at approximately the same rate. However, if their processing rates differ by a wide margin, this procedure could lead to errors.

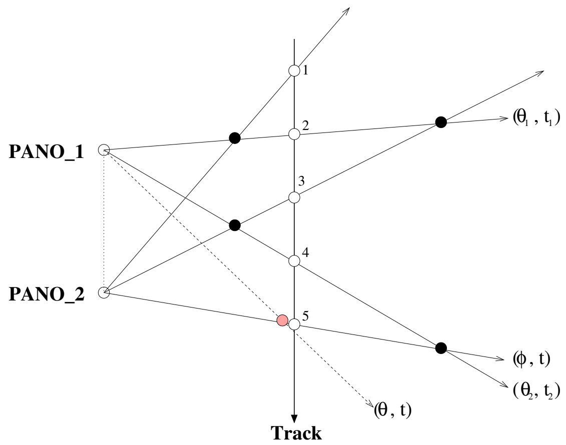


Fig. 4. Illustration of errors due to lack of synchronized matching across two sensors. PANO_2 detects the object at white circles 1, 3, & 5 while PANO_1 does at 2 & 4. The dark circles show the location estimates using the most recent bearings from the sensors while the gray circle shows the estimate at time t when interpolation is used

Let's demonstrate this by a simple example in which only the bearing angles to the moving subjects from the sensors are used. Suppose, two panoramic sensors PANO_1 and PANO_2 are registered in the RM and they report their bearings (to a single moving object in the scene) at different rates as shown in Fig. 4. If UA-proxy were to push the latest reported feature sets from the sensors to the UA, the result of fusion would be inaccurate, as shown in Fig. 4, because of timing discrepancy between the two sets. The white circles show the real positions of the moving object along the track. The dark circles show the error in triangulation caused due to matching datasets not synchronized in time.

So in such situations, using suitable interpolation techniques (polynomials or splines), the feature sets must be interpolated in time. Interpolation algorithm must also be aware of certain properties of the feature sets like the angular periodicity in the case of bearings. MDB is capable of interpolating the data buffer to return the feature set values for the time requested by the UA-proxy (see Fig. 5). This ensures the feature sets being used to fuse are close in time. However, it should be noted that for doing the interpolation, we need to assume that a linear or spline interpolation will approximate the motion of a human subject between two time instants, t_1 and t_2. In a typical walk of a human subject, the motion track can be approximated to be piecewise linear, so the bearing requested for time t in PANO_1 can be calculated as

$$\theta = \frac{(\theta_1 - \theta_2)}{(t_1 - t_2)} (t - t_1) + \theta_1. \tag{1}$$

where θ_1 and θ_2 are bearings measured in PANO_1 at time t_1 and t_2 respectively. The time t used in the above equation is the time for which a measurement of bearing, ϕ, is available from PANO_2. The angular periodicity of bearing angles has not been shown in Eqn. 1 for the sake of simplicity. However it has been incorporated in the interpolation calculations on the real system. We can see that the result of triangulation using bearings θ (in PANO_1) and ϕ (in PANO_2), represented by a gray circle in Fig. 4, has reduced the error considerably. Real scene examples will be given in Sec. 7.

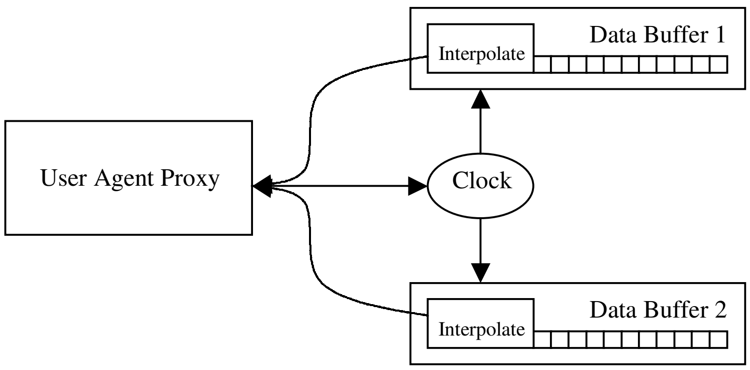


Fig. 5. Interactions between User Agent Proxy and Data Buffer

This algorithm will work provided all the different layers in the hierarchy have a global notion of time. Using Network Time Protocol (NTP), one could achieve time synchronization resolution of the order of 5ms among the nodes of a Local Area Network (LAN). If the fastest sensor in the system would take at least twice this time (every 10ms) to produce a report, then this resolution is acceptable. Typically this is valid because cameras have a frame rate of 25Hz with added overload in processing (i.e. a report at every 40ms at most from a camera).

6 Vision Sensor Nodes

6.1 Panoramic Camera Nodes

Effective combinations of transduction and image processing is essential for operating in an unpredictable environment and to rapidly focus attention on important activities in the environment. A limited field-of-view (as with standard optics) often causes the camera resource to be blocked when multiple targets are not close together and panning the camera to multiple targets takes time. We employ a camera with a panoramic lens [3] to simultaneously detect and track multiple moving objects in a full 360-degree view.

Figure 6 depicts the processing steps involved in detecting and tracking multiple moving humans. Four moving objects (people) were detected in real-time while moving in the scene in an unconstrained manner. A background image is generated automatically by tracking dynamic objects though the background model depends on the number of moving objects in the scene and their motion. Each of the four people were extracted from the complex cluttered background and annotated with a bounding rectangle, a direction, and an estimated distance based on scale from the sensor. The system tracks each object through the image sequence as shown in Fig. 6, even in the presence of overlap and occlusion between two people. The dynamic track is represented as an elliptical head and body for the last 30 frames of each object. The human subjects reversed directions and occluded one another during this sequence. The vision algorithms can detect change in the environment, illumination, and sensor failure, while refreshing the background accordingly. The detection rate of the current implementation for tracking two objects is about 5Hz.

With a pair of panoramic sensor nodes, 3D location of multiple moving objects can be determined by triangulation. This pair could then act as a virtual sensor node which reports the 3D location of moving objects to the resource manager. But there are two issues that must be considered namely, matching objects from widely seperated viewpoints and triangulation accuracy when the object is collinear with the cameras. Robust matching can be done by using special features from the sensor nodes like size, intensity and color histogram of the motion blobs in addition to the basic features [17]. When the object is in collinear configuration with the cameras (and if they are the only ones available at that instant), we employ the size-ratio method described in [17]. By using the ratio of the size of objects as seen from the two cameras, their distances from each camera could be calculated (see Fig. 7a). Figure 7b shows two subjects (walking in opposite directions on a rectangular track) being successfully tracked by the algorithm.

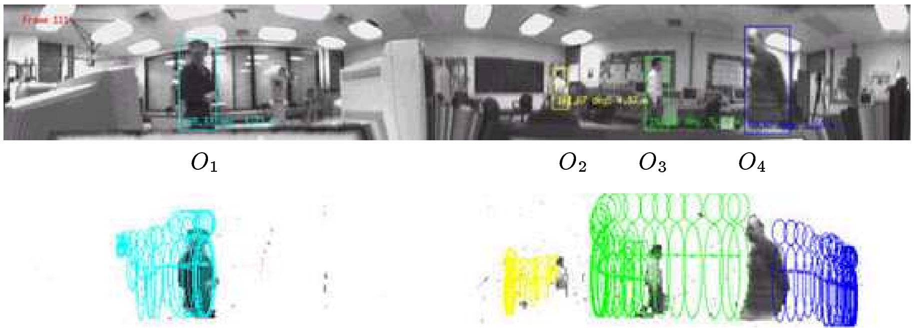


Fig. 6. Motion detection & tracking in a panoramic camera

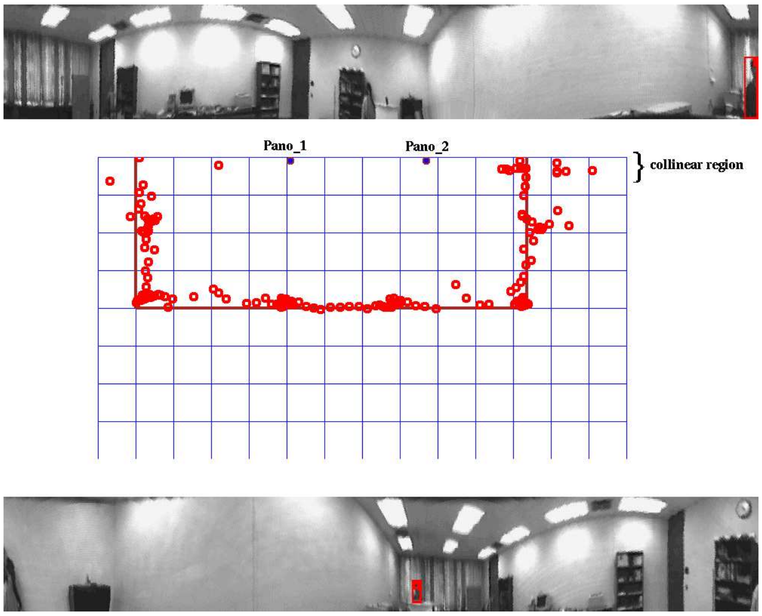
There are outliers in the track caused due to false detections and mismatches which can be rejected using the fault-tolerance mechanism described later in Sec. 7.1

6.2 Pan-Tilt-Zoom Camera Nodes

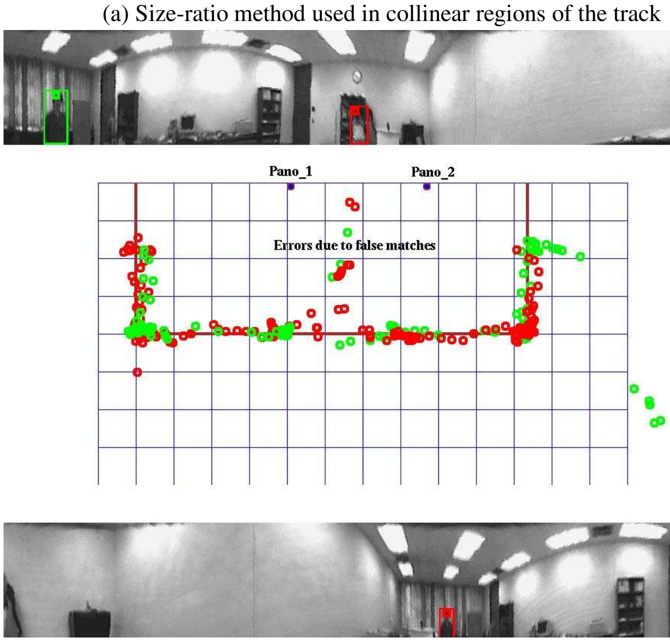
The PTZ cameras are another type of sensors used in our system. A PTZ camera can function in two modes - 1) motion detection, and 2) target tracking. In mode 1, the camera remains still and functions in exactly the same way as a panoramic camera, except that now it has a narrow field of view with high resolution. Though the area covered is very limited due to narrow field of view, it is better than panoramic camera for face detection. In mode 2, the camera gets information about a moving target from the higher level and tries to pursue the target. In this mode, it could continuously receive information from the higher level for target location, or it could take over tracking itself (see Fig. 8). Whenever a face is detected in its field of view, it could zoom in and snap a face shot of the moving person immediately. Face shots fall under the catergory of special features for this sensor node and are sent to user agents on demand.

7 Experimental Results

In this section, the implementation of the system as well as preliminary experimental results has been described. The smart room consists of four vision sensors to monitor the activities of human subjects entering the room. Our experiments were conducted using different arrangements of sensors which are shown in Figs. 11 & 13. Two of the sensors are panoramic cameras represented as Pano-I and Pano-II respectively, while the other two are Sony Pan-Tilt-Zoom (PTZ) cameras represented as PTZ-I and PTZ-II respectively. A single resource manager coordinates the cameras and reports their availability to the Track User Agent (Track-UA). The goal of this user agent is to track multiple humans in the smart environment and get face shots of the human subjects (see Fig. 9 and Fig. 10). The efficacy of the system can be evaluated based on two criteria, namely fault-tolerance and accuracy of the tracking. The former criterion evaluates the usefulness of hierarchical design while the latter evaluates vision algorithms and synchronization between multiple sensors.



(a) Size-ratio method used in collinear regions of the track



(b) False matches in the center of the track could be rectified by a third camera

Fig. 7. A virtual sensor node comprising of two panoramic cameras providing 3D locations of objects in the scene

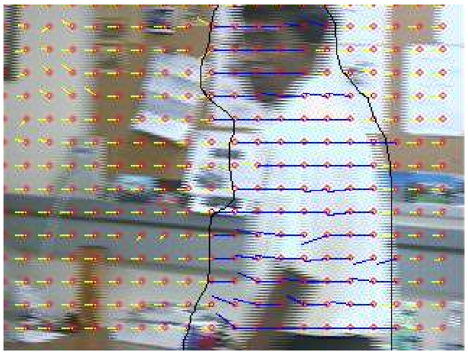


Fig. 8. Illustration of using flow to track moving object in a PTZ camera node

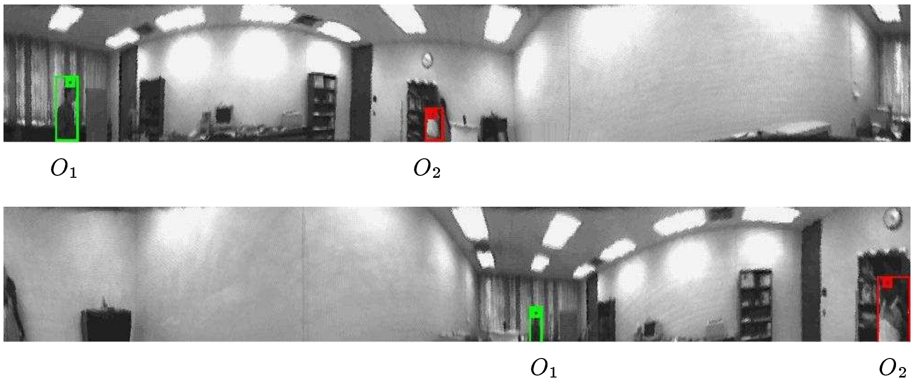


Fig. 9. Frames from PANO-I & PANO-II showing objects O_1 & O_2 being tracked

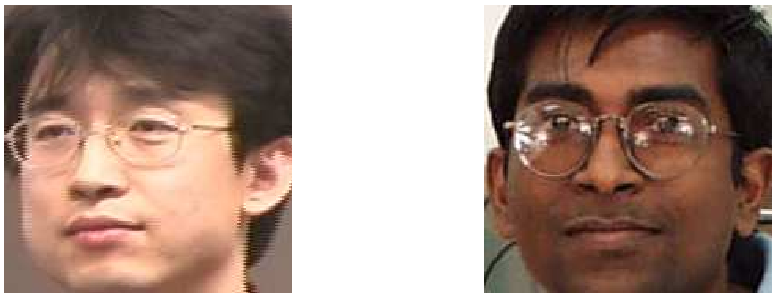


Fig. 10. Face shots of O_1 from PTZ-I and O_2 from PTZ-II respectively

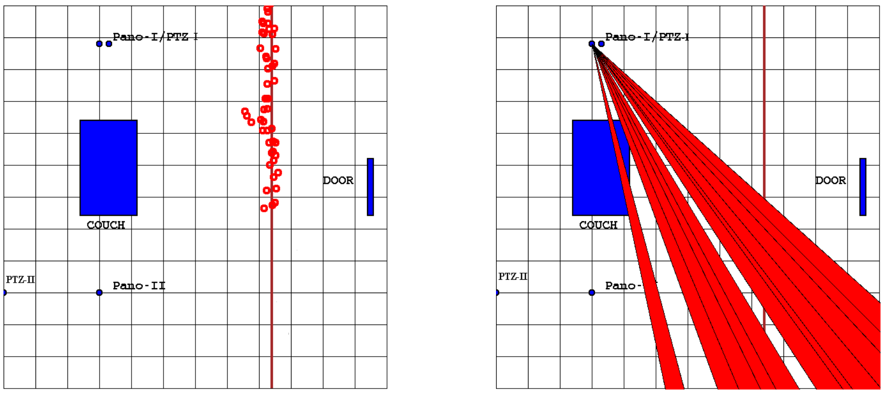
7.1 Fault-Tolerance Evaluation

The first criterion was evaluated by generating faults at the sensor nodes and observing how the system reacts. As explained in Sec. 3, the resource manager is at the core of the fault-tolerant operation of the system. In our system, the resource manager maintains an availability list. This list is pushed to the Track-UA upon occurrence of certain events like a sensor coming online or a sensor failure. The Track-UA uses the rule-based decision making engine shown in Table 1 to take appropriate actions.

Table 1. The rule-based decision making engine in Track-UA

Availability	Action
At least one camera	Keep track of the heading of the human subjects
Panoramic PTZ pair that are close to each other	Use the bearing information from panoramic camera to pan the PTZ towards the most dominant human subject
Two Panoramic and one PTZ	Match objects across the panoramic cameras. Triangulate to find the 3-D location of each matched object. Use the PTZ to look at one of the objects
Two Panoramic and two PTZ	Same as previous state, except assign the two of the objects to the two PTZ

The system reacting to the event of Pano-II failing is shown in Fig. 11. When both the panoramic cameras are available, the 3-D location of the moving subject can be estimated by triangulation (Fig. 11(a)). In this case any PTZ camera (PTZ-I or PTZ-II) can be assigned to focus on the human subject. However if Pano-II fails, we can only use Pano-I to estimate the bearing of the subject. If no other cameras are available for 3-D localization via triangulation, we can only use PTZ-I that is closely placed with Pano-I to obtain the face of the human subject.



(a) (b)

Fig. 11. Pano-I and Pano-II are available in (a). Pano-II failed in (b)

In Sec. 6.1, we showed two kinds of failures when only a pair of panoramic cameras is used for 3D-localization namely, poor triangulation at collinear conditions and stereo mismatches. Both these errors can be detected and can be easily handled by a third camera as shown in Fig. 12. Collinear conditions can be detected easily from the bearing angles. The triangulation accuracy under such conditions can be improved by either using a third camera or by using the size-ratio method when one is not available. A more difficult problem is the detection of false objects due to stereo mismatches. Again by verifying each object candidates with a third camera, false ones can be rejected.

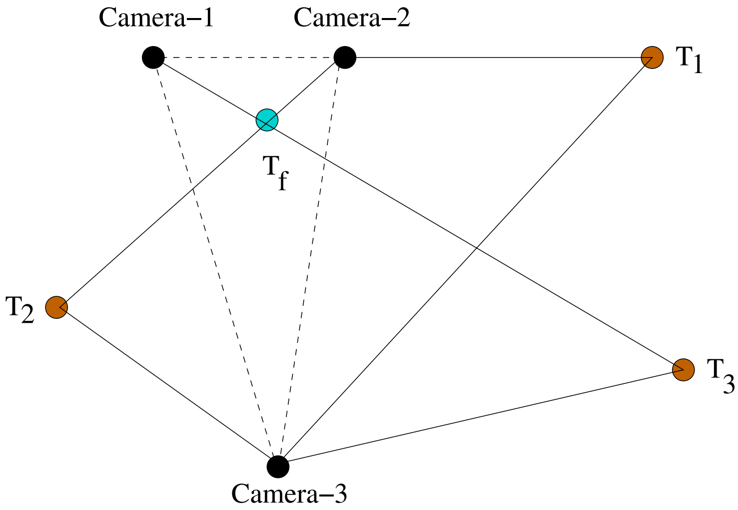
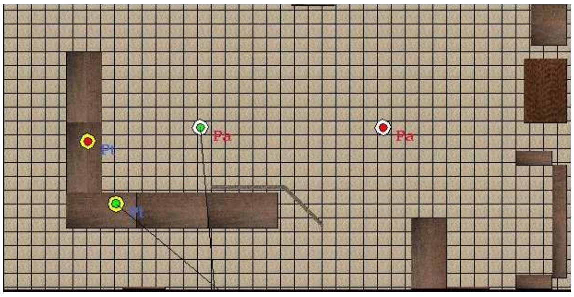
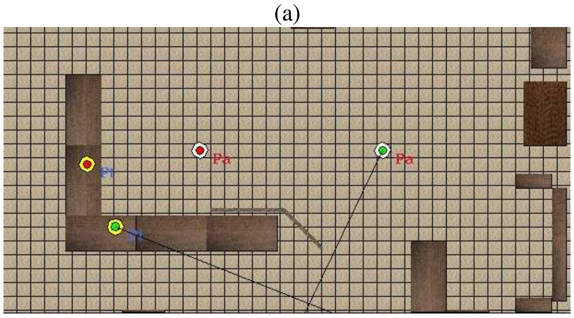


Fig. 12. Object T_1 is collinear with Camera-1 and Camera-2 but can be localized by (Camera-1,Camera-2) pair using size-ratio method or better by either (Camera-1,Camera-3) pair or (Camera-2,Camera-3) pair using triangulation method. A mismatch could result in the false object T_f which can easily be discarded when verified by Camera-3

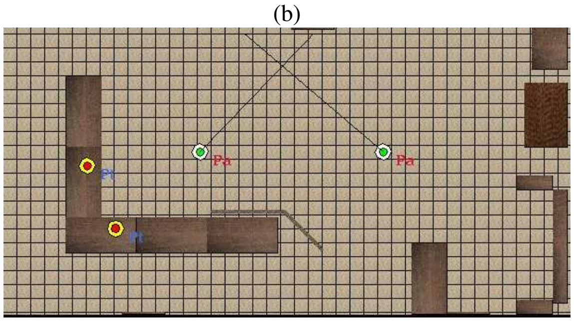
In a multi-sensor network, information from many cameras can thus provide a better degree of fault-tolerance and allow for dynamic resource reconfiguration to perform a particular task. Results from a real run are shown in Fig. 13. Figure 13a shows two sensors triangulating at the moving object. But as the object moves further to the right, it is occluded from the view of one of the sensors. So in Fig. 13b, we observe that another sensor is brought in to continue tracking. The same process is repeated for the case of collinear sensor geometry in Figs. 13c & 13d. We have discussed fault-tolerance by sensor reconfiguration in more detail in [5].



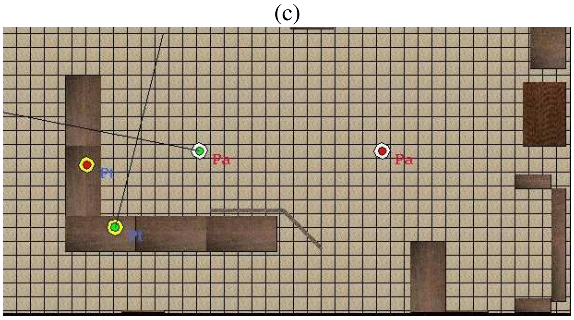
(a)



(b)



(c)

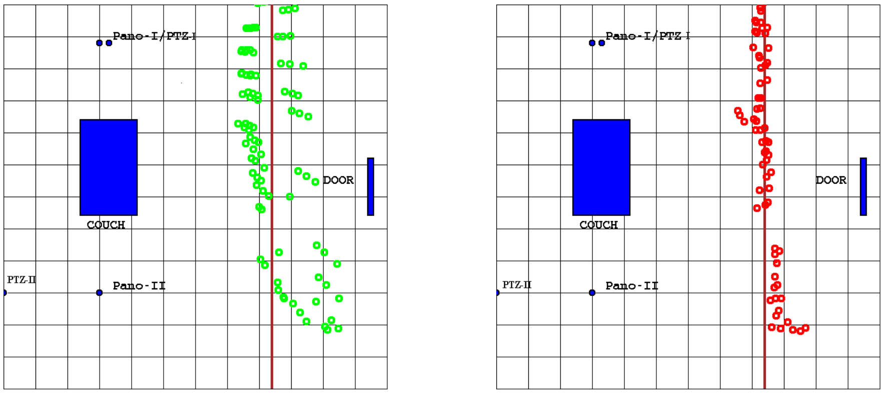


(d)

Fig. 13. Frames from a tracking task demonstrating pair-wise sensor mode changes induced on occurrence of faults like object track lost by one of the sensors (a-b), or collinear sensor geometry (c-d)

7.2 Synchronization Results

In this experiment, a person walked along a pre-determined path at a constant velocity and two panoramic cameras - Pano-I and Pano-II are used to track the motion (Fig. 14). Before the start of the experiment, the local clocks on all the sensor nodes are synchronized using NTP. In this experiment, Pano-II is set to process twice as fast as Pano-I. The result of target tracking under this situation is shown in Fig. 14a. We can notice that the error in the track result is quite large with a mean of around 60cm due of lack of synchronization. After we employed the interpolation method discussed in Sec. 5, the mean localization error is reduced within 15cm (Fig. 14b).



(a) (b)

Fig. 14. (a) Unsynchronized tracking and, (b) Synchronized tracking

8 Conclusion

A distributed sensor network architecture comprising of three levels of hierarchy has been proposed in this paper. The hierarchy consists are sensor nodes, resource manager and user agents. The resource manager acts as a proxy between the sensor nodes and the user agents allowing many user agents to simultaneously share sensor resources. The system was implemented using two types of vision sensors, namely panoramic cameras and pan-tilt-zoom cameras. The system was evaluated for its fault-tolerance performance and accuracy of tracking. A simple, cost-effective way of synchronizing data streams from heterogeneous sensors using NTP was discussed and the experimental results showed the practical utility of this approach.

Given the general nature of the proposed architecture, it is possible to add different types of sensors such as acoustic and pyroelectric sensors, and use them to perform a variety of tasks. The system will be extended to realize its full potential of having multiple user agents, each pursuing a specific goal in the smart environment, simultaneously using multiple resource managers in the multi-sensor framework. The system could be further extended to provide a human interface by building semi-autonomous user agents. A semi-autonomous user agent could interact with a human who can make decisions. While acting under guidance, the agent can learn and increase its confidence in handling certain situations. When it encounters similar situations in the future it can autonomously act without guidance.

Acknowledgements

We would like to thank other members of our research team - Gary Holness, Subramanya Uppala, S. Chandu Ravela, and Prof. Roderic Grupen - for their involvement in the development of this research. We would also like to thank Yuichi Kikuchi for his contribution to the development of the Pan-Tilt-Zoom sensor node.

References

[1] D. R. Karuppiah et al. Software mode changes for continuous motion tracking. In P. Robertson, H. Shorbe, and R. Laddaga, editors, *Self-Adaptive Software*, volume 1936 of *Lecture Notes in Computer Science*, Oxford, UK, April 17-19 2000. Springer Verlag.

[2] T. Matsuyama et al. Dynamic memory: Architecture for real time integration of visual perception, camera action, and network communication. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, Hilton Head Island, SC, June 2000.

[3] P. Greguss. Panoramic imaging block for three-dimensional space. U.S. Patent 4,566,763, January 1986.

[4] I. Haritaoglu, D. Harwood, and L. S. Davis. W4: Real-time system for detection and tracking people in 2.5d. In *Proc. of the 5th European Conf. on Computer Vision*, Freiburg, Germany, June 1998.

[5] G. Holness, D. Karuppiah, S. Uppala, R. Grupen, and S. C. Ravela. A service paradigm for reconfigurable agents. In *Proc. of the 2nd Workshop on Infrastructure for Agents, MAS, and Scalable MAS*, Montreal, Canada, May 2001. ACM. To appear.

[6] Z. T. Kalbarczyk, S. Bagchi, K. Whisnant, and R. K. Iyer. Chameleon: A software infrastructure for adaptive fault tolerance. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):1–20, June 1999.

[7] M. Kokar, K. Baclawski, and Y. A. Eracar. Control theory based foundations of self controlling software. *IEEE Intelligent Systems*, 14(3):37–45, May 1999.

[8] R. Ladagga. Creating robust-software through self-adaptation. *IEEE Intelligent Systems*, 14(3):26–29, May 1999.

[9] K. Marzullo and S. Owicki. Maintaining the time in a distributed system. *ACM Operating Systems Review*, 19(3):44–54, July 1985.

[10] D. L. Mills. Internet time synchronization: The network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, October 1991.

[11] D. L. Mills. Improved algorithms for synchronizing computer network clocks. *IEEE/ACM Transactions on Networks*, 3(3):245–254, June 1995.

[12] A. Nakazawa, H. Kato, and S. Inokuchi. Human tracking using distributed vision systems. In *In 14th International Conference on Pattern Recognition*, pages 593–596, Brisbane, Australia, 1998.

[13] Kim C. Ng, H. Ishiguro, Mohan M. Trivedi, and T. Sogo. Monitoring dynamically changing environments by ubiquitous vision system. In *IEEE Workshop on Visual Surveillance*, Fort Collins, Colorado, June 1999.

[14] A. Pentland. Looking at people: Sensing for ubiquitous and wearable computing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):107–119, January 2000.

[15] T. Sogo, H. Ishiguro, and Mohan. M. Trivedi. *Panoramic Vision: Sensors, Theory and Applications*, chapter N-Ocular Stereo for Real-time Human Tracking. Springer Verlag, 2000.

[16] Mohan M. Trivedi, K. Huang, and I. Mikic. Intelligent environments and active camera networks. *IEEE Transactions on Systems, Man and Cybernetics*, October 2000.

[17] Z. Zhu, K. Deepak Rajasekar, E. Riseman, and A. Hanson. Panoramic virtual stereo vision of cooperative mobile robots for localizing 3d moving objects. In *Proceedings of IEEE Workshop on Omnidirectional Vision - OMNIVIS'00*, pages 29–36, Hilton Head Island, SC, June 2000.