A FAULT-TOLERANT DISTRIBUTED VISION SYSTEM ARCHITECTURE FOR OBJECT TRACKING IN A SMART ROOM[*]

Deepak R. Karuppiah[1], Zhigang Zhu, Prashant Shenoy, Edward M. Riseman

Department of Computer Science, University of Massachusetts at Amherst, Amherst 01003

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.

[*] This work is supported by DARPA/ITO Mobile Autonomous Robot S/W (MARS) (Contract Number DOD DABT63-99-1-0004) and Software for Distributed Robotics (SDR) (Contract Number DOD DABT63-99-1-0022)

[1] Email: deepak@cs.umass.edu

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 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 [13].

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, [2], [5], [12], [13] & [14]. An integrated system of active camera network has been proposed in [15] for human tracking and face recognition. In [9], a practical distributed vision system based on dynamic memory has been presented. In our previous work [16], 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[7]. 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[6]. 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 [3]. In [4], 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 ($S_1 \ldots S_N$), resource managers ($RM_1 \ldots RM_K$), and user agents ($UA_1 \ldots UA_M$), as shown in Figure 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. 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] & [11].

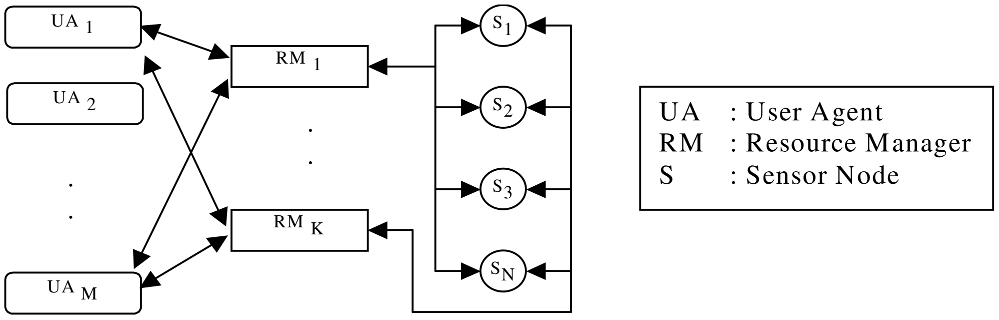


Figure 1 System Architecture

2 Sensor Nodes

The typical flow of information at a sensor node is shown in Figure 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 of the computational needs of 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 Section 6.

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 Figure 2. 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.

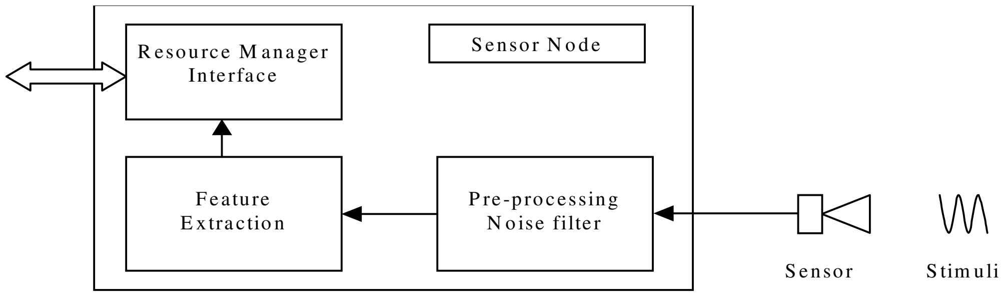


Figure 2 Sensor Node

3 Resource Manager

A resource manager structure is shown in Figure 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 Figure 1). This improves fault-tolerance in the event of failure of a particular resource manager and improves performance in reducing load per resource manager.

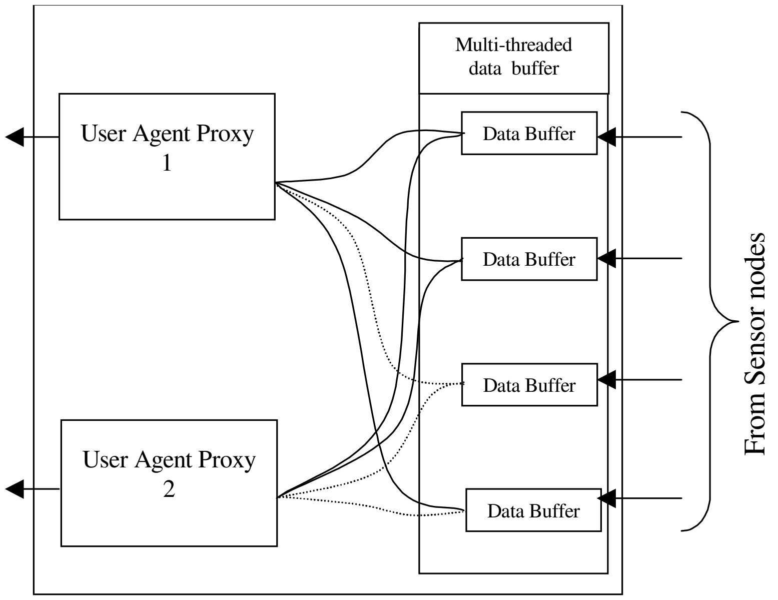


Figure 3 Resource Manager

3.1 Multi-threaded data buffer

Each moving object has certain distinguishing features like direction with respect to the camera, size and average intensity of the object in the image. These features can be used to match the objects across two cameras and thereby determine their 3-D location. 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 Section 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 Figure 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.

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. Another way to handle the combinatorics is to make the user agent semi-autonomous. A semi-autonomous user agent can 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. An example of user agent used in our system will be given in Section 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.

Let's demonstrate this by a simple example. 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 Figure 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 Figure 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. MDB is capable of interpolating the data buffer to return the feature set values for the time requested by the UA-proxy (see Figure 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

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

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. 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 Figure 4, has reduced the error considerably. Real scene examples will be given in Section 0.

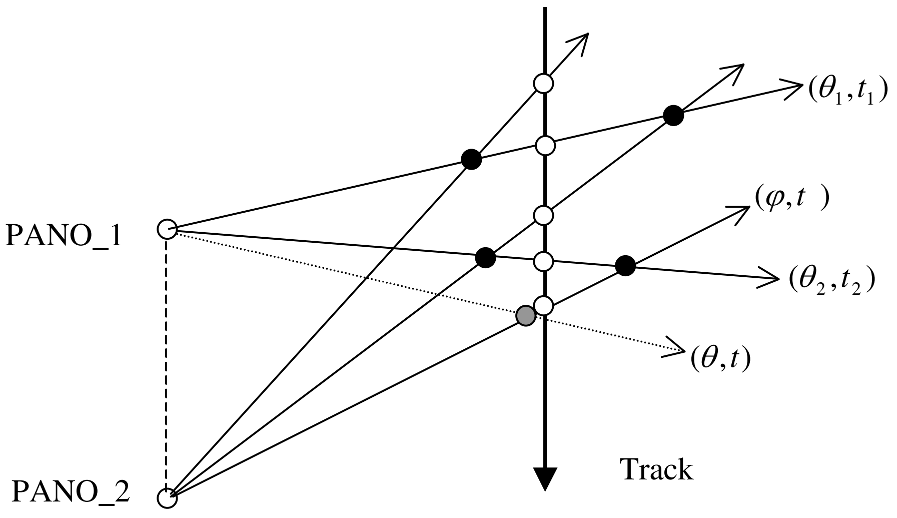


Figure 4 Illustration of errors due to lack of synchronized matching across two sensors

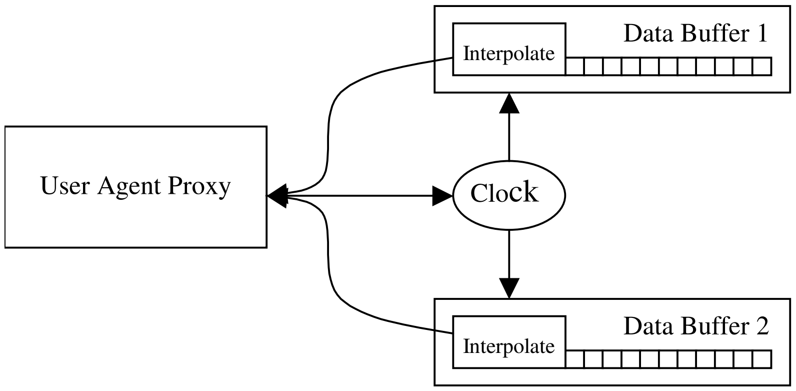


Figure 5 Interactions between User Agent Proxy and Data Buffer

This algorithm will work provided all the different layers in the hierarchy have a global notion of time. Using Network Time Protocol (NTP) 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).

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 [1] to simultaneously detect and track multiple moving objects in a full 360-degree view.

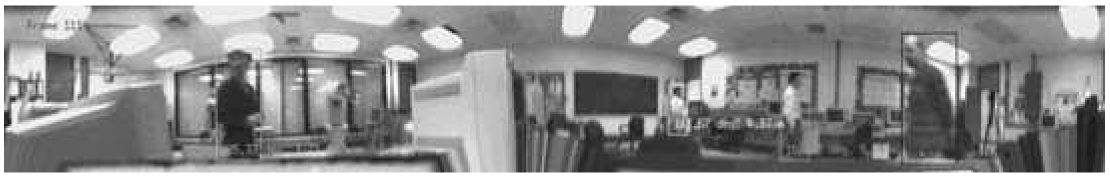


Figure 6 Motion detection in a panoramic camera

Figure 7 Human tracking in a panoramic camera

Figure 6 and Figure 7 depict 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 Figure 7, 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.

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, & 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 Figure 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.

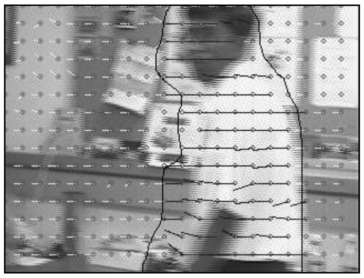


Figure 8 Illustration of using flow to track moving object in a PTZ camera node

7 Experimental Results

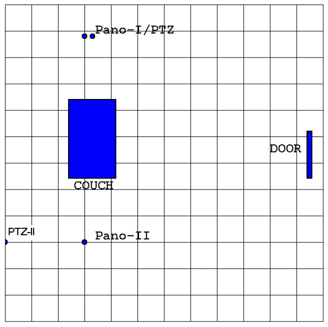


Figure 9 Layout of the smart room showing the positions of various sensor nodes. Each grid is of size 50cm by 50cm.

In this section, the implementation of the system as well as preliminary experimental results has been described. The layout of the room is shown in Figure 9. The smart room consists of four vision sensors to monitor the activities of human subjects entering the room. 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 Figure 10 - Figure 12). 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.

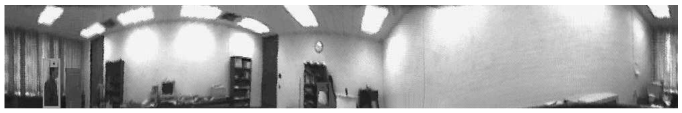


Figure 10 Tracking in Pano-I

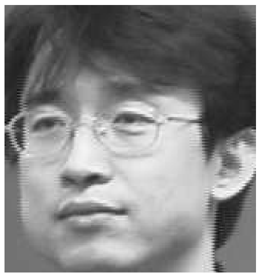


Figure 11 Face shot from PTZ-I

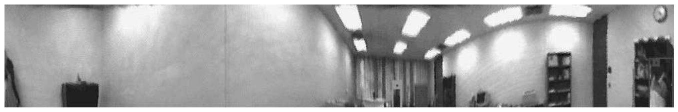


Figure 12 Tracking in Pano-II

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 Section 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 the table below 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 Figure 13 &. Figure 14. When both the panoramic cameras are available, the 3-D location of the moving subject can be estimated by triangulation (Figure 13). 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.

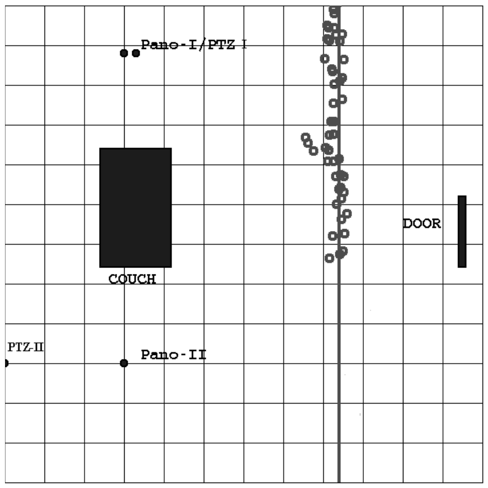


Figure 13 Pano-I and Pano-II are available

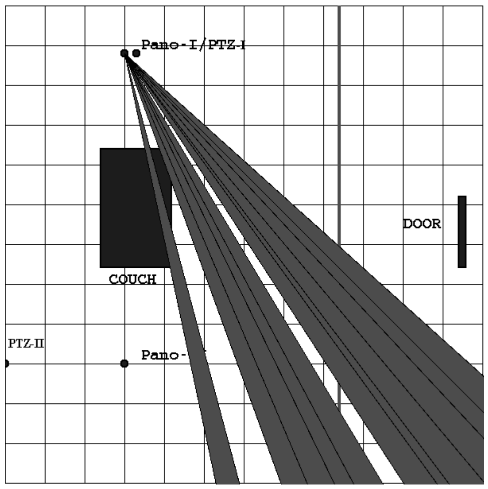
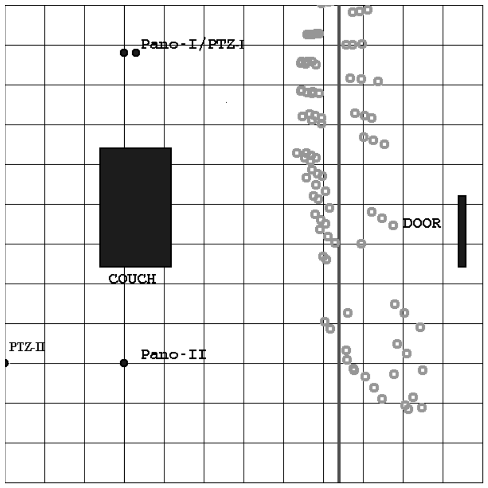


Figure 14 Pano-II failed.

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 (Figure 15 & Figure 16). 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 Figure 15. 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 Section 5, the mean localization error is reduced within 15cm (Figure 16).



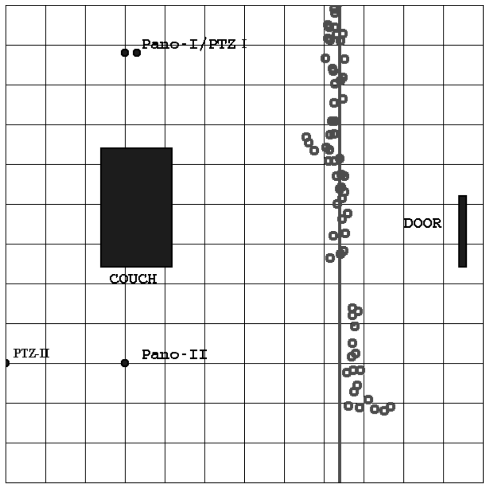


Figure 15 Unsynchronized Tracking

Figure 16 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 currently 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 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.

References

[1] Greguss, P. Panoramic imaging block for three-dimensional space, U.S. Patent 4,566,763 (28 Jan, 1986).

[2] Haritoglu, I., D. Harwood and L. Davis. W4: Real-time System for Detection and Tracking People in 2.5D. *ECCV,* 1998.

[3] Kalbarczyk, Z., *et al*, Chameleon: A Software Infrastructure for Adaptive Fault Tolerance. *IEEE Transactions on Parallel and Distributed Systems,* 10(6): 1-20, JUNE 1999.

[4] Karuppiah, D., *et al.* Software Mode Changes for Continuous Motion Tracking, *Active Software Composition Workshop*, Oxford, UK, April 2000.

[5] Kim C. Ng, H. Ishiguro, Mohan M. Trivedi, and T. Sogo, "Monitoring Dynamically Changing Environments by Ubiquitous Vision System," IEEE Workshop on Visual Surveillance, Fort Collins, Colorado, June 1999

[6] Kokar, M., K. Baclawski and Y. A. Eracar. Control Theory Based Foundations of Self Controlling Software, *IEEE Intelligent Systems,* 14(3): 37-45, May 1999.

[7] Ladagga, R. Creating Robust-Software Through Self-Adaptation. *IEEE Intelligent Systems*. 14(3): 26-29, May 1999.

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

[9] Matsuyama, T., *et al.* Dynamic Memory: Architecture for Real Time Integration of Visual Perception, Camera Action, and Network Communication. *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* Hilton Head Island, Vol.2, 728-735, JUNE 2000.

[10] Mills, D.L. Improved algorithms for synchronizing computer network clocks. *IEEE/ACM Trans. Networks* 3(3): 245-254, June 1995.

[11] Mills, D.L. Internet time synchronization: the Network Time Protocol. *IEEE Trans. Communications* 39(10): 1482-1493 October 1991

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

[13] Pentland, A. Looking at People: Sensing for ubiquitous and wearable computing. *IEEE Trans. Pattern Analysis and Machine Intelligence.* 22(1): 107-119, January 2000.

[14] Sogo, T., H. Ishiguro, M. M. Trivedi. N-Ocular Stereo for Real-time Human Tracking. *Panoramic Vision: Sensors, Theory and Applications*, (R. Benosman and S. B. Kang, eds.), Springer Verlag, 2000.

[15] Trivedi, M., K. Huang, I. Mikic. Intelligent Environments and Active Camera Networks. *IEEE Systems, Man and Cybernetics*, October 2000.

[16] Zhu, Z., K. D. Rajasekar, E. Riseman, A. Hanson. Panoramic Virtual Stereo Vision of Cooperative Mobile Robots for localizing 3D Moving Objects. *Proceedings of IEEE Workshop on Omnidirectional Vision – OMNIVIS'00.* Hilton Head Island, 29-36, JUNE 2000.