A Cost-effective Multiple Camera Vision System using FireWire Cameras and Software Synchronization

Piyush Kumar Rai
Undergraduate Student
Computer Science and Engg.,
IT-BHU, Varanasi-5,
UP, India.
rai_piyush@yahoo.co.in

Kamal Tiwari
Undergraduate Student
Computer Science & Engg.,
IIT Kanpur,
Kanpur, UP, India.
kamalt@iitk.ac.in

Prithwijit Guha
PhD. Student
Electrical Engg.,
IIT Kanpur,
Kanpur, UP, India.
pguha@iitk.ac.in

Amitabha Mukerjee
Professor
Computer Science and Engg.,
IIT Kanpur,
Kanpur, UP, India.
amit@cse.iitk.ac.in

Abstract:

In this paper, we describe a cost-effective Multiple-Camera Vision system using low cost simple FireWire web cameras. The FireWire cameras, like other FireWire devices operate on the high speed FireWire bus. Current supported bandwidth is 400 Mbps. Right from its introduction, the FireWire (synonymously known as IEEE 1394) bus interface specification has proved its capabilities and has been supported by both developers and users. Due to its low cost and ease in connecting, FireWire has been recommended as the technology to be used in Machine-Vision systems and Image-processing applications. We have developed a Multiple-camera synchronized Vision system using FireWire cameras. The synchronization has been achieved using "Software Triggering" which has been discussed in the paper. Possible applications of such a system have also been discussed in the paper. Our system does away with the need of costly cameras and frame-grabber cards which are generally used in conventional multiple-camera systems. The direct pixel-to-data correspondence without the need of a frame grabber or classic synchronization systems (requiring Hardware synchronization mechanisms) justifies the novelty of such a system. Calibration of the multiple camera system has also been discussed.

I .Introduction:

Vision is seen as the primary input for robot applications that need to be performed in short time and efficiently. Keeping this in mind, vision researchers around the world have been working on developing efficient algorithms and vision systems that can accomplish this task. As for vision systems are concerned, several panoramic and stereo vision systems have been developed in the past. Many multiple-camera systems have been developed which synchronization mechanisms (which hinder the mobility of such systems).

Hence, in our approach, we are using simple and low-cost FireWire web cameras for image-acquisition, which operate on FireWire (IEEE 1394). The IEEE 1394 specification offers a bandwidth of 400 Mbps, which is suitable for such systems dealing with large amounts of data transfer. These cameras do not have the capability of hardware synchronization as in case of high-cost "pan and tilt" cameras, so we propose a software based synchronization mechanism for such a system using a server-client model.

II. Features of the system:

The system offers following advantages over the existing systems:

Cost effectiveness: The FireWire web cameras used in the setup are much cheaper than the traditional high-cost pan and tilt cameras and frame-grabber cards.

Scalability: Any number of cameras can be added to such a system (of course, limited by the capability of communication channel, through which the individual cameras communicate to the server PC). Also, any number of cameras can be removed from the system without affecting the system. This makes the system entirely reconfigurable.

Software Synchronization: Provides a synchronized capture capability, which is very crucial for such multiple camera systems.

Setup and Hardware:

The setup has a 4 meter cubical covered with green curtains from three sides to minimize the noise due to light intensity fluctuations. Fourth side of the cubical is a calibrated white display screen controlled by a projector which is placed outside the cubical. An integral part of this setup is a set of three firewire cameras mounted on strategic position across this cube so as to cover the entire workspace within. The camera calibration is through the technique of SELF CALIBRATION which will accomplish the task of calibration in bits of seconds. This setup will help an accurate background subtraction and silhouette estimation. The 3 FireWire cameras mounted on 3 different machines running on Linux act as client PCs and send the captured images to a centralized server, which controls and coordinates the client PCs.

The library libraw1394 provides direct access to the IEEE 1394 bus through the Linux 1394 subsystem's raw1394 user space interface. Another library libdc1394 is intended to provide a high level programming interface for application developers who wish to control IEEE 1394 based cameras that conform to the 1394-based Digital Camera Specification. However, we wanted to make a new API on top of the existing API so that it is easier to use the cameras for image acquisition and processing in our application programs. So, instead of writing the lengthy routines, the users can call the simple functions in the new API and at the same time remain transparent to the lower-level functions.

Initially, the server waits for connections from different clients. When the connections are established, it triggers all the clients simultaneously for grabbing the images. The clients grab the images and send the images/processed results from images to the server where the server displays them after combining the results from clients.

Issues in Multiple camera setup

The server triggers the clients simultaneously but whether they get triggered at the same time or not, depends upon several factors. We shall discuss them one by one:

Network Latency*:* The server can send a multicast message to all the client. However, the difference of time in getting this message by first and last client will depend upon the current load of the network. In an unloaded network, this latency may be very small (~ 1ms), but for heavily loaded network, these delays may be indeterminist and significant.

Scheduling in Linux*:* The Linux kernel is non-preemptive. So, it is the Linux kernel which dictates how fast the clients can respond to the trigger signal by the server. Present Linux kernels have a time-slice of 10 ms which can cause different clients to be marked-off by the multiples of 10 ms. Running the clients with a real-time priority would be a way to get a deterministic behavior from the system but the standard Linux kernel doesn't support this. However, the RTLinux patch can be used for running our processes in Real-Time.

Camera specific features*:* Delays may also be caused by the camera drivers and the hardware. One can not know precisely how long does it take for the camera to start grabbing the frame after getting the trigger signal from the client. We assume this delay to be same for all the clients.

Synchronization of clients: Our setup uses cheap FireWire cameras, which do not have support for Hardware synchronization like the costly SONY DFW-V5000 digital camera. Some other FireWire based cameras do provide the support for external hardware based triggering, but these are much costly than the simple ADS PYRO camera we are using. Hardware triggering would require additional hardware and cabling which would hinder the mobility of the system. Therefore, we propose a software based

synchronization. The scheme is given below.

Software Synchronization: In our setup there are currently one server and three clients. Each client has a FireWire camera. The camera captures a 320X240 frame in YUV422 format, converts it to 320X240 RGB and sends to the server for further processing. For processing we need data coming from different clients to be synchronized, that is data sent from different clients should correspond to same time coordinate. For achieving this goal we need to synchronize the clocks of all the clients with the clock of the server. One way to do this is to simply send the time of the server clock to all the clients and setting the time of the client to the time value send by the server. But the problem in this approach is that the time taken to send time related data from server to client requires time, which adds a substantial discrepancy in synchronization of clients. This problem is solved by calculating the time for sending data from server to client over the network by sending some test data and calculates the network "lag". Now "lag" is added to server time and sent to all the clients. Clients accept this value as their time value thereby synchronizing their clocks with the server time.

Display of the data received: At the server, we have different processes running for different clients. So to display the data received by these processes at one place we need these processes to communicate with each other. For inter-process communication we have used the appropriate functions. Using them, we created a pool of shared memory and all the processes shared this pool. Now the child process running on the server, which is necessarily dedicated to a client, receives data from the client and writes it in the shared memory part described by a "SHMID" and also sets the flag corresponding to this client as "1" which is an indication to the parent process that data is available for this particular client. When value of this flag becomes "1" for all the child processes, it means that the synchronous data from all the clients is available and server may read it from the shared memory part described by the same value of "SHMGET". Once the data is read by the parent process, it sets the value of flags to "0" to ask more data from the child processes.

Synchronized Image Acquisition: The software architecture for this setup is shown in the next figure. There is one server, which has three processes running. One is the main process, which gives the command to clients for image grabbing and controls and coordinates the clients for synchronous image capture. One process uses the frames stored in the shared memory, does the required image processing and stores the processed images again in the shared memory. Another process also uses theshared memory and displays the images sent by all the clients or the processed image results.

The server synchronizes the clients according to its clock at the start of the program.

Synchronization Error measurements: Because of the reasons responsible for improper synchronization, the Clients may not be perfectly synchronized. As a test for this, we made a small test program which used to print numbers in an increasing order, every milliseconds. We put two cameras in front of the computer screen and let them grab the computer screen images when the test program is running. For small frame-rates, sometimes, the both images differed a bit(by around 5 milliseconds in some cases), but with 30 fps, that we were using in our experiments they were perfectly matching. Synchronization would considerably improve if we run the client processes with real-time priority.

Camera Calibration: Calibration of the cameras was done through self-calibration technique to avoid the manual work in

calibrating the setup every time before running an application in the setup.

III. Applications

We used our own custom made libraries for image processing routines. They include several APIs for image processing applications. All are written in C++. As our setup was ready with continuous image grabbing, we did several image processing experiments in Real-Time using a single camera as well as multiple camera setup. These included Online Edge Detection, Background subtraction, Histogram calculation, Meanshift Tracking, Skin Detection etc. For Tracking, we used the Mean Shift approach. In this approach, the most probable target position is found out in the current frame. The difference between the target model and the target candidates is expressed by a term obtained using "The Bhattacharya Coefficient". We implemented the single Meanshift tracker as well as the Multiple Meanshift Tracker, where three different part of the body (e.g. left & right hands and the face) could simultaneously be tracked. The results are shown at the server for all the three views.

The system was calibrated and used in 3D reconstruction and applications such as 3D Immersive environments. The calibration is achieved using the self-calibration mechanism. This method of calibration does not use a calibration object. By moving a camera in a static scene, the rigidity of the scene provides in general two constraints on the camera's parameters from one camera displacement by using image information alone. Three images taken by a same camera with fixed intrinsic parameters are sufficient to recover both intrinsic and extrinsic parameters.

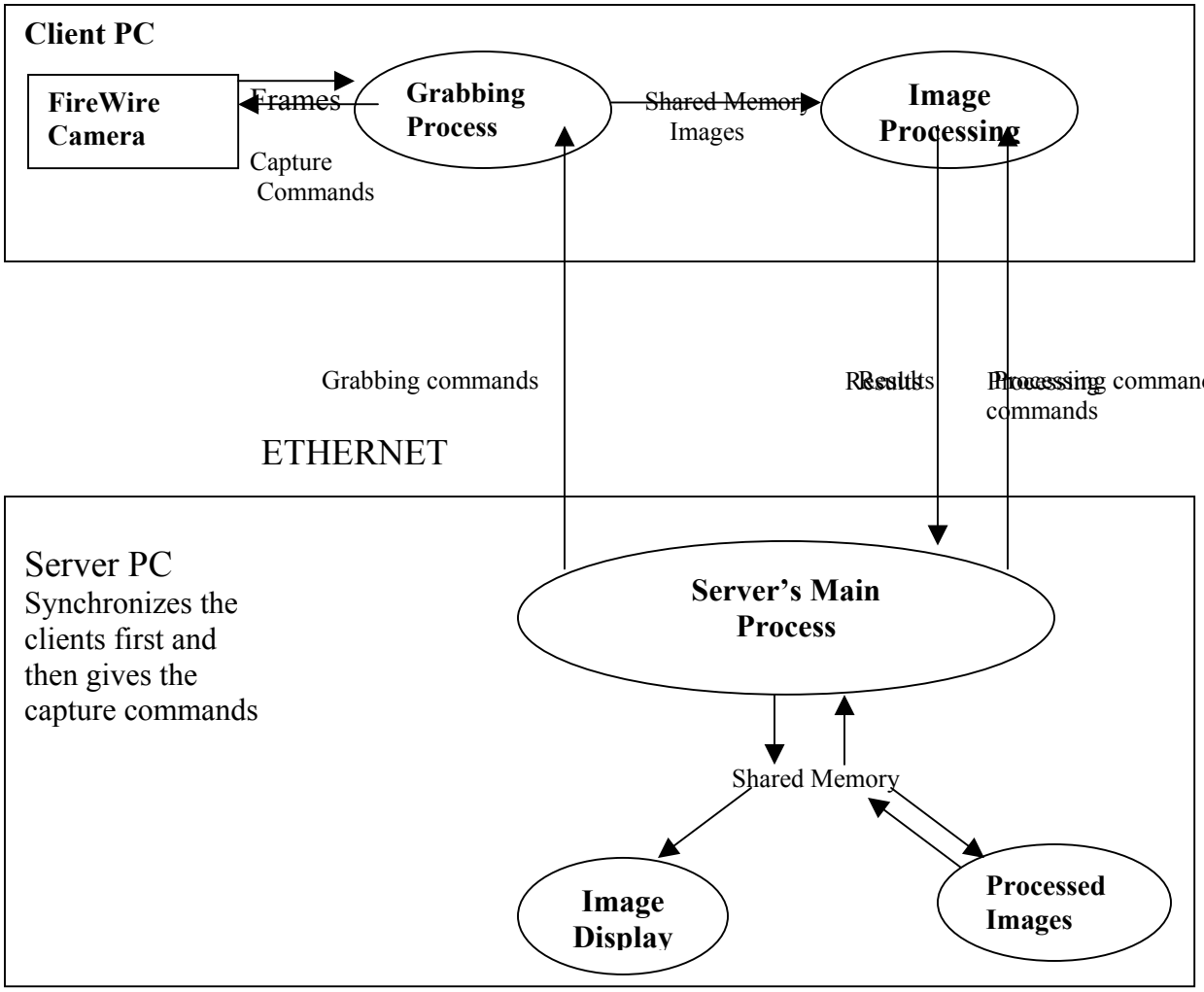
Virtual Galaxy: This is a virtual galaxy designed in OpenGL. The virtual galaxy is projected on the display screen and a person standing inside the cubicle holds a laser pointer which he can move in any direction with a constraint that the pointer is visible in all camera views. Now the 3D coordinates of this pointer are computed using the calibration parameters and the view of the displayed galaxy is changed according to the changing 3D coordinates of the pointer. In this way the motion of moving laser pointer is superimposed to the virtual galaxy. The distributed processing done at the clients takes off the load from the server and hence speed up the computationally intensive processes.

IV. Future work

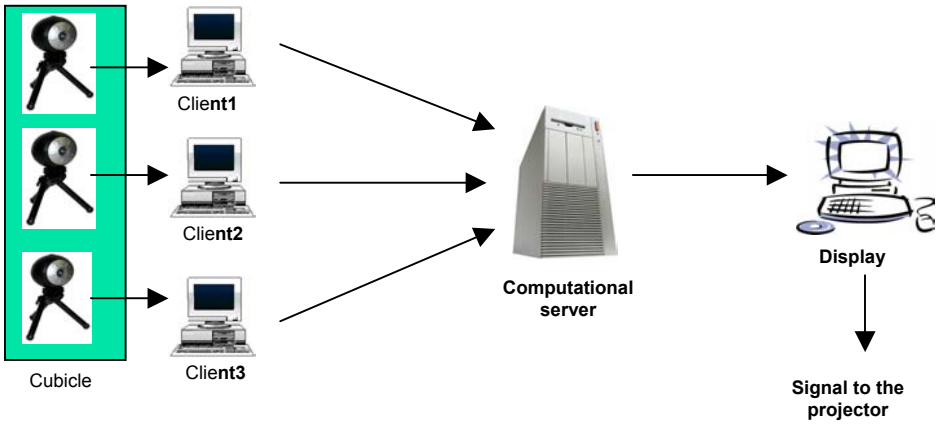
This setup is nothing but our virtual reality workspace in which a person will perform gestures and we will track them in 3-D coordinates using the views from the mounted cameras. The images from the cameras will be used to estimate the global coordinates of the human inside using stereo calibration.

In the Virtual Chess playing utility, a standard chess board will be projected on the white display screen and a skilled player standing inside the workspace of the setup will signal towards a chess board square appearing on the screen in front of him. This gesture will activate the actor standing on that particular square and the actor will be moved to a different square depending on the motion of the player's hand.

The architecture of the multicamera setup has been shown in the following diagrams.



Process which issues the grabbing commands to the FireWire cameras and stores the results in a shared memory. Another process does image processing at the client and sends the processed images to the server.



References

1.Tomas Svoboda and Peter Sturm II. A convenient multi-camera self-calibration for virtual environments. In Computer Analysis of Images and Patterns, pages 183-190 **.**

2. Martin Armstrong, Andrew Zisserman, and Richard I. Hartley. Self-calibration from image triplets. In ECCV (1), pages 3-16, 1996.

3. Joshua Gluckman and Shree K. Nayar. Real-time software synchronization.

4. R. Hartley and A. Zisserman: Multiple View Geometry in Computer Vision, Cambridge University Press, UK, 2000

5. Ivana Mikic, Koshika Huang and Mohan Trivedi: Activity Monitoring and Summarization for an Intelligent Room, IEEE workshop for Human Motion, Pages-107-112,December 2000

6. Mei Han and Takeo Kanade: Creating 3D models with uncalibrated cameras: Proceeding of IEEE (WACV 2000), December 2000.

7. Mark Pollefeys, Reinhard Koch, and Luc Van Gool. Self-calibration and metric reconstruction inspite of varying and unknown intrinsic camera parameters. International Journal of Computer Vision, 32(1):7-25, August 1999.

8. Anurag Mittal and Larry S. Davis. M2tracker: A multi-view approach to segmenting and tracking people in a cluttered scene using region-based stereo. In A. Heyden, G. Sparr, M. Nielsen, and P. Johansen, editors. The seventh European Conference on Computer Vision, ECCV2002, volume 1 of LNCS, pages 18-36).

9. S. Veigl, A. Kaltenbach, F. Ledermann, G. Reitmayr, and D. Schmalstieg. Use of FireWire cameras in augmented reality, 2002.

10. X.-F. Zhang, An Luo, Wenjing Tao, and Hans Burkhardt. Camera calibration based on 3dpoint- grid. In ICIAP (1), pages 636-643, 1997.

11. Q. Luong and O. Faugeras. Self-calibration of a moving camera from point correspondences and fundamental matrices, 1997.