Appendix A

Video Surveillance System

Presently this chapter contains the course project, formulated in general and system independent terms. A project assignment typically involves several persons that do analysis, design, implementation, testing, and documentation over a period of several days. Four persons spending one to two weeks (or more, depending on earlier experiences) is normal.

A.1 Introduction

Low-cost video cameras are today widely used in consumer electronics and for web-based interaction. Another common use, using cameras with a somewhat higher quality but still reasonably priced, is surveillance of public or restricted areas.

Before the evolution of Internet and Intranet solutions, such systems typically consisted of a number of distributed cameras connected to some type of control room as depicted in Figure A.1. Neither computers nor networks were parts of these systems. Such a system does not scale up very well; cabling gets expensive and captured pictures cannot easily be handled for specific analysis and documentation.

Then with the widespread availability of computer networks, the natural development is to have digital camera nodes with built-in network connection. Hence, the camera acts as a web server for obtaining video frames/pictures via the net. A straightforward implementation of such a camera node is simply achieved by connecting a video camera to a PC equipped with a framegrabber board, as shown in Figure A.2. The project proposed in the sequel can be carried out based on such camera nodes, which also simplifies test and debugging in that the development tools can be used directly on the computer that grabs the video frames.

A device with more dedicated hardware is, however, more appropriate for the majority of applications; the camera unit gets cheaper, more reliable, requires less space, and is less visible. Such web-camera products have also

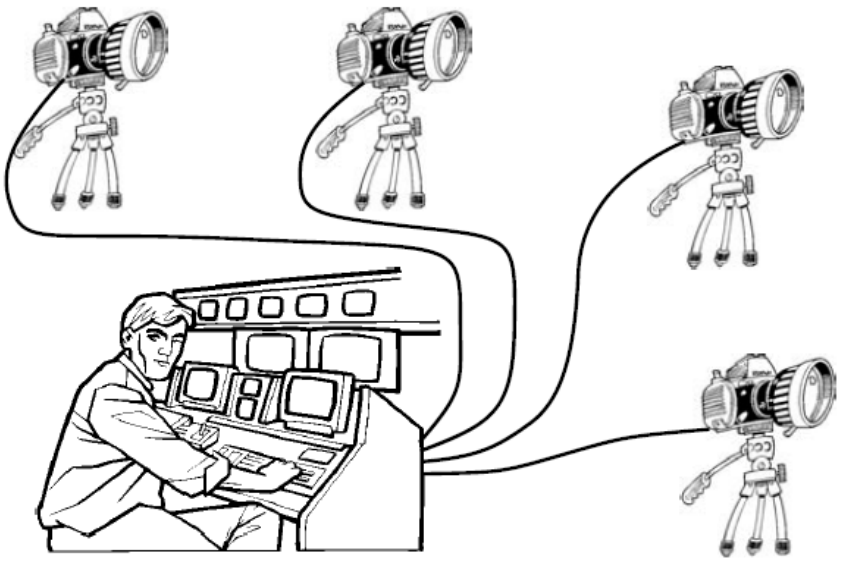

Figure A.1: Distributed cameras connected to an operator for surveillance/-supervision, possibly using pure analog techniques that today is unnecessarily expensive.

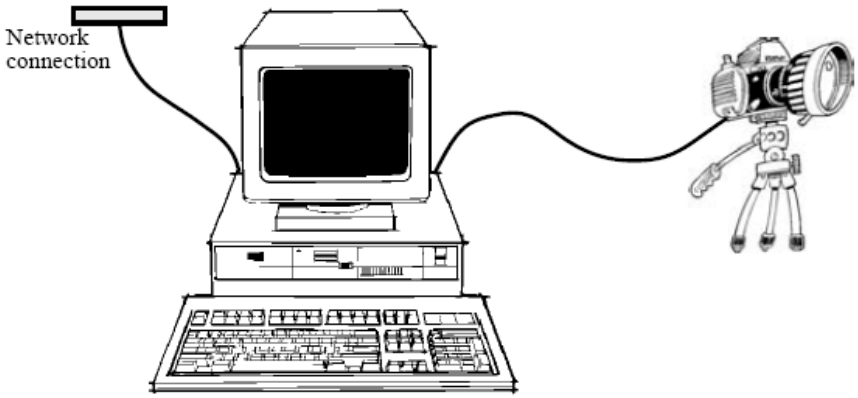

Figure A.2: A camera node consisting of camera, computer, and a connector for the local area network. Software is run on ordinary computers.

been developed, which has resulted in a variety of applications[1]. There are constantly new functions and serviced being developed based on such camera devices, and we see both manual and automatic use cases, for instance in following two directions:

- Applications where humans are set to observe on-line pictures. There is a need to assist the operator in detecting and handling certain situations, possibly occurring seldom but requiring immediate action.
- Applications where the camera information is used by other computers or machines in an automatic manner, either for feedback control of motions or for computerized supervision of automatic operation.

Considering the technical aspects, there are three important development issues: algorithms, hardware, and software. We are concerned with the software issue, which gets especially important when so called intelligent cameras are part of a system including other types of equipment. Then we need to have software that can be easily reconfigured to meet new requirements from adjacent equipment and from operator preferences. There is also a need for open systems, that is, the user of the products must be able to introduce new functionality which may not have been foreseen by the manufacturer of the camera or camera server.

Properties such as portability, superior networking support, and dynamic loading of additional classes and object, make the Java software technology particularly well suited for the software aspects. In the proposed project, we want to gain experience from different software solutions. That is, rather than going for the ultimate high performance system, we want to build a prototype that let us easily change the software and reason about properties of different solutions. Full efficiency is left for those working with optimized products.

From the perspectives of this real-time programming course, we focus on programming techniques that are mainly beneficial for control software and for multimedia software. A supervision system with advanced camera nodes represent the multimedia approach. Still, in terms of demands on timing and distributed computing, our experience should form a good basis for further work on control software, which the project hopefully inspires you to. To conclude, the supervision system we are about to develop is illustrated in Figure A.3.

Specifically, it was found that the Axis 211A (or most other types of Axis models) camera is a suitable platform/device for development of the proposed type of system. It features an embedded Linux-based computing, with embedded Java programming supported by the free LJRT platform.

Although this project was formulated and carried out well before similar products were available, there are now surveillance systems such as the AXIS

[1]See the Network Camera Servers at *http://www.axis.com/products/video/index.htm*

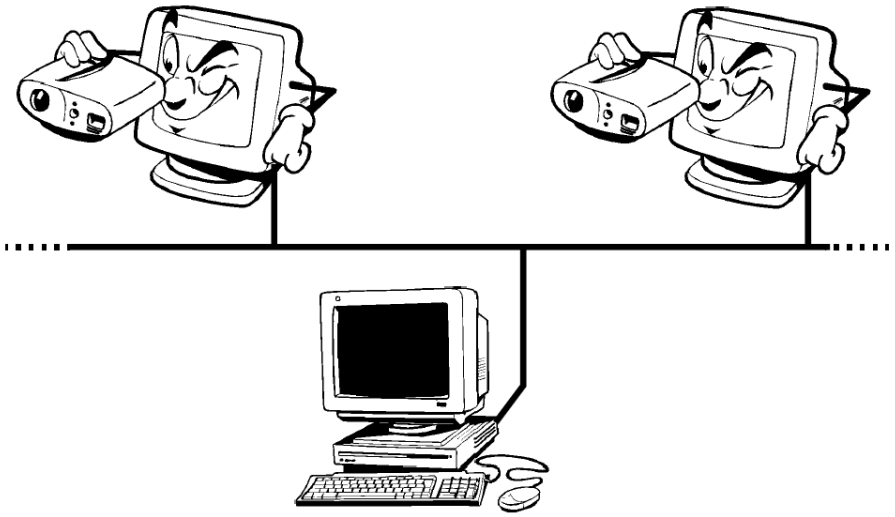

Figure A.3: Computer controlled web-cameras handled from an operator workstation.

Camera Station, which is an IP-Surveillance software that works with Axis network cameras and video servers to provide video monitoring, recording and event management functions. Users can record video continuously, on schedule, on alarm and/or on motion detection. The proposed project can be viewed as creating a portable (Java-based) subset of that type of product, but with the aim of getting experience rather that products or prototypes.

A.2 Requirements

Since just a prototype should be developed, the requirements are more about functionality than performance, even if we want to learn how design influences performance. Specifications:

1. Camera units are referred to as servers since they serve client applications with video images. After being started, a server should permit clients to connect and disconnect during operation.

2. One camera unit only needs to serve one connected client at a time.

3. The client software should be able to concurrently handle two camera units. Optionally, more units may be handled but, of course, with decreased performance.

4. The application should be written in 100% pure Java, except for available hardware interfaces which are accessed via available classes with methods declared as native.

5. Communication is accomplished via TCP/IP or UDP/IP using sockets in Java. Possibly, the UDP support may be missing, and additional support software supporting the communication might be provided.

6. The hardware supports different image/streaming formats but we only use JPEG for the compressed images sent over the network. Supporting classes and guidelines will be provided via the course web.

7. By default, the frame- grabber hardware gives 320 by 240 sized images with 24-bits color depth. We only need to work with that size also in the client software.

8. To obtain higher frame rate also for the case of a slow or loaded network, images are transferred streamed in JPEG format via sockets (see items 5 and 6) which should remain connected until the operator explicitly disconnects. It should be possible to reconnect thereafter.

9. Below the display area of each video picture, there should be an indicator or number showing the current delay (from capture time until received by the client) of the image. If the delay difference between received images are below a certain value which we denote the synchronization threshold, say 0.2 s, the images should be shown synchronized.

   The following implies that when we receive an image from a camera, we might have to delay displaying it for a time up to or equal to the synchronization threshold. When we have one image from each camera matching the synchronization condition, the oldest image should be displayed. Then, after a period of time equal to the time passed between the two images, the second image should be displayed.

   By synchronized display we mean:

   - Images from the same camera are to be displayed by the client in the same temporal order as they were captured at the servers.
   - Two image streams captured by two different cameras shall, if possible, be displayed at the client such that frames captured simultaneously at a certain real time should be displayed at the same time too. The different cameras and frame grabbers at the different servers are, however, not synchronized, and network delays can be different. Therefore, looking at two specific frames from two different cameras, display should be separated in time with as much time as passed between capturing the two images. For instance, if

one image was captured 0.1 second before the second image, the second image should be displayed 0.1 s after the first.

- Captured images come with a time stamp from the device driver. You may assume that clocks on different servers are synchronized, or optionally you can include clock synchronization (at startup) in your communication protocol.

If the difference in image delay between the two cameras is larger than the synchronization threshold, the system shall enter asynchronous mode and display the images as soon as they arrive at the client. The system shall automatically alternate between synchronous and asynchronous mode as the delays change. The user shall be clearly informed of the current mode of operation, synchronous or asynchronous.

It is reasonable to accept a few frames out-of-sync without entering asynchronous mode. The purpose of this requirement is that two cameras that are mounted and displayed side-by-side to visualize a wider scene, should present a wide and consistent view to the operator. Consider the case when a running person passes the two cameras from left to right. On the screen, it should then look the same, possibly with both images delayed but the person should pass through from left to right in the same way.

10. To support the operator, the system should provide two modes: Idle and Movie. In mode Idle, images are transmitted at a low fixed constant rate, say one image per 5 seconds.

    In mode Movie, images should be transmitted at the highest possible rate and with shortest possible delay, which are effected by network and display performance. The hardware supports frame-rates up to the standard (European/PAL) TV frequency of 25 fps (frames per second). Possibly, the hardware (e.g., in case of multiple users or less efficient implementations) might not provide full frame-rate. Rates around 12 fps are typical.

11. The camera server should detect if there is any motion in the captured images. If so, the server should inform the client and the system should enter mode Movie. In the client application, the user should be informed clearly about which mode is active and what camera server triggered the latest movie mode.

12. Motions are detected by comparing pixel color values; there is a motion when the sum of the differences over the pixels between two frames are above a certain threshold. Software (either in C specially for the hardware at hand, or implemented purely in Java, but in any case callable

from your Java software) for detecting motions based on the grabbed images will be provided, to be called directly within your software. Since not all pixels are needed in practice, and fully decompressing images on the server for motion detection would substantially increase the CPU load on the server. Therefore, the class providing motion detection decodes only some of the pixels (in each JPEG square), or special hardware support is used.

13. By default the system starts up in mode Idle for all cameras. When Movie is detected for one camera, all other cameras should also enter mode Movie. This mode remains until the operator decides to enter mode Idle again.

14. In addition to the streamed video, a tiny http server should permit clients to grab an image using a web browser. There will be an available implementation of this feature which should be kept for testing purposes.

15. It should be possible via the client (either as part of the GUI or as debug options) to enforce synchronous mode as well as asynchronous mode. In a similar manner, as when selecting Idle according to item 13, it should be possible to enforce both Idle and Movie mode. For normal operation, there should be a choice Auto.

A.3 Equipment and APIs

There are 10+ Axis 211A Cameras available on the student network @LTH. These are named argus-1 to argus-10 and so on. More specific information such a the placement (subject to change during the course) is given on the course web page. Thanks to the embedded Linux environment and internal design of the device, several users can use one camera at the same time, but that will reduce performance per user so there is currently a limit of five users.

There is also an emulated (or faked) camera available as software, which is to be used for initial development and testing. For instance, to run the Java-based software in the camera, we use compiled Java (via C and the LJRT tools). That provides the efficiency needed, but you cannot (yet) use a debugger during embedded execution The emulated camera is written in pure Java and should work on any platform and also together with debugging tools. The timing properties of the `getJPEG` call is approximately emulated. When developing the code you should use the emulation mode as much as possible and only occasionally test against the real camera.

For networking, the TCP/IP-based communication is more tested than UDP when using the cross-compilation via C, but in any case try to be minimalistic such that the source code gets simple.

The available classes that forms the API you will use, are provided via the course web pages. Updates and enhancements may happen during the course, so stay tuned.

A.4 Work plan

The work is to be carried out in project groups of four persons. To provide guidance, sessions with four or five project groups comprise one session group. Initial testing can be carried out using virtual camera objects, i.e., simulated frame grabbing as mentioned above. The project groups will have to share physical cameras.

Classes used to access the hardware is provided via the web page of the course. The camera classes also provide simulated frame grabbing to permit testing on any machine and without using any camera. Generally, the following plan should be appropriate:

- Analyze the problem including its concurrency and timing properties. When there are mode changes (e.g., Idle to Movie), where is that best detected and how is the corresponding state for a pair of cameras maintained?

- Divide the work into subproblems, one part per person or two parts for two persons. One way is to split the work in one server/network and one client/gui part. Even better but not mandatory: Use XP methodology and start with development of a minimalistic system.

- Specify each part in terms of classes, threads, communication, protocols, etc. Should you use a pull protocol (client requests each image) or a push protocol (server pushes images to the client)? Think about possible network delays and their implications for image synchronization.

- Pieces of source code could be developed during the design phase, but before the full implementation work begins, the tutor should carefully review your design proposal.

- Develop test cases in terms of test stubs and data that support testing of one part at a time. Experiences from test and integration should be included in your report.

- When the subsystems work properly, connect them together and debug the system. Let also other (than the implementers) persons operate the system, and fix the bugs.

Divided into work weeks, the following suggested schedule is good to follow:

Week 1 Introduction meeting. Meet the group, exchange contact info. Start creating a design. Download the project Eclipse workspace. Check out the versioning system (subversion). Study network communication examples. Create a simple homepage.

Week 2 Create design and hand in (pdf). The design should show threads, their responsibilities, how they communicate, how various events in the system are triggered and handled, such as send image, change between movie and idle, etc. Receive feedback on design and ok for implementation from supervisor.

Week 3 Have part of the system running using the fake camera API.

Week 4 Run parts of the system against the real Axis cameras using the camera proxy API.

Week 5 Have full implementation of the system ready. Show the implementation to supervisor. Receive feedback. Prepare documentation of system.

Week 6 Hand in web link to homepage containing project and documentation at beginning of week. Receive web link for another group homepage. Review their work. Present your project at the end of the week.

Week 7 The project experiences are summarized in the lecture.

A.5 Hand-ins

Week 2 - Design

During week 2 you are required to hand-in a design document. The document should be in pdf format and should contain an explanation of your design (UML diagrams are welcome).

The document should show, for both server and client:

- Threads and their responsibilities, and passive objects when you use them
- By what means the threads communicate and what information is exchanged between threads
- Text showing how the threads handle various events in the system (send an image, change between movie and idle, change between synchronous and asynchronous image viewing).
- Text showing how the synchronous and asynchronous viewing modes are handled by the threads (only client).

How to hand-in

The hand-in is done by replying to an email that will be sent to you, with the pdf as attachment.

Week 6 - Homepage

The final hand-in consists of a link to a web homepage. The page should provide content as described below:

Contents

- The names of the group members and a summary (typically 10-30 lines) of your experiences (what were the major difficulties, what should you make different if you started all over again, etc.)
- Links to two pdf documents:
  - A reference manual (typically 6-12 pages) including a users guide for operating the system and a description of the design of the system. Do not forget to make good figures. Pieces of source code can in some cases be the best way to describe a design in the manual, but pictures that graphically depicts your design (e.g. in UML) are usually better.
  - A sales-oriented presentation (typically five slides) showing the benefits and features, but also currently known limitations, of your surveillance system. The potential customers are the members of the other groups, and hence they know about this type of system/product so focus on the solutions (and experiences) that are special for your solution.
- Links to executable code for your client and server, preferably as runnable jar files but can also be zip archives. The reference manual should contain instructions for how to download and run your system using the proxy camera API and also possibly (optional) the Java2C-translated native binary.
- Link to source code for your project. Preferably as zip archives of your Eclipse workspace(s). The reference manual should refer to this code when you describe the design of the system. The source should be possible to build for the proxy camera API. If build instructions are non-trivial they need to be included in the reference manual.

How to hand-in

The hand-in is done by replying to the email that was sent to you for the week 2 hand-in.

A.6 Examination

During the final meeting you will present your project to the group and defend it. You will also review the project of another group and discuss your findings with the group. The supervisor might ask you to correct your project from discovered concurrency problems and require an additional hand-in of the corrected code. Otherwise, if specifications are fulfilled and the documentation/presentation is acceptable, the project is finished.

Code review

After the submission deadline of the web link in the final hand-in, a link to another group will be emailed to you. Your task is to examine the code. During the questions session you will point out the findings from the review. The defending/presenting group will then have a chance to explain/defend their choices. To prepare you should:

- Review the code for any concurrency problems and design issues. Comment on both bad and good choices (in your opinion).
- Execute the solution of the other group. Comment on how easy/not easy it was to get the solution running.
- Read their documentation. Comment on both bad and good reporting (in your opinion).

Presentation

Typically the project sales presentation is delivered by the group, followed by questions asked by the reviewing group, followed by general questions. The whole procedure should last around 10-20 minutes. During the questions session it is good if the defending group is prepared to show the code that is being discussed. The presentation and questions session is moderated by the supervisor. The supervisor also provides a projector for the presentation.