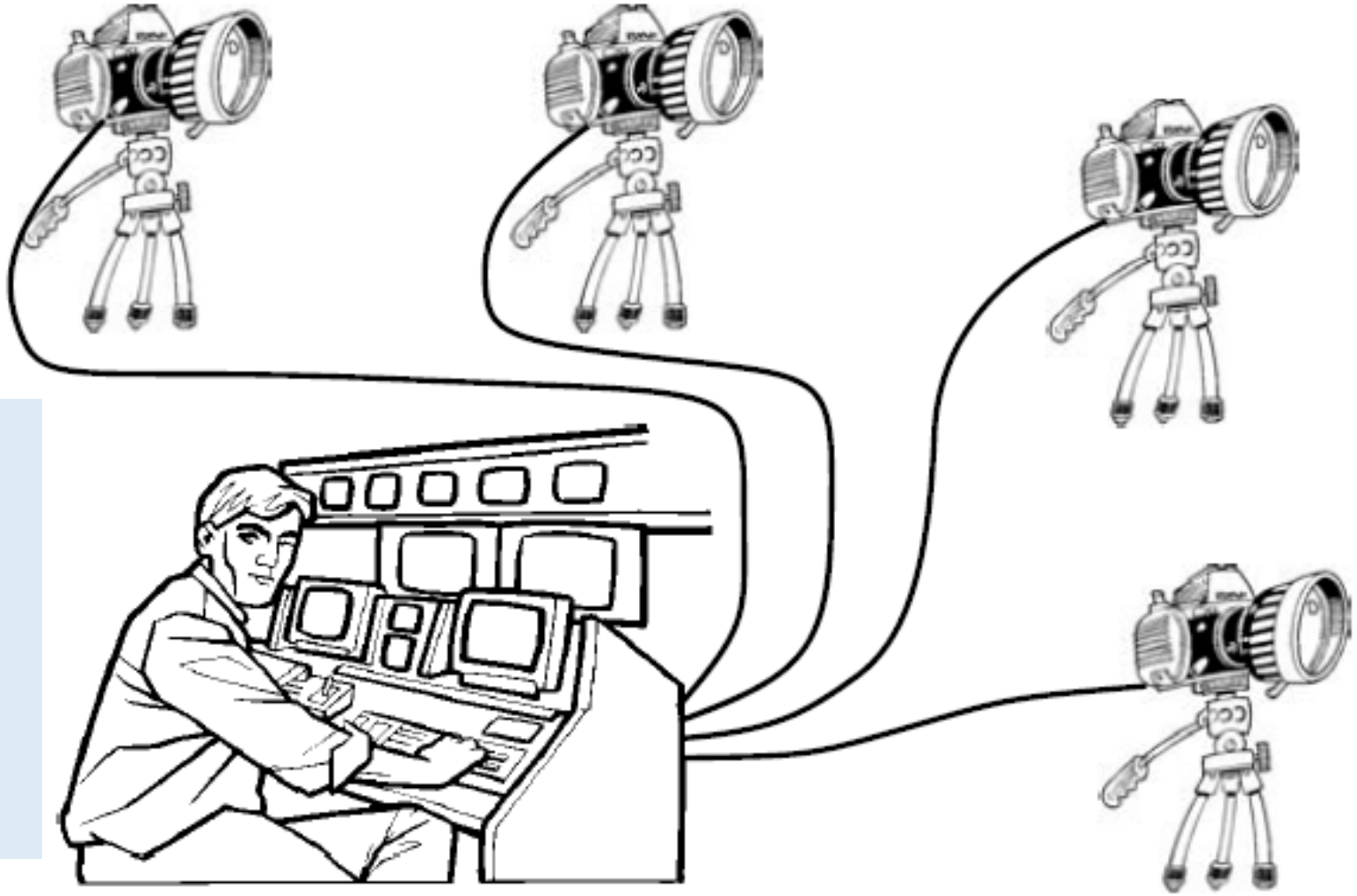
EDA040 camera project

Mathias Haage, 2016

Background

- Digital video capturing; local storage or video output
- Networking and computing embedded in camera, programmable
- Computer power as old PC



Digital network cameras connected to an operator for surveillance and supervision

> Compact, vandal- and dust-resistant design

> HDTV 1080p/3 MP

> Wide viewing angle of 134°

> Digital PTZ and multi-view streaming

> Edge storage

AXIS M3006-V Network Camera

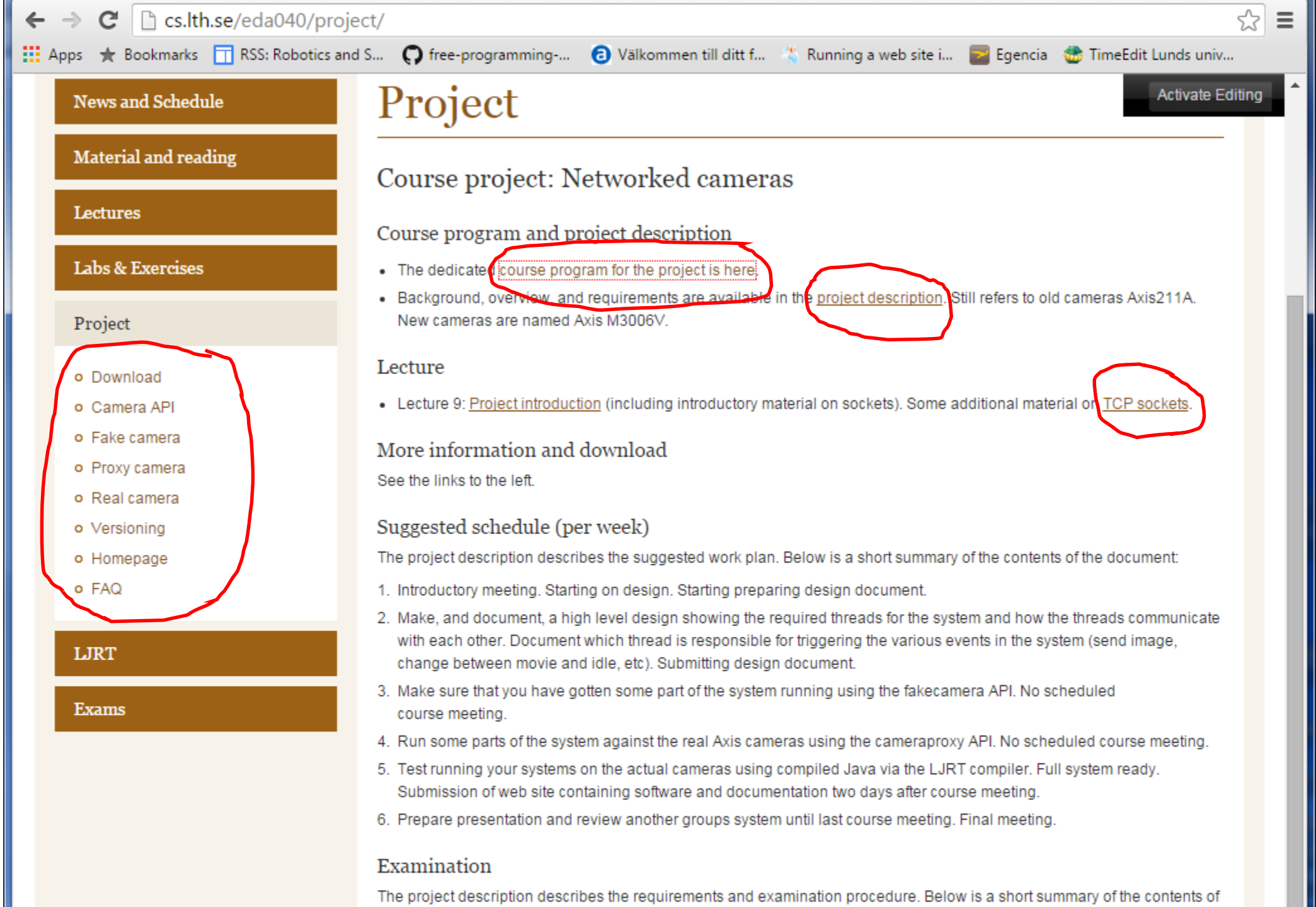
Fixed mini dome with 3-megapixel wide view and HDTV 1080p

Provides streamed JPEG images over ethernet…

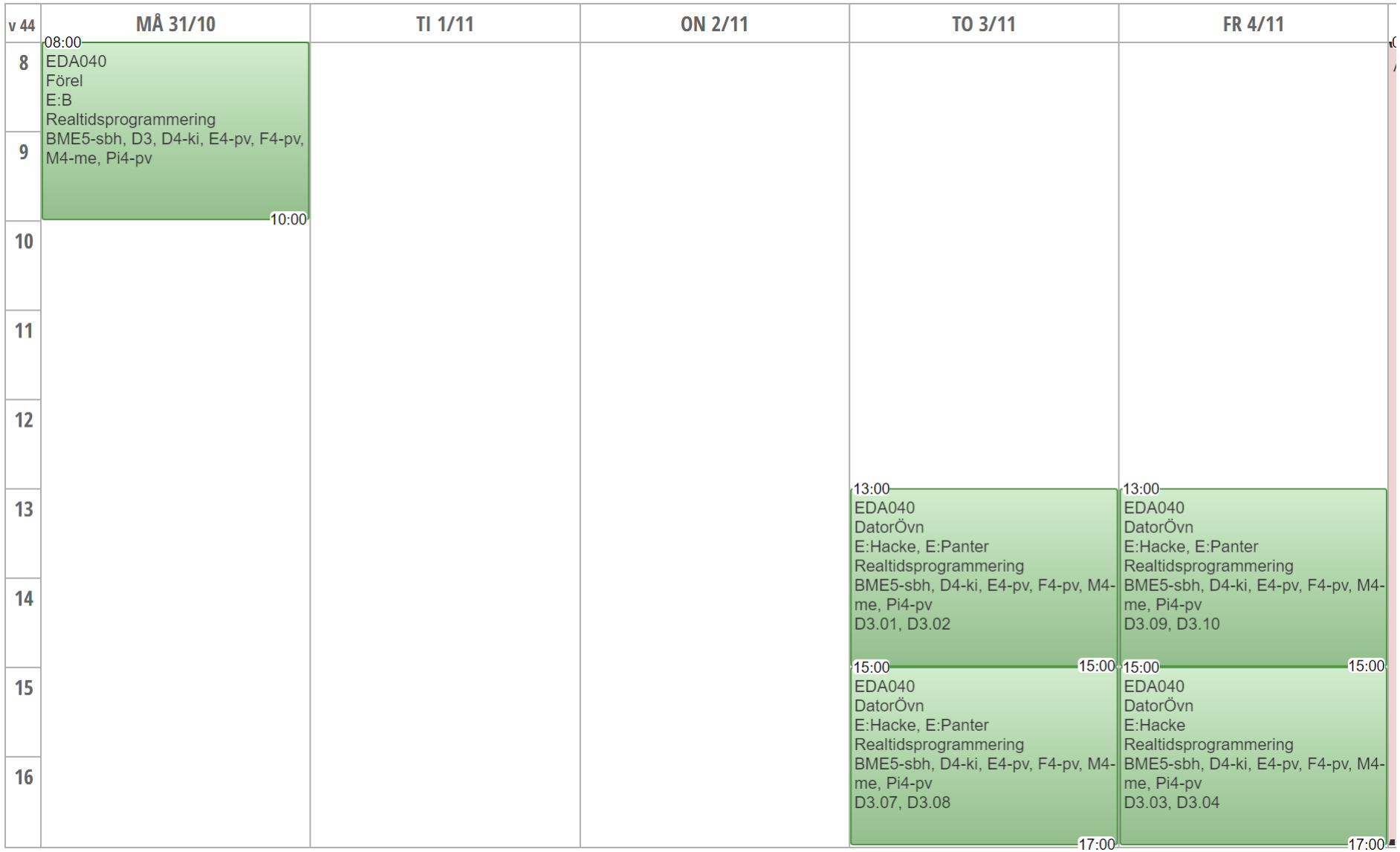
Adress: argus-N.student.lth.se where N is 1…10

Docs

- Project home page
- Important documents
- Software and docs
- Groups
- Peer-review



Schedule


v 44
MÅ 31/10
TI 1/11
ON 2/11
TO 3/11
FR 4/11
8
9
10
11
12
13
14
15
16
08:00
EDA040
Förel
E:B
Realtidsprogrammering
BME5-sbh, D3, D4-ki, E4-pv, F4-pv, M4-me, Pi4-pv
10:00
13:00
EDA040
DatorÖvn
E:Hacke, E:Panter
Realtidsprogrammering
BME5-sbh, D4-ki, E4-pv, F4-pv, M4-me, Pi4-pv
D3.01, D3.02
15:00
13:00
EDA040
DatorÖvn
E:Hacke, E:Panter
Realtidsprogrammering
BME5-sbh, D4-ki, E4-pv, F4-pv, M4-me, Pi4-pv
D3.09, D3.10
15:00
15:00
EDA040
DatorÖvn
E:Hacke, E:Panter
Realtidsprogrammering
BME5-sbh, D4-ki, E4-pv, F4-pv, M4-me, Pi4-pv
D3.07, D3.08
17:00
15:00
EDA040
DatorÖvn
E:Hacke
Realtidsprogrammering
BME5-sbh, D4-ki, E4-pv, F4-pv, M4-me, Pi4-pv
D3.03, D3.04
17:00

Groups

- 4-person teams (really important to keep this number)
- 5 teams per session (= one computer room + one supervisor)

- Sign up during this lecture

- Group A = special group!! (C programming)

Examination

- Hand-ins - during week 2 and 6 you are required to handin some material. You will get an email that you will reply to in order to hand in. The week 2 handin should be a pdf. The week 6 handin should be a web link pointing to a homepage for your project (containing report, presentation, run instruction, runnable code, and source code). A short description of what material the report and presentation should contain is available in the project description document.
- Review - after the final handin you will recieve the link to another groups project homepage. You will run their code and perform a code review looking for realtime problems.
- Final meeting - you will present your project, this will be followed by the reviewing group asking questions on your implementation. After all presentations are done, all groups will show their efforts in a demo session. Any remaining realtime problems (as determined by the teaching assistant) should be fixed and handed in one week later.
- Important handin dates:
  - **Wednesday the 9/11 at 10.00 – design as specified in project description**
  - **Tuesday 6/12** at **10.00 – code and presentation as specified in project description**

Suggested work schedule

- Introductory meeting. Starting on design. Starting preparing design document.
- Make, and document, a high level design showing the required threads for the system and how the threads communicate with each other. Document which thread is responsible for triggering the various events in the system (send image, change between movie and idle, etc). Submitting design document.
- Make sure that you have gotten some part of the system running using the fakecamera API. No scheduled course meeting. Email your supervisor and/or schedule a meeting if needed.
- Run some parts of the system against the real Axis cameras using the cameraproxy API. No scheduled course meeting. Email your supervisor and/or schedule a meeting if needed.
- Test running your systems on the actual cameras using compiled Java via the LJRT compiler. Full system ready. Submission of web site containing software and documentation.
- Prepare presentation and review another groups system until last course meeting. Final meeting.

Suggested development tactics

- First development against **fakecamera** API. Is a software simulation of the camera. Runs everywhere and is great for testing and most development.
- Then test against **proxycamera** API. A software library that connects to the ”real” cameras so that your software does not need to be cross-compiled for the cameras. Great for checking your GUI and network logic.
- Last compile for the camera using the **realcamera** API. Produces an executable binary that need to uploaded to the camera. WARNING: places severe constraints on the Java that can be used (no java.util, no generics, 1.4 compliant only, …)

- A good tactic is to cycle through the above points several times during the developments, adding some functionality each time. Try **NOT** to follow the waterfall development model.

Project

- What should be done?
  - Each group will write a **viewer** and a **camera** software according to specification while adhering to proper concurrency practices.

- Proper concurrency:
  - No race conditions
  - No data corruption (a common mistake is to protect references to an array but not the array itself, i.e. screen flickers…)
  - No deadlocks
  - Multi-threaded design according to the course
  - Thread synchronization according to the course (monitors suggested)

Resources

- Two new resources to be used in the project; **GUI** and **network**

- Look at **example code** on project homepage for how to use each

About this course

News and Schedule

Material and reading

Lectures

Labs & Exercises

Project

- Download
- Camera API
- Fake camera
- Proxy camera
- Real camera
- Versioning
- Homepage
- FAQ

LJRT

FAQ

Swing is not thread-safe

When programming the user interface it is necessary to know that Swing is not thread-safe. Special care is needed to use Swing in multi-threaded applications. Below are a few useful links on the topic:

- General info
- Javamex on invokeLater (look also at invokeAndWait)
- Swing timers

Swing

- Some libraries are not safe to use from multiple threads. Examples are most gui:s, such as Swing.
- One solution is to have a dedicated thread run all calls on the library. Other threads must then communicate their needs to this thread. Swing has such a dedicated thread, called EDT (**Event Dispatch Thread**). It is started automatically when Swing is used.
- This can be done using **delegation**, where Runnable objects are passed between threads.
- Callbacks are often used to respond to events in Swing, such as button presses.
- Since the EDT thread is running Swing, the EDT thread will be calling the callbacks. Any lengthy operations and the GUI will become unresponsive.

Code – SwingUtilities

```
byte[] jpeg = new byte[AxisM3006V.IMAGE_BUFFER_SIZE];
int len = camera.getJPEG(jpeg, 0);
SwingUtilities.invokeAndWait(new Runnable() {
        public void run() {
                gui.refreshImage(jpeg);
        }
});
SwingUtilities.invokeLater(new Runnable() {
        public void run() {
                gui.refreshImage(jpeg);
        }
});
```

About this course

News and Schedule

Material and reading

Lectures

Labs & Exercises

Project

- Download
- Camera API
- Fake camera
- Proxy camera
- Real camera
- Versioning
- Homepage
- FAQ

LJRT

Exams

Fake camera

The fake camera package simulates the behavior of a real camera using a pre-recorded movie as image source. Images are read from a set containing 247 images. Motion detection is triggered in the intervall 86-240. This package should be used for most development to reduce the need for access to physical camera.

It should be no problem using this camera on your home computers.

Sample program using the fake camera:

```
import java.awt.BorderLayout;
import java.awt.Image;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingUtilities;

import se.lth.cs.eda040.fakecamera.AxisM3006V;

@SuppressWarnings("serial")
public class SimpleViewer extends JFrame implements Runnable {
  ImageIcon icon;
  boolean firstCall = true;

  public static void main(String[] args) {
    SimpleViewer viewer = new SimpleViewer();
    (new Thread(viewer)).start();
  }

  public SimpleViewer() {
    super();
    getContentPane().setLayout(new BorderLayout());
    icon = new ImageIcon();
    JLabel label = new JLabel(icon);
```

Proxy camera

The proxy camera consists of two programs. The proxyserver is a program pre-compiled for the Axis M3006V camera that acts as a server for streaming images. The AxisM3006V Java class in the proxycamera package acts as a client to receive images from the proxyserver.

To set up a proxy camera you need to first upload and start the proxyserver program on one of the Axis M3006V cameras:

1. Download proxyserver
2. Copy the program to the camera: scp proxyserver rt@argus-N:~/
3. Login to the camera: ssh rt@argus-N
4. Set execution rights: chmod u+x proxyserver
5. Start the program (on the camera): ./proxyserver XXXX

where N is 1..8 and XXXX is a port number picked by you.

The example Java program below tries to connect to argus-1 on port 5555. It shows captured images and prints detected motion and time difference between capture time and current time to the console:

```
import java.awt.BorderLayout;
import java.awt.Image;
import java.lang.reflect.InvocationTargetException;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

import se.lth.cs.eda040.proxycamera.AxisM3006V;

@SuppressWarnings("serial")
```

News and Schedule

Material and reading

Lectures

Labs & Exercises

Project

- Download
- Camera API
- Fake camera
- Proxy camera
- Real camera
- Versioning
- Homepage
- FAQ

LJRT

Exams

Real camera

For cross-compiling your program to the real Axis M3006V camera you need a cross-compiling tool chain. Such a chain has been prepared on the student computers (it is verified on login.student.lth.se). It compiles your Java program to C through a Java2C translator and then cross-compiles the resulting C code into native code for the camera. This code can then be uploaded and executed.

Instruction for cross-compiling and running the included example

1. Download build_axism3006v.zip. The zip contains:
   - build_axism3006v.sh - build script
   - build_src - source code folder containing:
     - se/lth/cs/eda040/realcamera folder - an implementation of the camera API targeting the embedded camera system
     - http folder - an example program (JPEGHTTPServer)
     - Main.java - a main program starting up the http server
     - Makefile - a file containing configuration data for the compilation
2. Unpack the zip and cross-compile the included example program by running the build script: ./build_axism3006v.sh
3. When the compilation is finished a new folder has been created (build_bin). The compiled executable is located in the build_bin/build folder. It is called Main. If an error has occurred the error log is available in build_bin/build and is called build.err.
4. Upload the executable to a camera: scp Main rt@argus-N:~/
5. Login to the camera: ssh rt@argus-N
6. You may need to set execution permission: chmod u+x Main
7. Run the executable: ./Main
8. Verify that the executable is running by opening a web browser and surfing to http : / / argus-N:6077 (alternatively run the JPEGHTTPClient and connect to host argus-N and port 6077).

Instruction for configuring the contents of the build_src folder to compile your program

1. Copy your source code to the build_src folder. Make sure to keep the package folder structure.
2. Edit the Makefile. Update the MAIN_NAME entry to name your startup class (the one containing the main method)

FAQ

LJRT

Exams

Activate Editing

JPEG HTTP server and client

This sample code illustrates how a very simple web server can be constructed and how network connections can be set up.

It is written for use together with the se.lth.cs.eda040.fakecamera class to supply the images, but can easily be modified for use in an Axis camera. Just change the import such that the se.lth.cs.eda040.realcamera is used instead. The program will listen for connections on port 6077. To test a server running on e.g. login.student.lth.se, direct your web browser to the address "http://login.student.lth.se:6077". You can alternatively use the example image viewer which is also available.

- Simple JPEG HTTP server example using fake camera

This sample code illustrates how the swing graphics library can be used to implement a simple user interface capable of displaying a JPEG image. It also shows how to set up a network connection to a server. The image viewer works together with the simple web server example above to form the client side of a client/server system displaying images sent over a network using the HTTP protocol. Instructions:

1. Start the web server according the instructions for the web server. It will listen for connections on port 6077.
2. Start the client with the command: java JPEGHTTPClient <server address> 6077
3. Press the "Get image" button to refresh the image.

- Simple JPEG HTTP client viewer example

Network code

This sample code illustrates in more detail how networking with a simple protocol supporting a header and varying size data can look like. The code contains three examples (named exampleX).

- Client
- Server

News and Schedule

Material and reading

Lectures

Labs & Exercises

Project

- Download
- Camera API
- Fake camera
- Proxy camera
- Real camera
- Versioning
- Homepage
- FAQ

LJRT

Exams

Camera API

The camera API consists of three packages

- se.lth.cs.eda040.fakecamera
- se.lth.cs.eda040.proxycamera
- se.lth.cs.eda040.realcamera

Each package contain one class, AxisM3006V, which implements the API shown below. To switch between the different implementations, simply change the import statement in your code.

```
public static final int TIME_ARRAY_SIZE = 8;
public static final int IMAGE_BUFFER_SIZE = 128 * 1024;
public static final int IMAGE_WIDTH = 640;
public static final int IMAGE_HEIGHT = 480;

/**
* Initialize resources used by the camera.
*/
public void init();

/**
* Set the location of a proxy camera.
* @param host is host of proxy camera (argus-N)
* @param port is port of proxy camera
*/
public void setProxy(String host, int port);

/**
* Connect to the camera.
*
```

API in short – AxisM3006V class

```
public static final int TIME_ARRAY_SIZE
public static final int IMAGE_BUFFER_SIZE
public static final int IMAGE_WIDTH
public static final int IMAGE_HEIGHT

public void init();
public void setProxy(String host, int port);
public boolean connect();
public int getJPEG(byte[] target, int offset);
public boolean motionDetected();
public void getTime(byte[] target, int offset);
public void close();
public void destroy();
```

Code – simple viewer

```
public class SimpleViewer extends JFrame implements Runnable {
        ImageIcon icon;
        public SimpleViewer() {
                super();
                getContentPane().setLayout(new BorderLayout());
                icon = new ImageIcon();
                JLabel label = new JLabel(icon);
                add(label, BorderLayout.CENTER);
                this.pack();
                this.setSize(640, 480);
                this.setVisible(true);
        }
        public void run() {}
        public void refreshImage(byte[] jpeg) {}
}
```

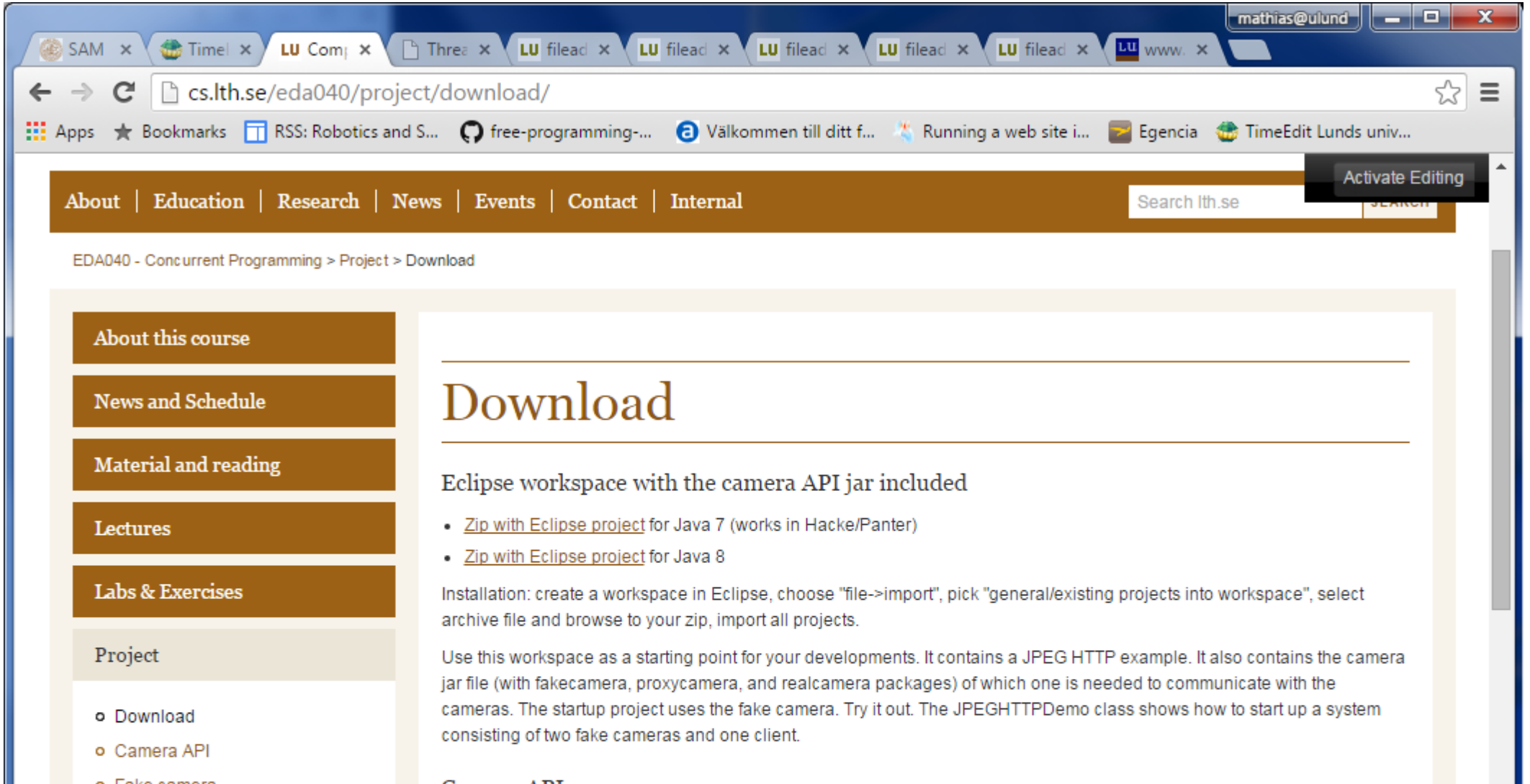
```
public void run() {
        AxisM3006V cam = new AxisM3006V();
        cam.init();
        cam.connect();
        for (int i=0; i<100; i++) {
                byte[] jpeg = new byte[AxisM3006V.IMAGE_BUFFER_SIZE];
                cam.getJPEG(jpeg, 0);
                SwingUtilities.invokeLater(new Runnable() {
                        public void run() {
                                refreshImage(jpeg);
                        } });
        }
        cam.close();
        cam.destroy();
}
```

```
public void refreshImage(byte[] jpeg) {
        Image image = getToolkit().createImage(jpeg);
        getToolkit().prepareImage(image,-1,-1,null);
        icon.setImage(image);
        icon.paintIcon(this, this.getGraphics(), 0, 0);
}
```

getTime – time definition

```
byte[] array;
long stime = System.currentTimeMillis();
int index = 0;
array[index++] = (byte) ((stime & 0xff00000000000000L)>>56);
array[index++] = (byte) ((stime & 0x00ff000000000000L)>>48);
array[index++] = (byte) ((stime & 0x0000ff0000000000L)>>40);
array[index++] = (byte) ((stime & 0x000000ff00000000L)>>32);
array[index++] = (byte) ((stime & 0x00000000ff000000L)>>24);
array[index++] = (byte) ((stime & 0x0000000000ff0000L)>>16);
array[index++] = (byte) ((stime & 0x000000000000ff00L)>>8);
array[index++] = (byte) ((stime & 0x00000000000000ffL));
```

Downloading and starting to program



Working together – GIT in the cloud

BitBucket

- You probably want a repository shared among the group members. Use for instance BitBucket.org to create one. It allows to create free repositories shared with up to five members. **Make sure to create a private repo for the project.**
- GIT commands:
  - Create local repository: git clone
  - Manage local repository: git status, git add, git commit
  - Share local repository: git push
  - Download others shared work: git pull
- Rule of thumb: do not edit the same files!

Studieinformation

Kursanmälan

Studie- och karriärvägledning

Datorsupport

- Datormiljö
  - Windows
  - Linux
  - Backup och borttappade filer
  - Installerade programvaror
  - Personlig hemsida
  - Regler
  - Studentenkät
- Datorsalar och schema
- Konton och lösenord
- E-post
- Utskrifter, skanning mm
- Egen dator, nätverk och programvaror
- Driftinformation och Forum
- Vanliga frågor / FAQ
- Andra supportkällor
- Anställd
- Om Datordriftgruppen DDG

Servicefunktioner

Personlig hemsida

Skapa din hemsida

Alla studenter kan skapa egna hemsidor direkt i sin egen hemkatalog. Det gör man genom att skapa katalogen (mappen) **public_html** i sin hemkatalog. I den katalogen skapar man sedan sin hemsida där startfilen **index.html** blir åtkomligt som:

http://users.student.lth.se/användarnamn/

till exempel http://users.student.lth.se/ay02sw7/

När du läst ovanstående kan du enkelt komma igång genom att kopiera mappen "public_html" ifrån "S:\Support" till din hemkatalog (H:\ ). Öppna filen index.htm med tex notepad eller Dreamweaver och jobba därifrån.

Observera att alla studenter som fått Google-apps mailkonto via StiL kan göra egna hemsidor på sin Google apps-inloggning. Office 365 har också stöd för att skapa egna hemsidor.

Speciellt för E-huset

Ni som har er hemkatalog på unixdatorerna i E-huset kan behöva ställa in rätt filskydd så att webservern kommer åt er hemsida. Webservern har nämligen inga särskilda rättigheter utan kör som en vanlig användare. Normalt ska det bli rätt från början, men har man egna initieringsfiler kan det hända att man behöver justera filskyddet.Det som krävs är att hemkatalogen och public_html är exekverbara av alla, samt att alla html-filer är läsbara av alla. Följande kommandon fixar det:

chmod a+x ~

chmod a+x ~/public_html

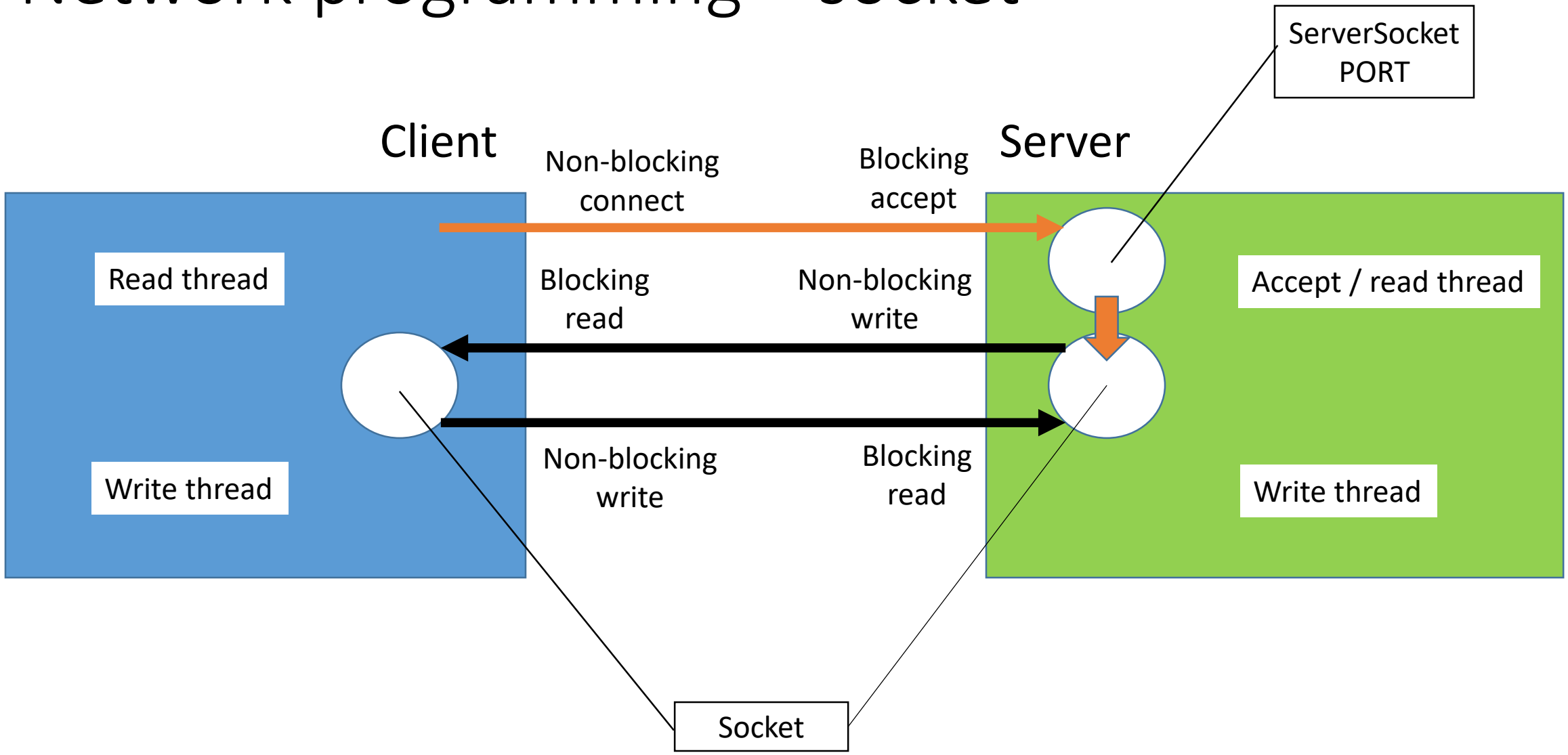
chmod -R a+r ~/public_html/*

Det sista kommandot kan behöva upprepas om man lägger till ytterligare filer.

Använda hemsidan för att dela ut filer

Man kan använda sin personliga hemsida för att föra över filer från sin hemkatalog till sin egen dator hemma eller till vilken nätansluten dator som helst på jorden.

Network programming – socket



Code – create a socket and write (client side)

```
Socket s = new Socket(" argus-7.student.lth.se", 6667);
s.setTcpNoDelay(true);
InputStream is = s.getInputStream();
OutputStream os = s.getOutputStream();
byte[] data = new byte[100];
os.write(data, 0, 100);
s.close();
```

Code – accept a socket and read (server)

```
ServerSocket ss = new ServerSocket(6667);
Socket s = ss.accept();
s.setTcpNoDelay(true);
InputStream is = s.getInputStream();
OutputStream os = s.getOutputStream();
byte[] data = new byte[100];
int read = 0;
while (read < 100) {
        int n = is.read(data, read, 100-read); // Blocking
        if (n == -1) throw new IOException();
        read += n;
}
os.write(1);
s.close();
```

Code – header

```
byte[] buffer; int len;
byte header_hi = (byte)(len / 255);
byte header_lo = (byte)(len % 255);
os.write(header_hi);
os.write(header_lo);
os.write(buffer, 0, len);
byte hi = (byte)is.read(); byte lo = (byte)is.read();
int size = (hi & 0xFF)*255 + (lo & 0xFF);
int read = 0;
while (read != size) {
        int n = is.read(buffer, read, size-read);
        if (n == -1) throw IOException();
        read += n;
}
```

Socket API

Constructors

```
public Socket(String host, int port)
        throws UnknownHostException, IOException;
```

Get-methods

```
public InputStream getInputStream() throws IOException;
public OutputStream getOutputStream() throws IOException;
```

Settings

```
public void setTcpNoDelay(boolean on) throws SocketException;
```

Disconnect

```
public void close() throws IOException;
```

ServerSocket API

Constructors
public ServerSocket(int port) throws IOException;

Wait for connection (blocking)
public Socket accept() throws IOException;

Disconnect server
public void close() throws IOException;

Further

- See network code samples on the project homepage

Specifications, part A: General

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

Specifications, part B: Data content and transport

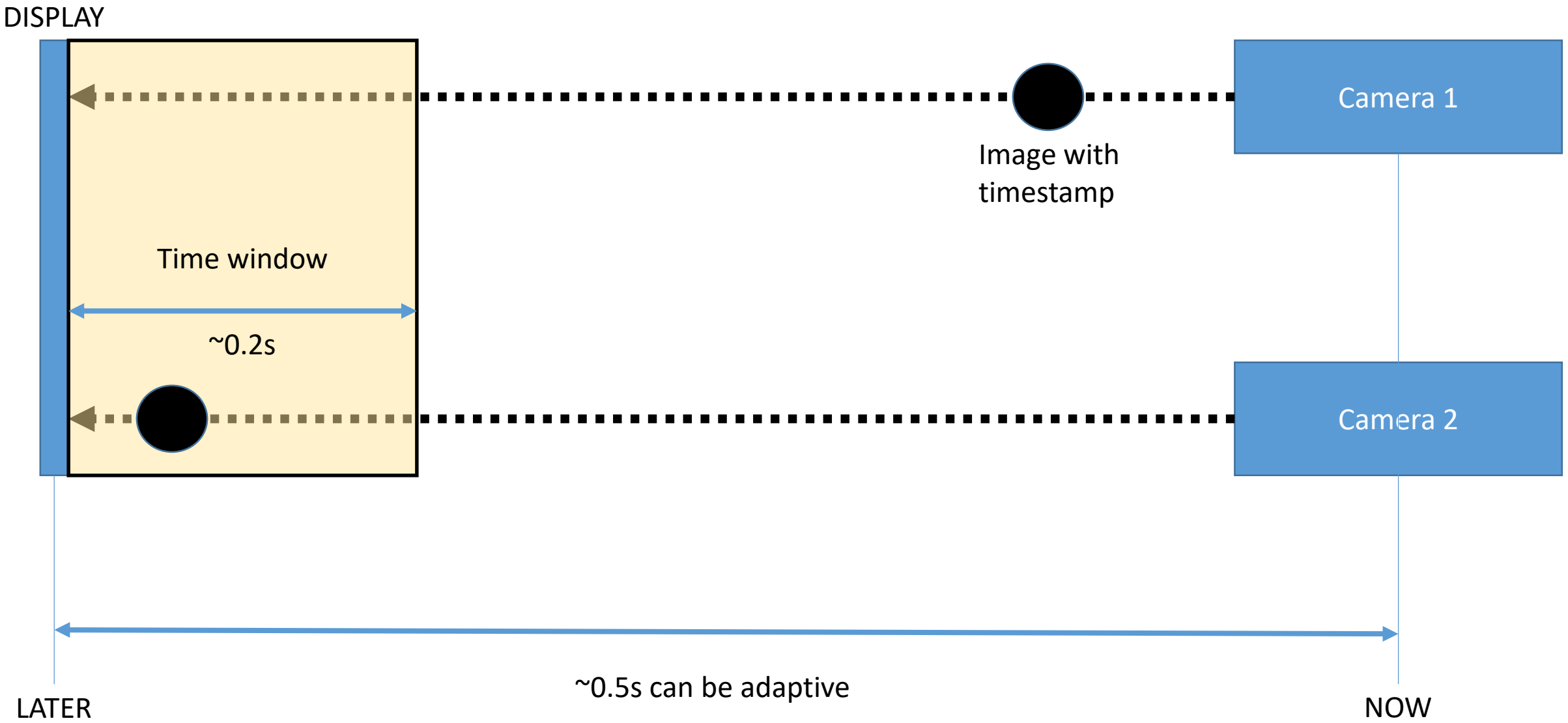
- **REQ 5** Communication is accomplished via TCP/IP, and optionally UDP/IP for the images, using sockets in Java.
- **REQ 6** Use JPEG for the compressed images sent over the network. (Supporting classes and guidelines provided via web pages.)
- **REQ 7** The default image format of the frame-grabber hardware should be supported, which means 640 by 480 sized images with 24-bits color depth. (You only need to support that size also in the client software.)
- **REQ 8** To obtain higher frame rate also for the case of a slow or loaded network, images are to be transferred streamed in JPEG format via sockets that should remain connected until the operator explicitly disconnects. It should be possible to reconnect thereafter.

Specifications, part C: Synchronization

- **REQ 9** Display delay for each video picture, and 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 purpose 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, for instance when a running person passes the two cameras. I.e.:
  - Images from the same camera are to be displayed by the client in the same temporal order as they were captured at the servers.
  - Image streams should be displayed with the same relative delay difference as they where captured.
  - Use image time stamp (from capture time), assuming clocks are synchronized. (Optional clock synch at startup.)

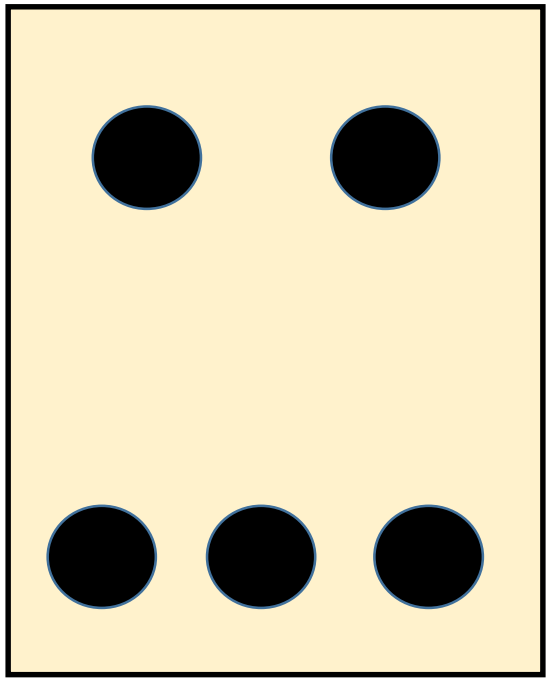
If the difference in image delay between the two cameras is larger than the synchronization threshold, the system shall enter asynchronous mode and display the images as soon as they arrive.

Req 9 – time window + view at fixed later time



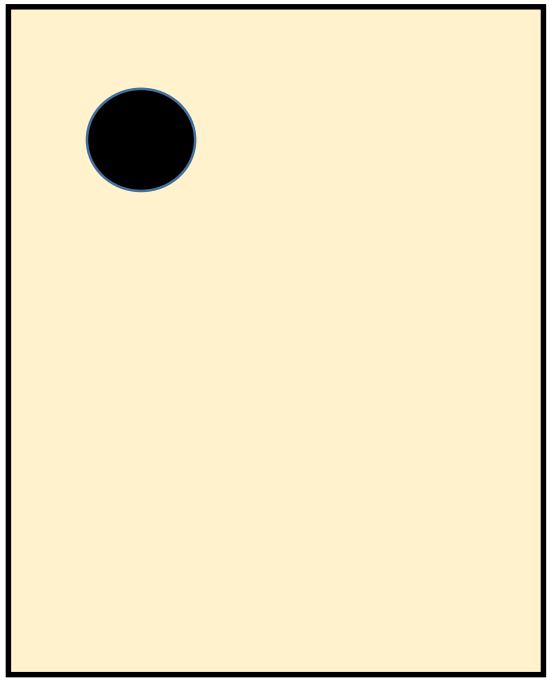
Req 9 – synchronous and asynchronous mode

Synchronous mode



View images in time order,
i.e. show images at fixed time delta after capture

Asynchronous mode



View images as quickly as possible,
disregarding timestamp

Specifications, part D: Motion detection

- **REQ 10** Provide two modes of operation: Idle and Movie:
  - Idle: Images are transmitted at a low xed constant rate, say one image per 5 seconds.
  - Movie: Images should be transmitted at the highest possible rate and with shortest possible delay, depending on network and display performance.

  The hardware supports frame-rates up to the standard (European/PAL) TV frequency of 25 fps (frames per second).
- **REQ 11** In case of any motion in the captured images, the server should inform the client and the system should enter mode Movie, and the user should be informed clearly about mode and triggering camera.
- **REQ 12** Perform motion detection on the server/camera side by using the available classes (possibly utilizing hardware support or decoding only some of the pixels in each JPEG square), which work by comparing sums of pixel color values.

Comment

- Code for motion detection is already available in the camera API

cs.lth.se/eda040/project/camera-api/

Apps Bookmarks RSS: Robotics and S... free-programming-... Välkommen till ditt f... Running a web site i... Egencia TimeEdit Lunds univ...

```
/**
* Connect to the camera.
*
* @return true if connected otherwise false.
*/
public boolean connect();

/**
* Read an image from the camera and put it in the array target starting
* at index offset. The size of target needs to be at least
* offset+IMAGE_BUFFER_SIZE.
*
* @param target reference to byte array to write image into.
* @param offset offset from the start of the byte array.
*
* @return the length of the image captured, 0 if no picture was captured
*/
public int getJPEG(byte[] target, int offset);

/**
* Return true if motion was detected in the latest image.
*/
public boolean motionDetected();

/**
* Copy the capture time of the latest image in the specified target byte array, starting at
* offset. The resolution is milliseconds.
*
* @param target is the byte array to be written into
* @param offset is the array starting position
*/
public void getTime(byte[] target, int offset);

/**
* Close the camera connection.
*/
```

Specifications, part E: Modes and debugging

- **REQ 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.
- **REQ 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.
- **REQ 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.

Design

- Analyze the problem including its concurrency and timing properties.
  - Where will blocking take place?
  - What are the shared resources?
  - What data ows can you identify?
- 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?
- 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)?
  - How should image synchronization work in presence of possible network delays.

Engineering

- 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 and one client part.
- Pieces of source code could be developed during the design phase, but before the full implementation work begins, the teacher should carefully review your design proposal.
- How can your programs be tested locally without networking, but with threads designed to handle the networking?
- 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.

Common concurrency problems

Protecting byte array in monitor. How NOT to do it

```
byte[] buffer; boolean hasImage;
public synchronized void put(byte[] image) {
        while (hasImage) wait();
        buffer = image;
        hasImage = true;
        notifyAll();
}
public synchronized byte[] get() {
        while (!hasImage) wait();
        hasImage = false;
        notifyAll();
        return buffer;
}
```

Protecting byte array in monitor. One way of doing it

```
byte[] buffer; boolean hasImage;
public synchronized void put(byte[] image) {
        while (hasImage) wait();
        System.arraycopy(image, 0, buffer, 0, image.length);
        hasImage = true;
        notifyAll();
}
public synchronized void get(byte[] image) {
while (!hasImage) wait();
        System.arraycopy(buffer, 0, image, 0, image.length);
        hasImage = false;
        notifyAll();
}
```

Common problems

- Opening and closing connection (socket and/or camera) for each captured image. Results in slow performance, and probably the native camera application will crash... Keep connection open.
- Using generics in the server implementation. Will not compile to native application. Also, using data structures such as linked lists, etc., will be problematic for the native application. Keep it simple on the server side and import as little as possible.
- Creating transaction threads in the server to handle each connection. Will not handle well in the native application...
- Image is flickering when viewed. You are not protecting your byte arrays...
- Camera does not switch immediately from idle to movie mode. Check your thread periods...

Common problems

- Using multiple sockets for each connection. Logic for tearing down and setting up the connection will be complicated... Beware!
- Debugging multithreaded programs using the default Eclipse debugger is difficult. A logging class goes a long way towards aiding debugging of your software. Preferably, the debug output can be turned off for the release version...

Finally…

Good luck!!!