Eidgenössische Technische Hochschule
Swiss Federal Institute of Technology
Zürich

Perceptual Computing
Computer Vision

Semester Thesis July 10, 2003

Semi-Automatic Calibration for the Ada/Expo.02 Vision Matrix

Christoph Kiefer
(D-INFK)

Advisor: **Martin Spengler**

Prof. B. Schiele
Institute of Scientific Computing
ETH Zürich

Table of Contents

List of Figures

Chapter 1

Introduction

ETH Zurich participated in the Swiss National exhibition Expo.02[1] (March 15 - October 20 2002) with a project named *Ada*[2].

Ada is named after Lady Ada Lovelace (1815-1852), one of the pioneers of computer science. "Ada - the intelligent space" is conceived as an artificial organism that can interact and communicate with its visitors (Figure 1.1)[1].

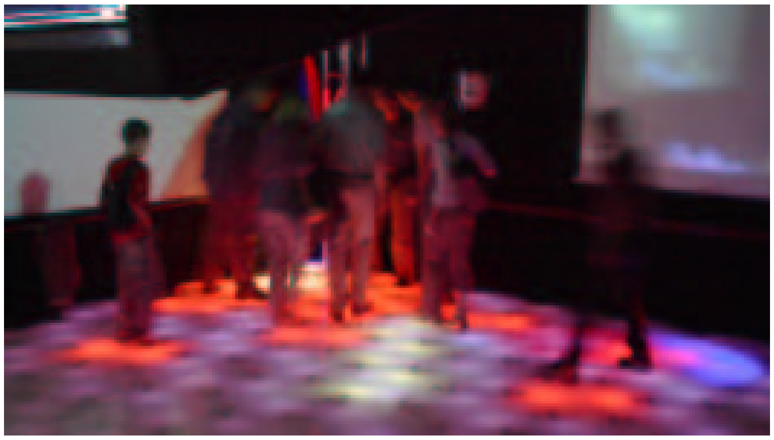


Figure 1.1: Ada - The Intelligent Space.

Different kinds of sensors are installed as Ada's sensory organs. They detect visitors' motion and generate input data. The sensory organs include an active sensory floor, microphones as well as four *pan/tilt* cameras at the ceiling. A pan/tilt camera is an electronic camera that can be moved left, right, up or down. Each camera was recording one quarter or section of the floor. The cameras were connected to a *quad split*. This is a special switcher that splits a television screen into four sections. Each section contains the video source of one camera.

The four cameras build up the so-called *Vision Matrix* of Ada. It consists of a grid of the four sections of the Ada floor. Playing-back the recorded DV tapes is actually showing the Vision Matrix during recording time. The Vision Matrix can be seen in Figure 1.2.

The Vision Matrix's main purpose is to watch visitors everywhere in the space. The

[1] www.expo02.ch
[2] www.ada-ausstellung.ch

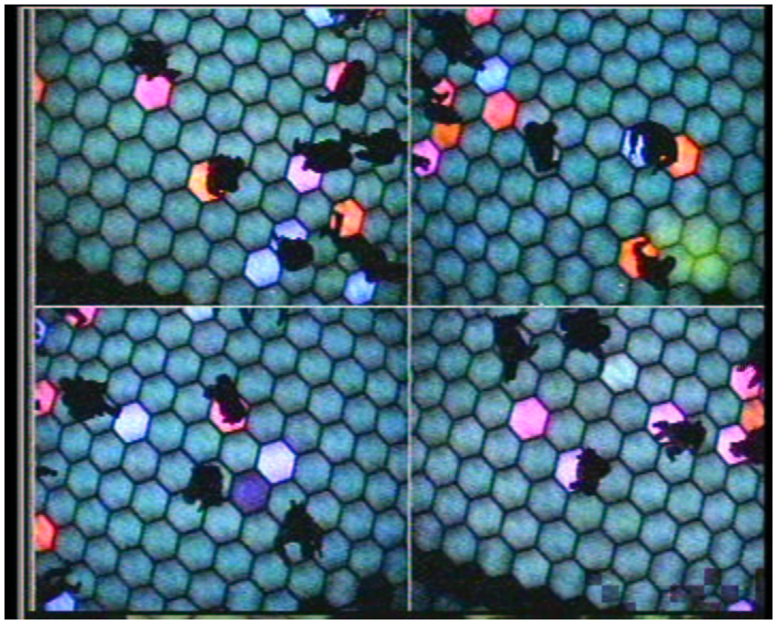


Figure 1.2: Ada Vision Matrix.
The Vision Matrix is a grid that contains the video sources of the four pan/tilt cameras mounted at the ceiling of the space.

video data that is captured from the Vision Matrix will be used for tracking of the visitors. That way, interactive communication between Ada and its visitors is possible.

Furthermore, Ada has a “skin” of 360 hexagonal pressure sensitive floor tiles that can detect the presence of visitors. Visual effects as rings or flowers can be generated using the RGB colored neon lights in each floor tile. Log files were created that store the *state* of the floor in inconstant time steps. That is, for every tile of the floor, the pressure on it and its color is written to the log. These characteristics of the Ada floor make it possible to identify certain points on the floor. These points are used as input to “Tsai’s method” to calibrate the cameras.

The aim of this semester thesis was to develop an application that

- grabs and displays the Vision Matrix from the Mini DV tapes.
- synchronizes the Vision Matrix with the corresponding log file of the Ada floor.
- uses “Tsai’s method” to compute a unified view of the Vision Matrix.

The developed application is called `AdaDVCamera`. This document is divided into five chapters. Chapter 2 describes how to grab the video data from the DV tapes by using *IEEE 1394* aka “FireWire”. Chapter 3 focuses on the synchronization of the video data with the log files. In chapter 4, “Tsai’s method” to compute a unified view of the Vision Matrix is explained. Finally, chapter 5 gives a conclusion and mentions future work. A user tutorial of `AdaDVCamera`, a demo program of `libraw1394` as well as an API documentation of `AdaDVCamera` is given in the appendix. The API documentation is also available as HTML version.

Chapter 2

Digital Video over IEEE 1394

IEEE 1394 is a standard defining a high speed serial bus. This bus is also named "FireWire" by Apple or "i.Link" by Sony. All these names refer to the same thing, but the official standard name "IEEE 1394" by IEEE[13] is used in this document. The DV camera is a "Sony DCR-VX200E PAL". The following sections describe the basic steps to communicate with a DV camera connected via IEEE 1394 to a Linux PC.

2.1 The IEEE 1394 Standard

A brief description of the standard is given in this section. The necessary terms used in this document are explained.

The IEEE 1394 serial bus is similar in principle to USB but runs at speeds of up to 800 Mbit/s. The upcoming IEEE 1394b standard even enables rates from 800 Mbit/s to 3.2GBit/s. The data rate is determined by the slowest active device on the bus. However, the bus can support multiple signalling speeds between individual device pairs. It has a mode of transmission that guarantees bandwith what makes it ideal for DV cameras.

The bus connects all devices via a serial data cable. Connectable devices are also referred to as *nodes*. The standard allows daisychaining of the nodes. Each node in turn may be the source of a new chain what makes it possible to construct tree structures. That way, each node has the ability to act as a bus repeater or "mini hub". There may be no loops in a chain. Each node is assigned a unique ID to be identified on the bus.

The distance between two nodes may be up to 4.5 meters whereas the total length of one chain needs to be less than 72 meters. This allows connecting a maximum number of 17 nodes per chain.

Data packets have a 64-bit address header which is divided into a 10-bit network address, a 6-bit node address and the remaining 48 bits for data memory addresses at the receiving node. This gives IEEE 1394 the ability to address 1024 networks of 63 nodes[1].

The maybe most important feature of the IEEE 1394 bus is to transmit data *asynchronously* as well as *isochronously*. A data packet is transfered asynchronously if the time of transmission does not matter but reliability. Asynchronous transfer provides acknowledged, guaranteed delivery of data and is targeted to a specific node with an explicit address. A control command would be an example therefor. In contrast, isochronous transmission is used if the time of the whole transmission as well as the time between consecutive transfers

[1]The last node is used as a broadcast address that addresses the entire bus.

of packets does matter. In this scenario a packet is discarded if it does not hold the time constraints. Isochronous transmission makes it possible to watch videos uninterrupted in a constant frame rate. 80% of the bus bandwidth is reservered for isochronous transmissions. The remainder is available for asynchronous transmissions.

One node on the bus acts as *bus manager*, an can also act as an *isochronous resource manager* or *IRM*. The latter allocates bus bandwidth for isochronous data transfers when a node requests them. An IRM allocates each isochronous transfer a *channel* consisting of so many bandwidth the node needs. A DV stream to a PC for instance is allocated about 30Mb/s. The number of channels depends on available bandwidth which in turn depends on reserved bandwidth of already existing channels.

The bus is cyclic. Isochronous transfers of all nodes are executed every 125μs. They have always higher priority than asynchronous transfers. 25μs of every bus cycle is reserved for asynchronous control data transfers. Only one data packet can occur every basic cycle for a particular isochronous transfer channel using that channel's allocated bandwidth. There may be multiple isochronous transfers at the same time, providing there is enough bandwidth available. Asynchronous transfers can have multiple data packets per basic cycle within the 25μs.

IEEE 1394 supports "Hot-Plugin" and removal of nodes without shutting down the whole system. A bus reset occurs after adding or removing a device. After a bus reset, nodes detect their neighbors and are assigned a new ID. More information is available in the following documents: [12], [15], [14] and [2].

2.2 Linux IEEE 1394 Subsystem

The core of the entire Linux 1394 subsystem is module `ieee1394`. It manages all *high-* and *low-level* driver modules in the subsystem and handles transactions. The low-level hardware driver modules are below the `ieee1394` module. One such driver for instance is `ohci1394` (1394 Open Host Controller Interface driver). A system having an OHCI compliant card would use this driver module to interface the 1394 card.

Above the `ieee1394` module are the high-level driver modules. One such high-level driver module is `raw1394` that provides an interface for user space applications (executing in user memory space not kernel memory space) to access the IEEE 1394 bus. Applications therefore need to be linked with `libraw1394` (see section 2.3) that handles the communication with the `raw1394` high-level driver module. When the `raw1394` driver module is initialized, it connects to the Linux character device "/dev/raw1394" used by `libraw1394` to connect to `raw1394` from user space[16][17].

2.3 Using `Libraw1394`

"`Libraw1394`[5] is a Linux library that provides direct access to the IEEE 1394 bus through the Linux subsystem's `raw1394` high-level driver module[4]." A key data structure defined in `libraw1394` is the *raw1394handle_t*. It encapsulates a connection to "/dev/raw1394". Every application using `libraw1394` needs such a handle to control one *port*. A port stands for one 1394 card or on board chip (e.g. OHCI compliant chips). To use the handle it must be correctly initialized. Figure 2.1 shows the correct order of function calls to initialize the handle and port.

Function *raw1394_new_handle()* returns a new handle which then needs to be connected to one port. Information about the number of available ports and its connected

```
...

raw1394handle_t handle;
int numcards = 0;
int card = 0;

// Maximum number of ports is 16.
struct raw1394_portinfo pinf[16];
handle = raw1394_new_handle();

if ((numcards = raw1394_get_port_info(handle, pinf, 16)) < 0)
 {
  cout << "couldn’t get card info" << endl;
  return 1;
 }
 else
 {
  cout << numcards << " card(s) found" << endl;
  for(int i=0; i<numcards; i++)
  {
   cout << "nodes on bus: " << pinf[i].nodes
<< " , card name: " << pinf[i].name << endl;
  }
 }

cout << "enter card: ";
cin >> card;

if (raw1394_set_port(handle, card) < 0)
 {
  cout << "couldn’t set port" << endl;
  return 1;
 }

...
```

Figure 2.1: Handle and Port Initialization.
Three functions from `libraw1394` are used to correctly initialize a handle that provides the connection to the IEEE 1394 subsystem: *raw1394_new_handle()*, *raw1394_get_port_info()* and *raw1394_set_port()*.

nodes is obtained by *raw1394_get_port_info()*. The choice of port is then reported by *raw1394_set_port()*.

Having obtained the handle and correctly initialized the port, the next step is to register an *iso handler*. The iso handler is a callback function used by *raw1394_loop_iterate()*. *raw1394_loop_iterate()* repeatedly calls this handler to process the received iso packets from the isochronous packet stream. In the `AdaDVCamera` application, this handler is implemented in class `DVCamera` as a static member function. The appropriate call to register an iso handler is *raw1394_set_iso_handler()*. There are other handlers that can be registered, for instance a *bus reset handler* that is called when a bus reset occurs. What happens on a bus reset is that the configuration of the bus changes. This is the case if a physical device is connected or disconnected from the bus. As mentioned in section 2.1, nodes are then assigned a new unique ID. The kernel and `libraw1394` identify such a configuration of the bus by *generation numbers*. A bus reset handler should call *raw1394_update_generation()* to update the generation number. The default bus reset handler in `libraw1394` will update this number automatically. Correct generation numbers are necessary because packets that are sent asynchronously to a connected device always get tagged with the node ID of the device. In case of the wrong ID, sending of the packed fails. This does not apply to isochronous transmissions since they are broadcast. They do not depend on bus configuration.

Port and handlers are set up at this point. The next step is to start receiving the isochronous stream from the DV camera. This is done by calling *raw1394_start_iso_rcv()* that starts filling up a ring buffer in kernel memory. This buffer needs to be processed by repeated calls to *raw1394_loop_iterate()* as mentioned above. The called iso handler obtains a data pointer to an iso packet in the kernel ring buffer. The iso handler copies the packet's data to an instance of class `Frame`. This is necessary to further process it later. One PAL frame consists of 300 iso data packets. To get one video frame, the iso handler needs to be called 300 times therefore. The processing of these frames is explained in section 2.5.

`AdaDVCamera` application is multi-threaded due to performance reasons. There are two threads implemented. The *main thread* defines the overall program logic, implements GUI elements and processes appropriate events. The *handler thread* is created if a user starts packet receiving, for instance by start playing the DV camera. This thread repeatedly calls *raw1394_loop_iterate()* that in turn calls the installed iso handler. The handler thread is implemented in class `IEEE1394IOHandler`. The code snippet in figure 2.2 shows the procedure of receiving iso packets.

```
...
//In main thread:
handler = new IEEE1394IOHandler();

...
// The main thread calls startPolling() of class IEEE1394IOHandler.
handler->startPolling();

...
// In function startPolling() of IEEE1394IOHandler:
// Start to receive iso packets.
raw1394_start_iso_rcv(_handle, _channel);
iterate = true;

// Start the handler thread.
start();

...
// The handler thread executes the following code:
while (iterate)
{
      // Poll the 1394 interface
      raw1394_loop_iterate(handle);
}

...
// At some point, the main thread sets iterate to false.
// This terminates the handler thread.
iterate = false;

// The last thing the handler thread does is to stop receiving
// iso packets.
raw1394_stop_iso_rcv(handle, channel);
...
```

Figure 2.2: Receiving Iso Packets.
raw1394_iso_rcv() starts receiving iso packets of a certain channel. The handler thread repeatedly calls *raw1394_loop_iterate()* which in turn calls the installed iso handler. The iso handler will copy a packet's data to an instance of class `Frame`. The main thread terminates the handler thread by setting "iterate" to "false". This stops the receiving of iso packets.

It turned out that this procedure alone was not enough to work correctly. *raw1394_loop_iterate()* uses *read()* system call that reads up a certain number of bytes from the handle's file descriptor. But *read()* will block if no data is available. The resulting code snippet is shown in figure 2.3 (only additional code is shown).

```
...
// In class IEEE1394IOHandler:
// A structure containing the handle’s file descriptor is initialized.
struct pollfd pfd[1];

pfd[0].fd = raw1394_get_fd(handle);
pfd[0].events = POLLIN | POLLPRI;
pfd[0].revents = 0;

...
while (iterate)
{
 // Poll the handle’s file descriptor for events.
 if((poll(&pfd[0], 1, -1) == -1))
 {
  cout << "error" << endl;
 }

 if(pfd[0].revents & (POLLIN|POLLPRI))
 {
   // Poll the 1394 interface.
   raw1394_loop_iterate(handle);
 }
}
```

Figure 2.3: Receiving Iso Packets using *poll()*.
A struct `pollfd` from `poll.h` is initialized with the handle’s file descriptor by calling *raw1394_get_fd()*. *raw1394_loop_iterate()* is only called if there is data to fetch from the file descriptor. This way, *read()* will not block.

The code in figure 2.3 makes use of *raw1394_get_fd()* that returns the file descriptor of the handle. This file descriptor can then be used for *poll()* calls. If “POLLIN” (there is data to read) or “POLLPRI” (there is urgent data to read) events occur, *raw1394_loop_iterate()* is guaranteed not to block.

When `AdaDVCamera` application terminates, a call to *raw1394_destroy_handle()* is performed. This closes the connection to “/dev/raw1394” and deallocates everything associated with it. Appendix C shows a simple demo program that summarizes the necessary steps to receive an isochronous data stream from a DV camera.

2.4 Controlling the DV Camera

This section focuses on the key mechanisms to send control commands to the DV camera. A brief overview of the necessary protocols is given.

2.4.1 Data Packet Transfer Protocols

For both isochronous and asynchronous transfers, there are several protocols specified to transfer data packets. The most important protocols for isochronous transfers are described by the “International Engineering Consortium (IEC)”[19] in document “IEC 61883 Digital Interface for Consumer Audio/Video Equipment”. The main components of IEC 61883 are shown in figure 2.4.

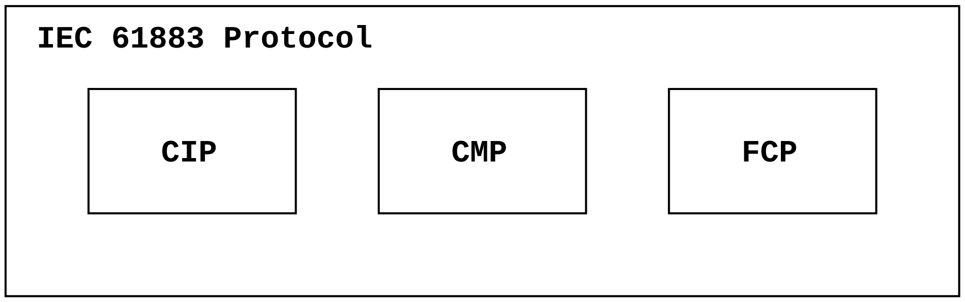


Figure 2.4: IEC 61883 Main Components.
The main components of IEC 61883 are *Common Isochronous Packets (CIP)*, *Connection Management Protocol (CMP)* and *Function Control Protocol (FCP).*

The *Common Isochronous Packets (CIP)* module defines the structure of the isochronous packets that are sent over the IEEE 1394 bus (see also section 2.5.1). The management of isochronous data flow should be done by procedures defined in the *Connection Management Protocol (CMP)*. These procedures specify how to start and stop isochronous packet streams. The *Function Control Protocol (FCP)* is required to send and receive commands for Audio/Video devices. It defines the asynchronous packet structure for high-level Audio/Video device control protocols such as the *Audio/Video Control (AV/C) Protocol*.

Controlling the DV camera is achieved by sending AV/C commands to the camera. The Audio/Video Control protocol defines a command set used to control devices such as video recorders and digital cameras. As mentioned above, it is based on FCP. The FCP module defines two registers: one for commands and one for responses. Figure 2.5 shows the principle of writing the registers.

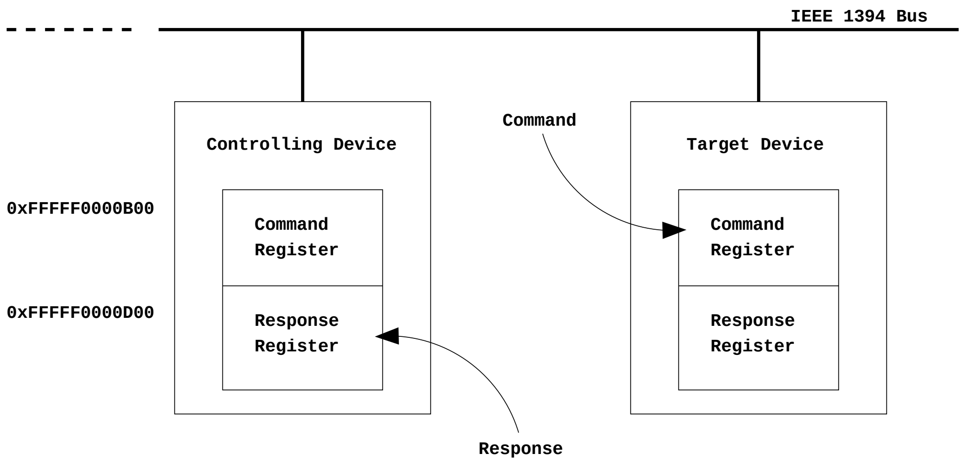


Figure 2.5: FCP Registers at Controlling and Target Device.
The controlling device writes the command to the FCP command register of the target device which in turn writes the response back to the FCP response register of the controlling device. The top addresses of the registers are depicted on the left.

An AV/C command or response is encapsulated in a FCP frame. The controlling device writes the command to a 512-bytes FCP command register of the target device which then writes the response back to the 512-bytes FCP response register of the controlling device[3].

The AV/C protocol is designed to work with digital video cameras, monitors, CD and cassette recorders, as well as amplifiers, tuners and receivers. The commands correspond very closely to the functions of these devices (“play”, “record”, “track time”, etc). The protocol specifies its own command packet structure (figure 2.6).

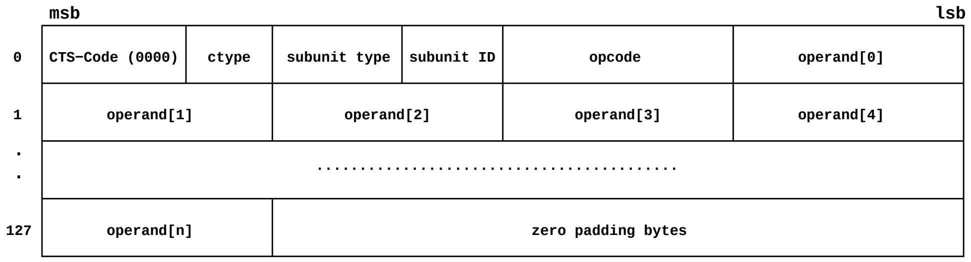


Figure 2.6: AV/C Command Structure.
At the beginning the *Command/Transaction Set Code (CTS-Code)* for Audio/Video (AV) devices can be seen. For this kind of devices, this is always 0. *ctype* defines the type of command, *subunit type* and *subunit ID* address the corresponding subunit. *opcode* refers to a specific command. *operand* specifies the command’s argument. The maximum packet size is 512 bytes (= 128 words on a 32 bit machine).

The *Command/Transaction Set Code (CTS-Code)* is 0 for Audio/Video (AV) devices. *ctype* defines the type of the command, e.g. a control or a status command. The *subunit type* specifies the type of the AV device, for instance a video monitor, tape recorder or video camera. If an AV device has more than one *subunit*, e.g. a tuner besides a recorder, the correct unit can be addressed by the *subunit ID*. The *opcode* defines one concrete command. “Play” or “Wind” are examples therefore. Finally, *operand[0]* to *operand[n]* refer to the command’s arguments, e.g. “Forward” or “Fast Forward”.

2.4.2 Implementation in `AdaDVCamera` Application

To send an AV/C command to the DV Camera, the FCP command register of the camera needs to be written with the appropriate bit pattern of the command. The bit pattern is obtained by ORing individual command pieces (refer to figure 2.6). The following code snippets show bit patterns of command pieces to form command “Play Forward”. They are defined in file `IEEE1394AVC.h`.

The Command/Transaction Set Code (CTS-Code). For AV devices, this is 0.

```
#define AVC_CTS_CODE 0
```

$\boxed{0}$

The type of command (ctype). “Play” is a control command.

```
#define AVC_CTYPE_CONTROL 0
```

$\boxed{0}$

The subunit type. The DV camera is a tape recorder.

```
#define AVC_SUBUNIT_TYPE_TAPE_RECORDER (4 << 19)
```

$$\boxed{\mathbf{1000000000000000000000}}$$

The subunit ID.

```
#define AVC_SUBUNIT_ID_0 0
```

$$\boxed{\mathbf{0}}$$

The command itself (opcode).

```
#define VCR_COMMAND_PLAY 0xC300
```

$$\boxed{\mathbf{1100001100000000}}$$

The command argument (operand[0]).

```
#define VCR_OPERAND_PLAY_FORWARD 0x75
```

$$\boxed{\mathbf{1110101}}$$

These command pieces need to be ORed. For simplicity there is another constant defined: “CTLVCR0”[2].

```
#define CTLVCR0 AVC_CTS_CODE
                | AVC_CTYPE_CONTROL
                | AVC_SUBUNIT_TYPE_TAPE_RECORDER
                | AVC_SUBUNIT_ID_0
```

$$\boxed{\mathbf{1000000000000000000000}}$$

Notice that in this case the CTS-Code and the command type could be omitted. They are both 0. The final “Play Forward” command is obtained by ORing CTLVCR0 with the rest:

```
#define AVC_PLAY_FORWARD CTLVCR0
                         | VCR_COMMNAD_PLAY
                         | VCR_OPERAND_PLAY_FORWARD)
```

$$\boxed{\mathbf{1000001100001101110101}}$$

This command is written to the FCP command register of the DV camera by calling *raw1394_write()*. The next IEEE 1394 bus cycle, the command is executed and the camera starts playing. Figure 2.7 shows the appropriate code fragment.

[2]Control Video Cassette Recorder, Subunit ID 0

```
...
// In class DVCamera:
// Convert command from host byte order to
// internet network byte order.
quadlet_t play = htonl(AVC_PLAY_FORWARD);

// Call sendCommand with correct node address: 1111111111000001
handler->sendCommand(handle, (0xffc0 | 1), FCP_COMMAND_ADDR, 4, &play);

...
// In class IEEE1394IOHandler:
// Write play command to FCP command register of node 1.
void IEEE1394IOHandler::sendCommand(raw1394handle_t handle,
                                    nodeid_t node,
                                    nodeaddr_t addr,
                                    size_t length,
                                    quadlet_t *cmd)
{
 raw1394_write(handle, node, addr, length, cmd);
}

...
// In class DVCamera:
quadlet_t stop = htonl(AVC_WIND_STOP);
handler->sendCommand(handle, (0xffc0 | 1), FCP_COMMAND_ADDR, 4, &stop);
```

Figure 2.7: Writing Commands using *raw1394_write()*.
Class `DVCamera` calls *sendCommand()* of class `IEEE1394IOHandler`. The node is addressed correctly (refer to section 2.1). The handler calls *raw1394_write()* that writes the command to the FCP register of the DV camera. The next IEEE 1394 bus cycle, the command is executed.

2.5 Handling Packet Data

2.5.1 Structure of Isochronous Packets

The handler thread repeatedly calls *raw1394_loop_iterate()* which in turn calls the registered iso handler. The handler is passed a pointer to the kernel ring buffer that contains the received iso packets. To get one video frame from the DV camera, the handler needs to be called 300 times (compare section 2.3). The structure of an iso packet is shown in figure 2.8.

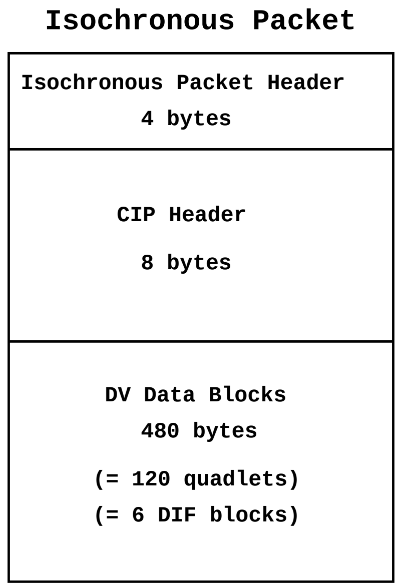


Figure 2.8: Structure of an Iso Packet.
The packet consists of a 4-bytes isochronous packet header. It gives information about the data length of the packet and the channel number that was used. The CIP header contains the source node ID. The data blocks contain audio and video data.

The isochronous packet header gives information about the data length and the used channel number. The header's length is 1 *quadlet* or 4 bytes. The CIP header follows. It contains the ID of the source node, the total data block size in quadlets as well as timestamp information for synchronization of a transmitter and receiver node. The data blocks are organized in *Digital Interface Format (DIF)* sequences. A PAL frame contains 12 DIF sequences. Each DIF sequence is made of 150 DIF blocks at a length of 80 bytes. This gives a total PAL frame size of 144000 bytes. Each DIF block contains a header identifying the *section type* of the DIF block and its position in the data stream. There are five possible section types: *DIF sequence header block, DIF sequence subcode block, DIF sequence VAUX block* and *DIF video block*[21][22]. Figure 2.9 shows the 3-bytes header of a DIF block.

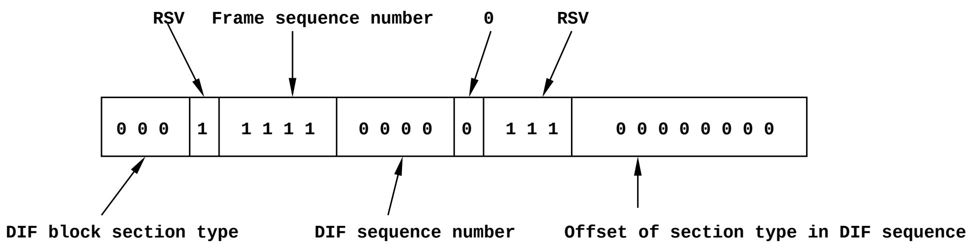


Figure 2.9: DIF Block Header.
The 3-bytes header contains three main values: The section type of the block, the DIF sequence number of the block and the DIF block's offset into the DIF sequence.

The three main values of the DIF header block are the section type of the block, the DIF sequence number and the DIF block's offset into the DIF sequence.

2.5.2 Queue Management

The handler copies the packet data to an instance of class `Frame` dependent on the section type of the DIF block header. A frame object holds a data buffer large enough to store the data of a PAL frame (144000 bytes). There a two data structures implemented in `AdaDVCamera`: the *buffer_queue* and the *output_queue*. At the beginning, all frame objects are on the buffer_queue. The handler thread then calls the iso handler which takes an "empty" frame object from the buffer_queue and copies 480 bytes iso packet data to the data buffer in the frame object. The object is then moved to the output_queue. Frames that are on the output_queue are ready to get displayed. Each frame has therefore a *decoder* (defined in `libdv`[6]). After having displayed a frame, it is appended to the buffer_queue by the main thread to be reused. At all times, all frame objects share one of the two queues, either the buffer_queue or the output_queue. The queues can be seen in Figure 2.10.

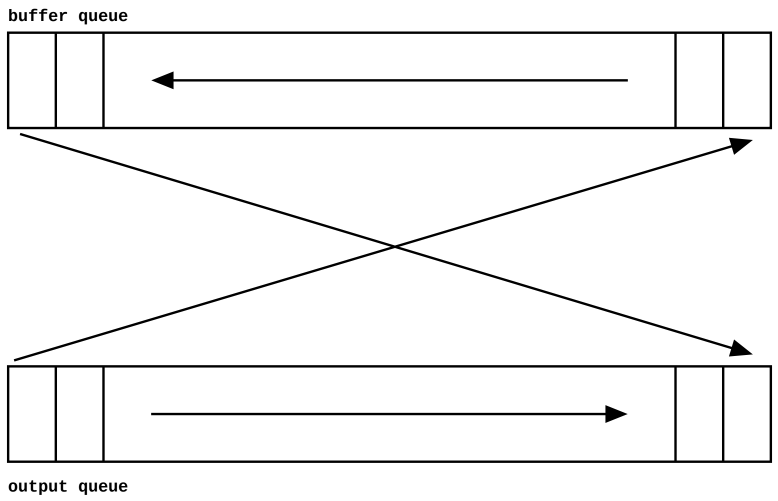


Figure 2.10: Buffer_queue and Output_queue.
The iso handler takes a frame object from the buffer_queue and copies iso packet data to it. Then the handler appends the object to the output_queue. Frames on the output_queue are ready to get displayed. Afterwards, the main thread appends the frame object to the buffer_queue.

Class `Frame` offers methods to process the video frame. The decoder is used for instance to extract RGB values from the frame's data buffer or to extract the timecode of the video frame.

The queues need to be locked if a thread wants to access them. This is necessary due to concurrency reasons. The main thread takes frames from the output_queue and displays them. Thereafter the main thread appends the frame to the buffer_queue. The handler thread does the inverse operation: It takes frames from the buffer_queue and appends them to the output_queue. Figure 2.11 shows basic queue locking mechanisms.

```
Mutex buffer_mutex;
Mutex output_mutex;
Frame *frame;

...
// Lock the output queue, this statement is blocking.
output_mutex.lock();
frame = output_queue.first();
output_mutex.unlock();

// Process the frame, e.g. display it.

// Put the frame back to the buffer queue.
buffer_mutex.lock();
buffer_queue.append(frame);
buffer_mutex.unlock();
...
```

Figure 2.11: Queue Locking.
The queues need to be locked because both, the main thread and the handler thread access these queues. The queues are critical resources. Mutual exclusion must be guaranteed.

Chapter 3

Synchronization

This chapter focuses on the synchronization of a video stream with a log file of the Ada/Expo.02 floor. This synchronization is necessary to select correct pairs of corresponding points of the visualization of the log file and the video. These points will be used as input for the camera calibration algorithm (section 4.1).

3.1 Purpose of Synchronization

The video stream and the log file need to be synchronized for an exact *calibration* of the installed cameras. "Tsai's method"[24][23] is used to compute a *unified view* (refer to chapter 4). Tsai needs pairs of corresponding points of the visualization of the log file and the Vision Matrix. Points selected on the floor are given in world coordinates (millimeters), points selected on the Vision Matrix in image coordinates (pixels). Therefore, to choose correct pairs of points, the video stream and the log must be synchronized.

3.2 Timestamps and Timecodes

Each video frame has a *timecode* in milliseconds. The timecode is the time passed since the cameras started recording.

Example of a timecode:

$$\boxed{\text{00:23:08.106}}$$

Log files of the Ada floor consists of ASCII data. They are arranged line by line. Each line starts with a *timestamp* in milliseconds. The timestamp is the time passed since midnight, first January 1970. Figure 3.1 shows a portion of a log file. Timestamps are emphasized in bold face.

Each line of the log file starts with a timestamp (emphasized in bold face). The following 720 entries of that line are pressure and color of each of the 360 tiles. The pressure on a tile is not used but its color. The colors are encoded as signed integers.

Notice the difference between timecode and timestamp used in this document. The former one designates the time passed since an arbitrarily point in time, whereas the latter one stands for the time passed in milliseconds since midnight, first January 1970. In Timecodes are converted to timestamps to perform calculations in `AdaDVCamera`.

```
1034755208106 62 -16646144 45 -16646144 45 -16646144 64 ...
1034755208166 65 -16646144 45 -16646144 46 -16646144 67 ...
1034755208225 64 -16646144 43 -16646144 51 -16646144 61 ...
1034755208336 66 -16646144 44 -15922164 46 -16646144 62 ...
1034755208394 63 -16646144 42 -9803672 46 -16646144 65 ...
```

Figure 3.1: Portion of Ada/Expo.02 Log File.
The first entry per log line is the timestamp in milliseconds. The next entry state pressure on the first tile. The third entry describes the tile's color at that moment. The total number of tiles is 360. The color of a tile is encoded as signed integer.

Example of a timestamp:

$$\boxed{1034755208106}$$

The corresponding conversion to readable date/time is:

$$\boxed{\text{Wed, 16 Oct 2002 08:00:08.106}}$$

3.3 Synchronization

The video data and the log data are not sampled at the same frequency. The video data is sampled at a constant PAL frequency of 25Hz. The frequency of the log varies between 0.73Hz and 33.33Hz and is 14.51Hz on average. The frequency of sampling the log file depends on the network load at Ada/Expo.02.

The timecode T_{F_0} of the first video frame F_0 is not equal zero. It is something like:

$$\boxed{00:00:00.21}$$

Starting the motors of the cameras takes a few milliseconds. This pre-capture time is written to the tape as the first timecode. Fact is, the time passed at frame F_N since the cameras started to record is the difference of $(T_{F_N} - T_{F_0})$.

To synchronize video frames with the log file, a *reference timestamp* T_{ref} needs to be specified. The reference timestamp says when, since midnight, first January 1970, a video frame was captured. Remember, the video tape only stores timecodes. Possible reference timestamps are shown in the video at the beginning of each tape. Figure 3.2 shows the first frame of a video tape. Timestamps are recognizable in the lower left quarter.

The accuracy of this timestamps is seconds. This leads to another problem, because the precision of the timestamps in the log file is milliseconds. Figure 3.3 shows the situation.

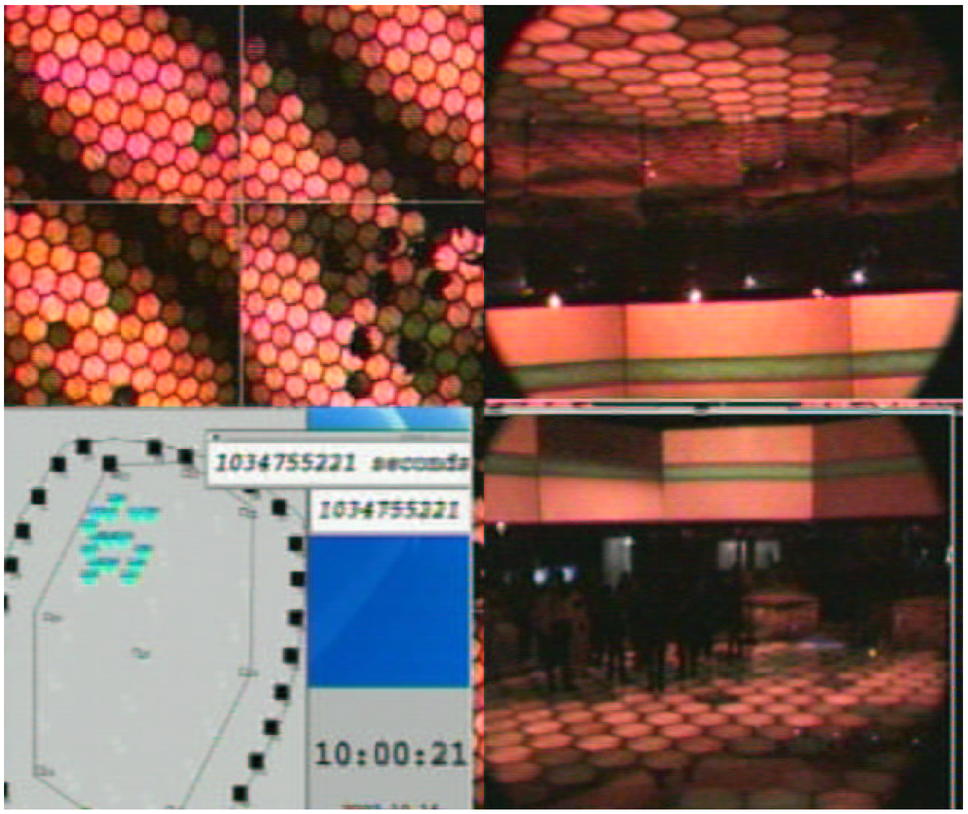


Figure 3.2: First Video Frame with Possible Reference Timestamp.
A possible reference timestamp can be seen in the lower left quarter of the video frame.

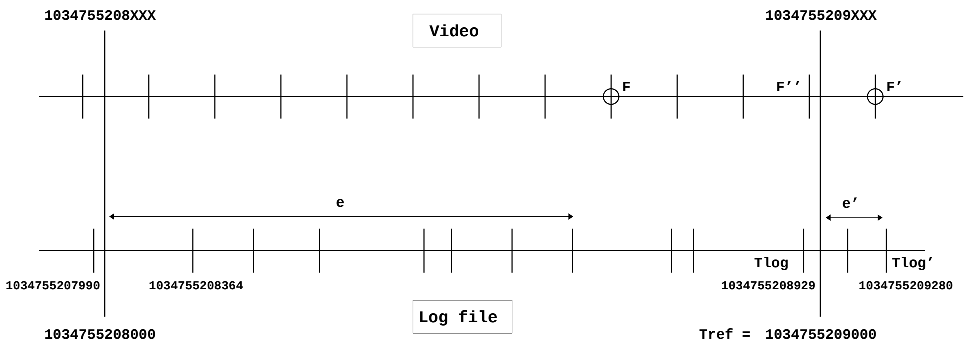


Figure 3.3: Choosing the Reference Timestamp.
Frame F is currently displayed. It shows timestamp 1034755208. The "XXX" states that the accuracy is seconds not milliseconds. Choosing this timestamp as reference timestamp would result in an initial synchronization error of e. Going on until frame F' and choosing 1034755209 as reference timestamp gives a smaller synchronization error e'.

The *initial synchronization* should be as exact as possible. The accuracy of the initial synchronization defines the precision of further synchronization. The goal is therefore to minimize the initial synchronization error. In the example depicted in figure 3.3, frame F shows timestamp 1034755208. The "XXX" states that the accuracy is seconds not milliseconds. F' is the first frame that shows a timestamp that just altered one second (1034755209). This timestamp should be chosen as T_{ref} because the error e' is smaller than e. The timestamp that is closest to the reference timestamp is searched in the log file. This is T_{log} (1034755208929). The log entry of that timestamp is then visualized. If the reference timestamp is 1034755208, T_{log} would be 1034755207990. This results in a bigger initialization error of seven log entries versus two log entries in the example above.

It is obvious from figure 3.3 that the log entry $T_{log'}$ would even better match F'. Or F'' would better match T_{log}. It is therefore reasonable to either move manually one video frame backwards or one log entry forwards. The time between F' and F'' or $T_{log'}$ and T_{log} is said to be the *correction value* C. All this leads to the following equation to synchronize the log file with the video stream. T_{log_N} refers to video frame F_N.

$$T_{log_N} = T_{ref} + (T_{Frame_N} - T_{Frame_0}) + C \tag{3.1}$$

Example: Synchronize Frame F_N with the log file. F_N has timecode:

$$\boxed{\text{00:23:08.106}}$$

The conversion to a timestamp in milliseconds is:

$$\boxed{1388106}$$

The reference timestamp is:

$$\boxed{1034755209000}$$

The timestamp of the first video frame is 21 and the correction value is 180 for instance. This leads to:

$$T_{log_N} = 1034755209000 + (1388106 - 21) + 180 = 1034756597260$$

Using equation 3.1 to compute the initial synchronization is fine. It results in a small synchronization error. But the frequency at which the log data was sampled is not constant. It varies between 0.73Hz and 33.33Hz. This means there will always be an error in further synchronization.

Chapter 4

Camera Calibration and Tsai Method

This chapter will briefly explain the *calibration* of the installed cameras to compute the *unified view*. "Tsai's method" is used therefore[25][24]. Camera calibration is to define *intrinsic* and *extrinsic* camera parameters. Intrinsic parameters are for instance *lens distortion* and *focal length*. Examples for extrinsic parameters are the position and the orientation of the camera. This parameters are used to compute the *unified view*. A unified view is the projection of the four overlapping camera views of the Vision Matrix into a common coordinate system.

4.1 Selection of Points

The first step to calibrate a camera is to select pairs of corresponding points within the view of that camera. A pair consists of a *world point* (point in world coordinates) and an *image point* (point in image coordinates). The world point is selected on the Ada floor, the image point on the video frame. At least seven pairs of points need to be selected per camera. Better results will be obtained if more than seven pairs are selected. Figure 4.1 shows a pair of corresponding corner points in the image and the world coordinates system.

Two files are created next. The *cdfile (camera data file)* holds the image and world points. It is used to generate the *cpfile (camera parameter file)* which contains the camera parameters. The cpfile is created by using the Tsai software[23]. A cdfile and a cpfile can be seen in figures 4.2 and 4.3. "Zw" values in the cdfile are zero since all selected world points lie in a single plane.

The center coordinate $C_n^i = (x^i, y^i)$ for camera view n, $n = 0..3$, are computed in world coordinates $C_n^w = (x^w, y^w)$. These coordinates are computed by calling *image_coord_to_world_coord()* of class `TsaiCal`. Refer to figure 4.4. These world coordinates are used to find out in which camera view an arbitrary world point $P^w = (x^w, y^w)$ of the *Tsai image* lies. The Tsai image is synonym to the unified view. The procedure of selecting points is repeated for every camera view. Each camera gets its own cd- & cpfile. At that point, every camera is calibrated, i.e. they are in a common coordinate system.

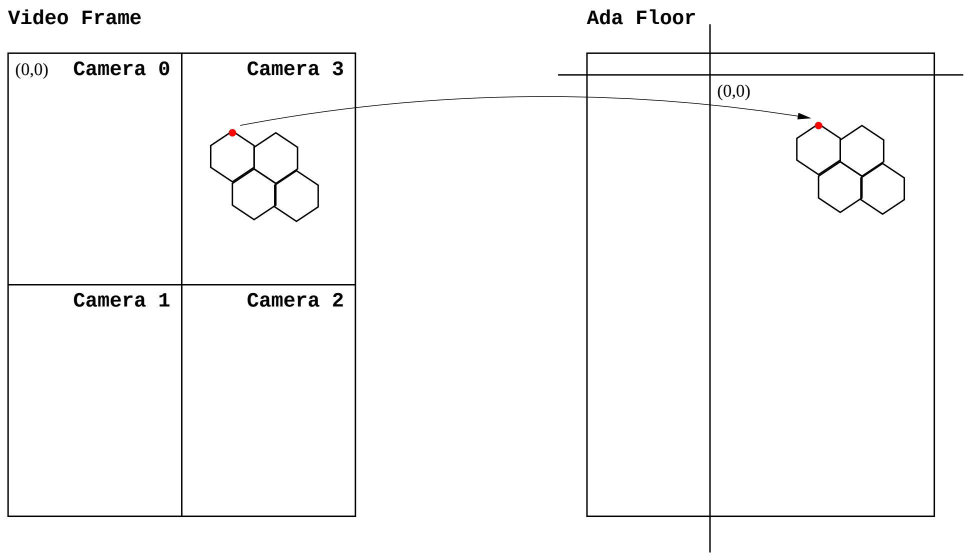


Figure 4.1: Pair of Corresponding Corner Points.
The image coordinate system is depicted on the left side. Corresponding points get selected on the Vision Matrix as well as on the Ada floor (world coordinate system).

```
# a cdfile
#
# Xw Yw   Zw    Xi    Yi
# ---------------------------------
-2310 -7783.33    0      52     323
-1650 -7773.33    0      81     317
 -990 -7773.33    0     115     309
 -330 -7773.33    0     149     302
  330 -7783.33    0     183     294
  990 -7783.33    0     216     283
 1650 -7783.33    0     249     274
```

Figure 4.2: Camera Data File (cdfile).
The cdfile contains coordinates of corresponding world and image points. “Xw”, “Yw” and “Zw” are world coordinates. “Zw” values are zero because all selected world points lie in a plane. World points are given in millimeters. “Xi” and “Yi” are image coordinates. The unit of image points is pixels.

```
# a cpfile
#----------------

# Intrinsic camera parameters.
 5.7600000000e+02 # Ncx: Number of sensor elements in
                    camera’s x direction.
 5.7600000000e+02 # Nfx: Number of pixels in frame
                    grabber’s x direction.
 2.3000000000e-02 # dx: X dimension of camera’s sensor
                    element (in mm).
 2.3000000000e-02 # dy: Y dimension of camera’s sensor
                    element (in mm).
 2.3000000000e-02 # dpx: Effective X dimension of pixel
                    in frame grabber.
 2.3000000000e-02 # dpy: Effective Y dimension of pixel
                    in frame grabber.
 1.3652641152e+02 # Cx: Z axis intercept of camera
                    coordinate system.
 4.9805723022e+02 # Cy: Z axis intercept of camera
                    coordinate system.
 1.0000000000e+00 # sx: Scale factor to compensate for
                    any error in dpx.

 # Calibration constants.
 4.1792644328e+00 # f
-4.9644775779e-02 # kappa1
-3.5136282543e+02 # Tx
-8.0826044350e+02 # Ty
-3.4830068050e+02 # Tz
 7.5465870915e-01 # Rx
 2.7502668339e-01 # Ry
 1.0705791606e-01 # Rz
 0.0000000000e+00 # p1
 0.0000000000e+00 # p2
```

Figure 4.3: Camera Parameter File (cpfile).
The cpfile contains intrinsic parameters as well as calibration constants of the calibrated cameras.

4.2 Computing the Unified View

The unified view is computed by performing the following steps for every point $P^w = (x^w, y^w)$ of the Tsai image. For each camera view n, an instance of class `TsaiCal` is instantiated with the corresponding cpfile of that view.

1. Find out in which camera view the point P^w lies. For this, the distances between point P^w and each of C_n^w, $n = 0..3$, is compared.
2. Take the corresponding class instance of `TsaiCal` and call *world_coord_to_image_coord()* This computes the image coordinates P^i of point P^w.
3. Look up the color of the pixel P^i in the video frame. Fill pixel P^w in the Tsai image with that color.

Figure 4.4 summarizes the whole procedure.

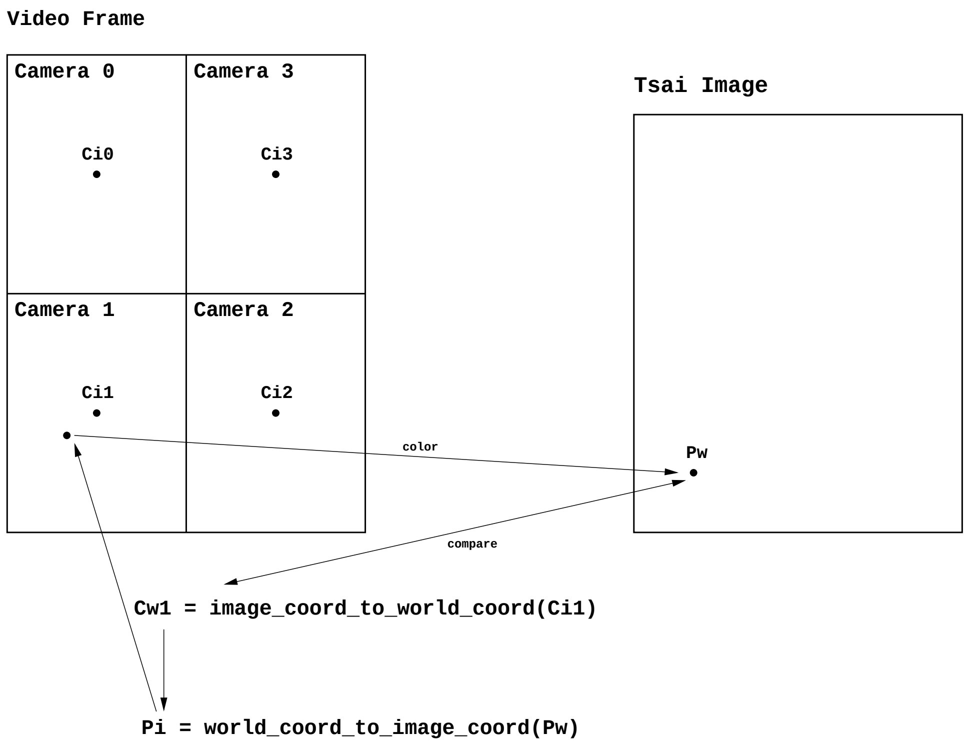


Figure 4.4: Procedure to Compute the Unified View.
The center coordinate $C_n^i = (x^i, y^i)$ for camera view n, $n = 0..3$, are computed in world coordinates $C_n^w = (x^w, y^w)$. Method *image_coord_to_world_coord()* of class `TsaiCal` is used. These world coordinates are used to find out in which camera view an arbitrary world point $P^w = (x^w, y^w)$ of the Tsai image lies. Point P^w is then computed into image coordinates P^i. Method *world_coord_to_image_coord()* is used. The color of pixel P^i in the video frame is written to pixel P^w in the Tsai image.

Chapter 5

Conclusions and Future Work

This document presented an application to read in the Ada Vision Matrix from a DV camera. The camera is connected to a Linux PC via IEEE 1394. The `libraw1394` library is used to control the DV camera. Control commands can be sent to the camera. This commands are specified in the Audio/Video Control (AV/C) protocol. The IEEE 1394 bus and its associated protocols are complex. More functions could be implemented into `AdaDVCamera`.

It was then explained how to synchronize the Vision Matrix with a log file of the Ada floor. This leads to some problems: The log data was sampled with an inconstant frequency. Timecodes are encoded into the video stream but timestamps are stored in the log file. Specifying an accurate reference timestamp is especially critical. The Vision Matrix shows possible reference timestamps whose accuracy is only seconds. The synchronization error can not be eliminated, only minimized.

Tsai's method is used for camera calibration. The cameras are calibrated after a user has selected enough pairs of corresponding image and world points for each camera view. A unified view of the Vision Matrix can be computed then.

5.1 Future Work

Future work should encode the unified view into an MPEG-2 video stream. Besides this, a few GUI elements need to be implemented. For instance a *grid* of the Ada floor that can be put onto the unified view to measure its quality. Or a *magnifying glass* to select points on the Vision Matrix more precisely. In the current implementation of `AdaDVCamera`, log files are parsed step by step. Future work should also be to internalize these log files and eliminate unnecessary values, e.g. pressure on a floor tile. This would speed up the synchronization of the Vision Matrix and the log files.

Appendix A

Installation Notes

`AdaDVCamera` was developed under *Red Hat Linux 7.3*. The latest version of `Qt`[27] was used. That is version 3.2.0 beta 2. The following steps are necessary to compile `AdaDVCamera` on Linux. It is assumed that the CD ROM drive is "/mnt/cdrom". The "<install dir>" is a placeholder for a user specified installation directory.

1. Insert the CD ROM into the CD ROM drive and mount the drive if necessary ("mount cdrom"). Change to the corresponding directory ("cd /mnt/cdrom").

2. Make sure to have `Qt` installed on the system. It needs to be installed with threading support. Install it from the CD ROM if it is not already installed:

   ```
   cd /mnt/cdrom/libs
   tar xzvf qt-x11-free-3.2.0b2.tar.gz -C <install dir>
   cd <install dir>/qt-x11-free-3.2.0b2
   export QTDIR=<install dir>/qt-x11-free-3.2.0b2
   ./configure -thread
   make
   ```

3. Install the DV codec `libdv`:

   ```
   cd /mnt/cdrom/libs
   tar xzvf libdv-0.99.tar.gz -C <install dir>
   cd <install dir>/libdv-0.99
   ./configure
   make
   ```

4. Install `libraw1394` that provides direct access to the IEEE 1394 bus through the Linux subsystem:

   ```
   cd /mnt/cdrom/libs
   tar xzvf libraw1394_0.9.0.tar.gz -C <install dir>
   cd <install dir>/libraw1394_0.9.0
   ./configure
   make
   ```

5. Install `FFMPEG`[30]. It holds the `libavcodec` and `libavformat` libraries that are used to encode the unified view to a video file.

```
cd /mnt/cdrom/libs
tar xzvf ffmpeg-cvs-2003-07-07.tar.gz -C <install dir>
cd <install dir>/ffmpeg-cvs-2003-07-07
patch -p0 < patch.diff
./configure
make
```

6. Install the Tsai calibration code:

```
cd /mnt/cdrom/libs
cp Tsai-method-v3.0b3.tar.Z <install dir>
cd <install dir>
uncompress Tsai-method-v3.0b3.tar.Z
tar xvf Tsai-method-v3.0b3.tar
cd Tsai-method-v3.0b3
make all
```

7. Make changes to the paths section of the Makefile to match the installation directories which were specified during the installation process of all the other libraries.

8. Install and compile the `AdaDVCamera` sources:

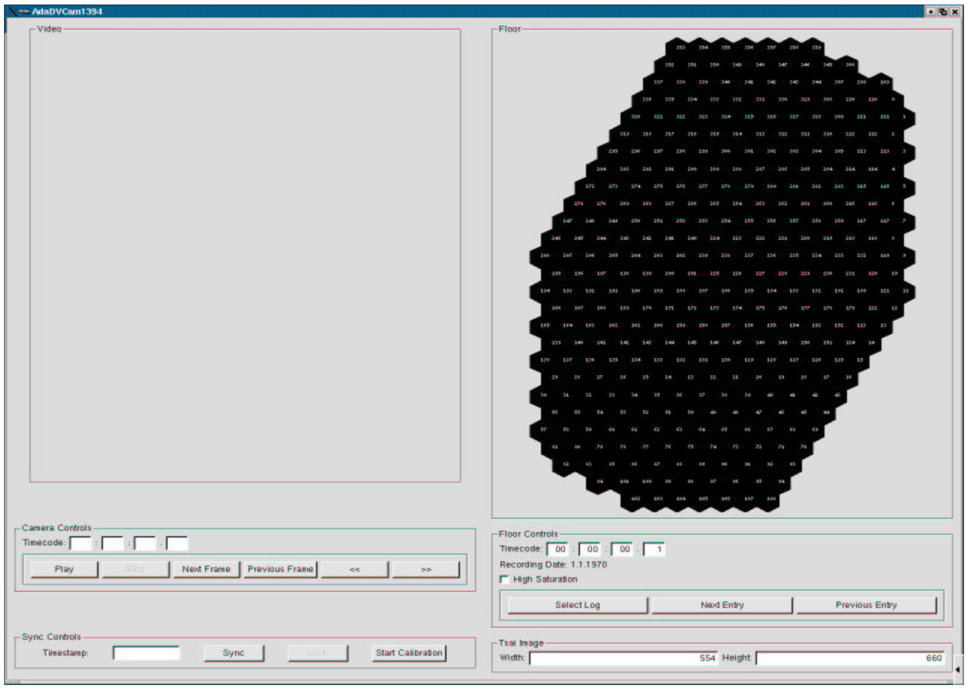
```
cd /mnt/cdrom/src
cp * <install dir>/AdaDVCamera
cd <install dir>/Tsai-method-v3.0b3.tar
cp TsaiCal.* ccal_fo <install dir>/AdaDVCamera
cd <install dir>/AdaDVCamera
make
```

9. In the directory where the `AdaDVCamera` sources are located, type “make doc” to obtain a *Doxygen* documentation. The documentation can be found under the doc directory in both tex and html format.

Appendix B

User Tutorial for `AdaDVCamera`

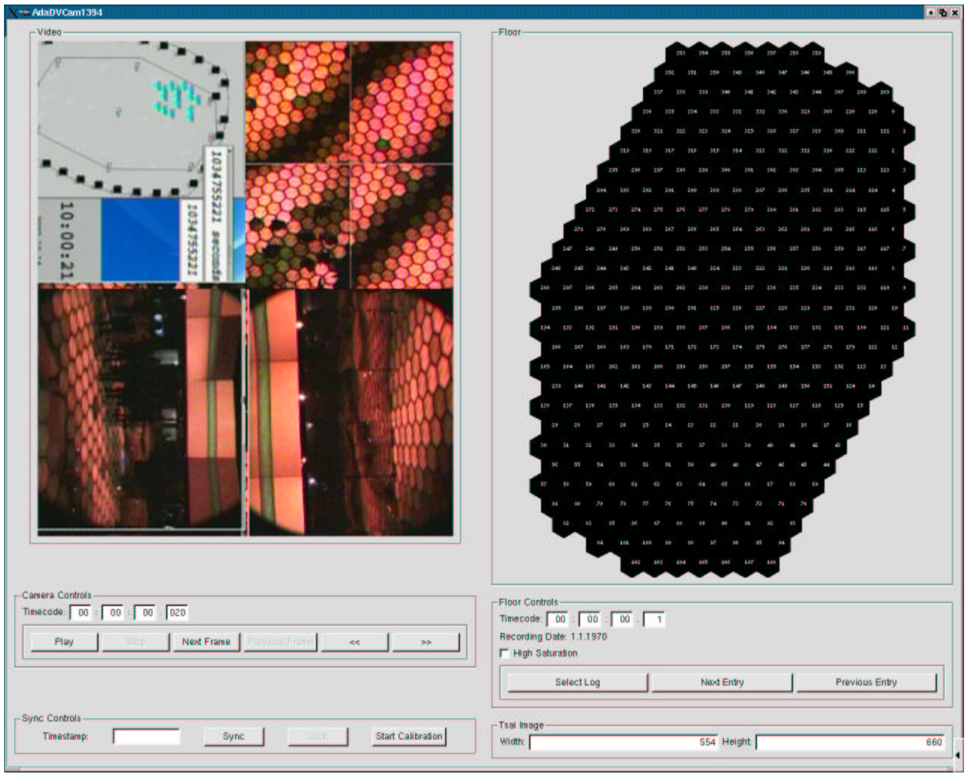
1. Connect the DVCamera to the IEEE 1394 interface and insert a Mini DV tape. Rewind the tape if it is not already rewinded. Start `AdaDVCamera`. Figure B.1 shows `AdaDVCamera` after start up. It is divided into two parts: On the left side the Vision Matrix will be displayed in the *video widget*, on the right side the corresponding log entry in the *floor widget*. The associated controls are discussed later.



Timecode: 00 : 00 : 00 : 1
Recording Date: 1.1.1970
Width: 554 Height: 660

Figure B.1: `AdaDVCamera` after Start Up.
`AdaDVCamera` is divided into two parts: On the left side the Vision Matrix will be displayed in the *video widget*, on the right side the corresponding log entry in the *floor widget*. Several control items help the user to work with the application. These controls are discussed later.

2. Press the “Next Frame” button. The DV camera starts playing but shows only the first frame on the DV tape. The camera will automatically stop after a few seconds. The timecode of the frame is shown inside the “Camera Controls” box. A possible reference timestamp is shown in the top left quarter of the Vision Matrix. The situation is shown in Figure B.2.



Camera Controls — Timecode: 00 : 00 : 00 . 020

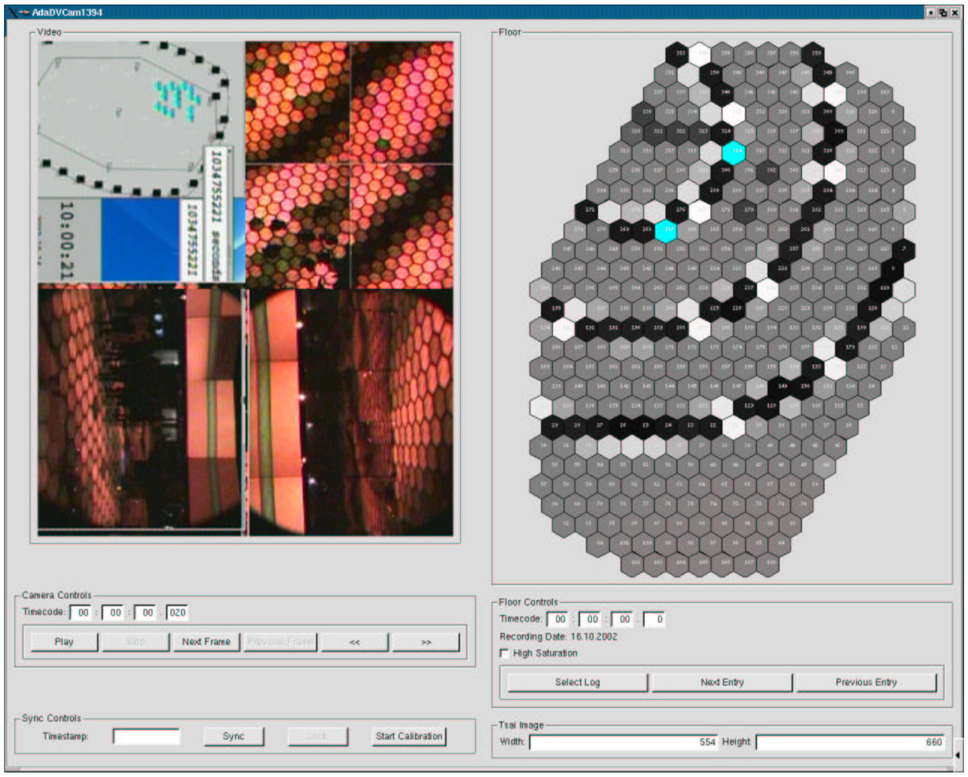
Floor Controls — Timecode: 00 : 00 : 00 . 1

Recording Date: 1.1.1970

Tsai Image — Width: 554 Height: 660

Figure B.2: Display the First Video Frame.
`AdaDVCamera` displays the first Video frame in the video widget. The timecode of the frame can be seen inside the “Camera Controls” box.

3. Load the log file by pressing the “Select Log” button. This will open a dialog window letting the user choose a log file. The first entry of the log file is automatically displayed in the floor widget on the right side. Figure B.3 shows `AdaDVCamera` after having chosen the log file.



Camera Controls Timecode: 00 : 00 : 00 . 020

Floor Controls Timecode: 00 : 00 : 00 . 0

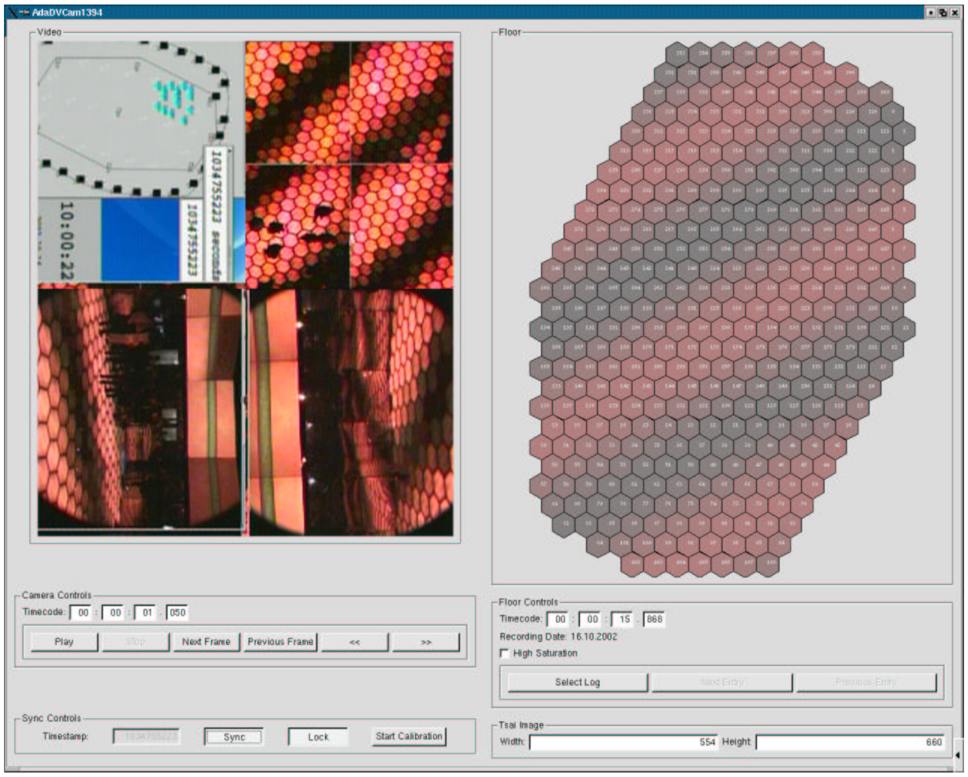
Recording Date: 16.10.2002

Sync Controls Timestamp:

Tsai Image Width: 554 Height: 660

Figure B.3: Select the Log File.
Press the “Select Log” button to choose a log file. The first entry of the log file is displayed automatically in the floor widget on the right side.

4. Set the reference timestamp. Press the “Next Frame” button until a timestamp is shown that just altered one second (compare chapter 3). Enter this timestamp in the appropriate field inside the “Sync Controls” box and press enter. Press the “Sync” button. `AdaDVCamera` displays the log entry that matches best the reference timestamp (figure B.4).



Camera Controls
Timecode: 00 : 00 : 01 . 050

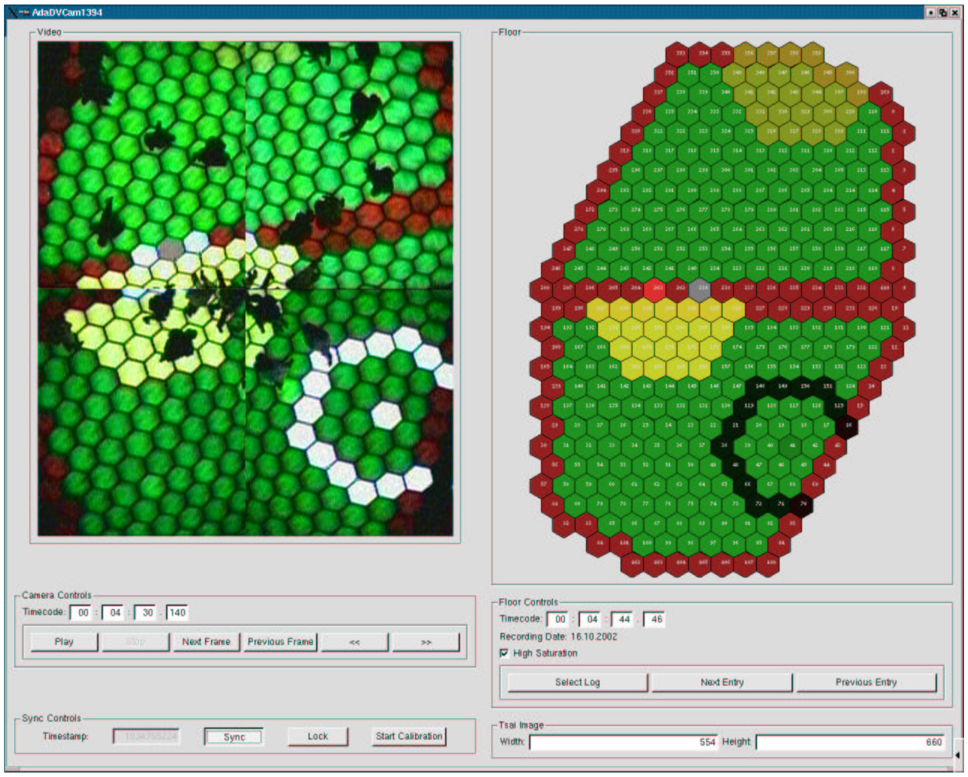
Floor Controls
Timecode: 00 : 00 : 15 . 868
Recording Date: 16.10.2002
☐ High Saturation

Sync Controls
Timestamp: 1034755223

Tsai Image
Width: 554 Height: 660

Figure B.4: Enter the Reference Timestamp and Synchronize.
Enter the chosen reference timestamp into the appropriate field inside the “Sync Controls” box. After pressing the “Sync” button, the corresponding log entry that matches best the reference timestamp is displayed in the floor widget on the right side.

5. Find a video frame that is suitable for camera calibration. Thus a video frame that shows some patterns (e.g. flowers, stars, rings etc.). Pressing the “Play” button starts playing the DV camera. In parallel, corresponding log entries are displayed in the floor widget. Press the “Stop” button when a suitable video frame appears. It is also possible to fast-forward/rewind the tape by pressing “»”/“«”. Press “Stop” to stop fast-forwarding/rewinding or “Play” to restart the camera. The video widget and the floor widget are *locked*. That means, the floor widget will automatically display corresponding log entries when the DV camera is playing. Pressing the “Lock” button will change this behavior. That way, a preciser synchronization of the Vision Matrix and the log file can be achieved by manually pressing the “Next Entry” or the “Previous Entry” button inside the “Floor Controls” box. It is also possible to move one video frame forwards or backwards. Therefore press the “Next Frame” or “Previous Frame” button. Figure B.5 shows a situation that is suitable to start camera calibration. A user may enter the image size of the unified view into the appropriate fields inside the “Tsai Image” box before starting the calibration.



Timecode: 00 : 04 : 30 . 140

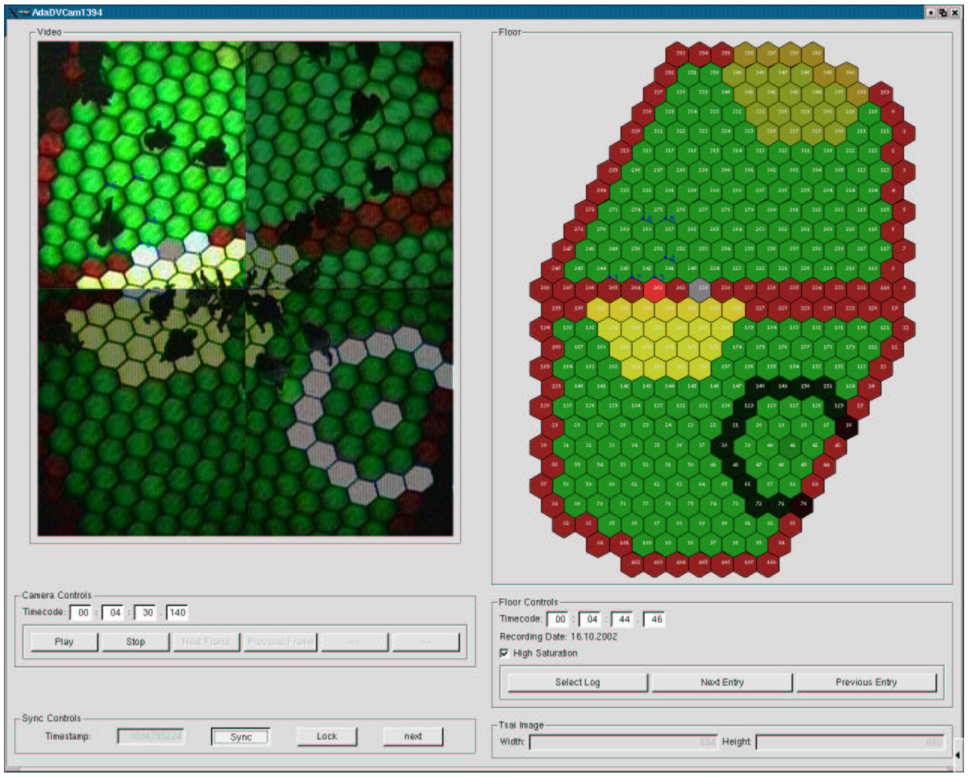
Timecode: 00 : 04 : 44 . 46
Recording Date: 16.10.2002
☑ High Saturation

Timestamp: 1034755224

Width: 554 Height: 660

Figure B.5: Suitable Situation to Start Camera Calibration.
Find a video frame that is suitable for camera calibration. Thus a video frame that shows some patterns (e.g. flowers, stars, rings etc.). Start playing the camera or fast-forward it. The video widget and the floor widget are locked, i.e. playing the camera will automatically update the floor widget with the corresponding log entry of the displayed Vision Matrix. Press the “Lock” button to change this behavior. Move manually to a suitable position by pressing the “Next Frame”/“Previous Frame” button or “Next Entry”/“Previous Entry” button.

6. Start camera calibration by pressing the “Start Calibration” button. This highlights the top left quarter of the Vision Matrix. Choose at least seven points within that quarter. Afterwards select the corresponding points in the floor widget (figure B.6). An equal number of points needs to be selected in both, the Vision Matrix and the floor widget. Having done this, press the “Next” button inside the “Sync Controls” box. This highlights the next quarter of the Vision Matrix. It is possible to control the DV camera during calibration. This means, a user can still move forwards and backwards in the video widget as well as in the floor widget.



Camera Controls
Timecode: 00 : 04 : 30 . 140

Floor Controls
Timecode: 00 : 04 : 44 . 46
Recording Date: 16.10.2002
☑ High Saturation

Sync Controls
Timestamp: 1034755224

Tsai Image
Width: 554 Height: 660

Figure B.6: Selection of Points for Camera Calibration.
Select corresponding points in the Vision Matrix and in the floor widget for every quarter of the Vision Matrix. Press the “Next” button to proceed to the next quarter. It is still possible to control the camera during calibration.

7. As soon as corresponding points are selected for the last quarter of the Vision Matrix (top right), press the “Convert” button inside the “Sync Controls” box. This computes the unified view. Figure B.7 shows the result. The result is also stored in the current working directory under “result.png”. Restart camera calibration if the result is not good enough.

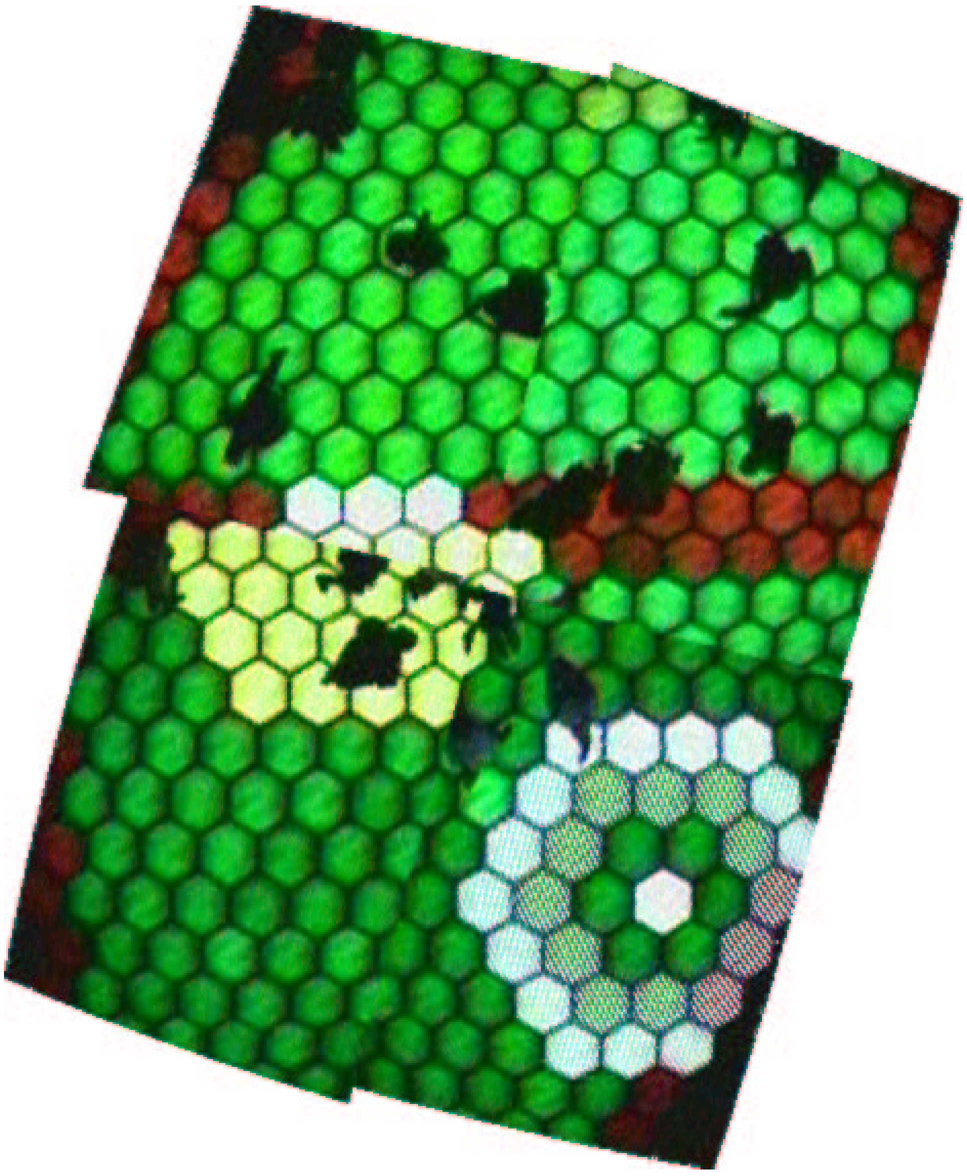


Figure B.7: Unified View.
The result of the computation gets displayed.

Appendix C

Libraw1394 Example Program

```
/********************************************
 *
 * Very simple program to show some Linux
 * libraw1394 library functions.
 * The program sets up the necessary handle
 * and port and then reads USHRT_MAX packets
 * from the isochronous video stream and prints
 * out the section type of the packets to
 * standart out.
 *
 * author: Christoph Kiefer
 * date: 10.6.2003
 *
 ********************************************/

#include <iostream>

#include <raw1394.h>
#include <csr.h>

#include <sys/poll.h>
#include <netinet/in.h>
#include <values.h>

using namespace std;

bool iterate = false;
unsigned short int count = 0;

int handler(raw1394handle_t handle, int channel,
size_t length, quadlet_t *data)
{
  if (count == USHRT_MAX)
    {
      iterate=false;
      return 0;
    }
  // the meaning of packets smaller than 16
  // bytes is not known, therefore not useable
  if (length > 16)
    {
      // actual DV data starts at quadlet 4
      unsigned char *p = (unsigned char*) & data[3];
      int section_type = p[0] >> 5;
      cout << "section type of packet number "
       << count  << " is " << section_type << endl;
      count++;
      return 0;
    }
  else
    {
      count++;
      return 0;
    }
}

int main(int argc, char **argv)
{
  raw1394handle_t handle;
  struct pollfd pfd[1];
  int numcards;
  int port, channel;
  struct raw1394_portinfo pinf[16];
  iso_handler_t oldhandler;

  // get the handle to the kernel side of raw1394
  handle = raw1394_new_handle();
  if (!handle)
    {
      cout << "couldn’t get handle" << endl;
      return 1;
    }

  // get port info (number of cards,
  // connected nodes)
  if ((numcards = raw1394_get_port_info(handle, pinf, 16)) < 0)
    {
      cout << "couldn’t get card info" << endl;
      return 1;
    }
  else
    {
      cout << numcards << " card(s) found" << endl;
      for(int i=0; i<numcards; i++)
{
  cout << "nodes on bus: " << pinf[i].nodes
  << " , card name: " << pinf[i].name << endl;
}
    }
  cout << "enter card: ";
  cin >> port;
  // sets the port
  if (raw1394_set_port(handle, port) < 0)
    {
      cout << "couldn’t set port" << endl;
      return 1;
    }
  else
    {
      // prints out number of nodes on port,
      // ID and IRM (isochronous resource
      // manager id)
      cout << "found " << raw1394_get_nodecount(handle)
<< " nodes on bus, local ID is "
<< (raw1394_get_local_id(handle) & 0x3f)
<< " IRM is " << (raw1394_get_irm_id(handle) & 0x3f)
<< endl;
    }

  // generation number (configuration) of port
  cout << "current generation number of port is "
       << raw1394_get_generation(handle) << endl;

  cout << "enter channel: ";
  cin >> channel;
  // sets the handler
  oldhandler = raw1394_set_iso_handler(handle, channel, handler);

  // file descriptor of handle to poll later
  pfd[0].fd = raw1394_get_fd(handle);
  pfd[0].events = POLLIN|POLLPRI;
  pfd[0].revents = 0;

  // the isochronous video stream starts
  raw1394_start_iso_rcv(handle, channel);

  iterate = true;
  while(iterate)
    {
      // Poll the 1394 interface
      if((poll(&pfd[0], 1, -1) == -1))
{
  cout << "error" << endl;
}
      if(pfd[0].revents & (POLLIN|POLLPRI))
{
  // process packets, calls iso handler repeatedly
  raw1394_loop_iterate(handle);
}
    }

  // stop the isochronous video stream
  raw1394_stop_iso_rcv(handle,channel);
  cout << "program finished" << endl;
  return 0;
}
```

Appendix D

Source Code Documentation

D.1 AdaDVCamera_Application Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

D.2 AdaDVCamera Class Reference

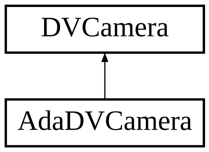
Class to represent the digital video camera that was installed at Ada at Expo.02.

```
#include <adadvcamera.h>
```

Inheritance diagram for AdaDVCamera::

Public Methods

- **AdaDVCamera** ()



Protected Methods

- virtual void **frameToQImage** (**Frame** ∗frame)
- virtual void **checkForMemoryAllocation** ()

Private Attributes

- **AdaPrettifier** ∗ **_adaPrettifier**

D.2.1 Detailed Description

Class to represent the digital video camera that was installed at Ada at Expo.02.

AdaDVCamera inherits **DVCamera** (p. 58) and therefore uses the same functionality as **DVCamera** (p. 58) but overloads methods **frameToQImage**() (p. 45) and **checkForMemoryAllocation**() (p. 44). This is necessary because there were four pan/tilt color cameras installed at Ada whose four individual video streams were put together into one using a quad split.

Author:
Christoph Kiefer

Date:
6/6/2003

D.2.2 Constructor & Destructor Documentation

D.2.2.1 AdaDVCamera::AdaDVCamera ()

Constructor

D.2.3 Member Function Documentation

D.2.3.1 virtual void AdaDVCamera::checkForMemoryAllocation () `[protected, virtual]`

Checks if the image buffer and the QImage which are needed in method **frameToQImage**() (p. 45) are already allocated. It gives the class the possibility to allocate only the needed amount of memory. This amount is less than in class **DVCamera** (p. 58) because not the whole raw frame data is used.

Reimplemented from **DVCamera** (p. 61).

D.2.3.2 virtual void AdaDVCamera::frameToQImage (Frame ∗ *frame*) `[protected, virtual]`

This overloaded method extracts and rearranges the four individual images from the raw frame data and stores them in the QImage in the right way. The result is one properly arrange QImage.

Parameters:

frame The frame to copy to a QImage.

Reimplemented from **DVCamera** (p. 61).

The documentation for this class was generated from the following file:

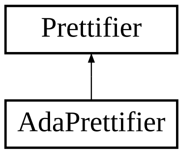
- **adadvcamera.h**

D.3 AdaPrettifier Class Reference

Class to prettify the video data grabbed at Ada/Expo.02.

`#include <adaprettifier.h>`

Inheritance diagram for AdaPrettifier::



Public Methods

- **AdaPrettifier** ()
- virtual void **rearrange** (QImage &image, unsigned char ∗data_buffer)

D.3.1 Detailed Description

Class to prettify the video data grabbed at Ada/Expo.02.

This class reads video data from a data buffer and prettifies it. Thus the video data of the four sub images is written to a QImage in the right rotated way.

Author:

Christoph Kiefer

Date:

6/6/2003

D.3.2 Constructor & Destructor Documentation

D.3.2.1 AdaPrettifier::AdaPrettifier ()

Constructor

D.3.3 Member Function Documentation

D.3.3.1 virtual void AdaPrettifier::rearrange (QImage & *image*, unsigned char ∗ *data_buffer*) `[virtual]`

This virtual method implements how to extract the four individual images from the data buffer. To know at which positions extraction should start, one unarranged image was measured out with an image processing tool.

Parameters:

image QImage to hold the final data.

data_buffer Holding the original frame data.

Implements **Prettifier** (p. 80).

The documentation for this class was generated from the following file:

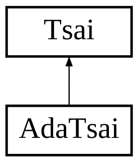
- **adaprettifier.h**

D.4 AdaTsai Class Reference

Inherits class **Tsai** (p. 84) and implements methods to use **Tsai** (p. 84) in the Ada./Expo.02 example.

```
#include <adatsai.h>
```

Inheritance diagram for AdaTsai::



Public Slots

- virtual void **startCalibration** ()
- virtual void **createCalibrationFile** (int fileNumber, QPtrList< **imagePoint** > ∗iPoints, QPtrList< **worldPoint** > ∗wPoints)
- virtual void **convert** (double wWorld, double hWorld, int w, int h)
- void **setCurrentImage** (QImage image)

Public Methods

- **AdaTsai** (int tsaiInstances, QImage &oldImage, QImage &tsaiImage)

Private Attributes

- double **_midPointsWorld** [8]
- int **_midPointsImage** [8]

D.4.1 Detailed Description

Inherits class **Tsai** (p. 84) and implements methods to use **Tsai** (p. 84) in the Ada./Expo.02 example.

The class implements the abstract methods from class **Tsai** (p. 84) to fulfill the calibration of the four cameras at Ada/Expo.02.

Author:
Christoph Kiefer

Date:
6/6/2003

D.4.2 Constructor & Destructor Documentation

D.4.2.1 AdaTsai::AdaTsai (int *tsaiInstances*, QImage & *oldImage*, QImage & *tsaiImage*)

Constructor

Parameters:
tsaiInstances Number of instances of class TsaiCal.
oldImage Image out of which a unified view should be computed.
tsaiImage The image which will hold the unified view.

D.4.3 Member Function Documentation

D.4.3.1 virtual void AdaTsai::convert (double *wWorld*, double *hWorld*, int *w*, int *h*) `[virtual, slot]`

Computes the final unified view. For every given world point it calculates the shortest distance to a middle point of one image sector. That world point is then mapped to an image point by the means of the corresponding camera parameters.

Parameters:
wWorld x-coordinate in the world (in mm) of point w
hWorld y-coordinate in in the world (in mm) of point h
w x-coordinate in the image (that image that hold the unified view), in pixels
h y-coordinate in the image (that image that hold the unified view), in pixels

Implements **Tsai** (p. 85).

D.4.3.2 virtual void AdaTsai::createCalibrationFile (int *fileNumber*, QPtrList< imagePoint > ∗ *iPoints*, QPtrList< worldPoint > ∗ *wPoints*) `[virtual, slot]`

Writes the calibration files and uses the calculated camera parameters to compute the world point of the middle point of the image to be unified.

Parameters:

fileNumber Number or index of the calibration files that get written.

iPoints A list of image points.

wPoints A list of world points.

Implements **Tsai** (p. 86).

D.4.3.3 void AdaTsai::setCurrentImage (QImage *image*) `[slot]`

Sets a new source image out of which a unified view should be calculated.

Parameters:

image Source image to calculate unified view.

D.4.3.4 virtual void AdaTsai::startCalibration () `[virtual, slot]`

The middle points of each image sector are set up. The image out of which the unified view is to be calculated should used therefore. This middle points are necessary to later determine the shortest distance between that middle point and another world point. And this in turn is absolutely necessary to define which camera parameters to take to finally map the world point back to an image point in the unified view.

Implements **Tsai** (p. 86).

The documentation for this class was generated from the following file:

- **adatsai.h**

D.5 AdaViewer Class Reference

Image viewer class.

```
#include <adaviewer.h>
```

Public Slots

- void **selectPoint** (QPoint point)
- void **magnify** (QMouseEvent ∗mme)
- void **setImage** (QImage image)
- void **setMagnify** (bool state)

Public Methods

- **AdaViewer** (QWidget ∗parent=0, const char ∗name=0, WFlags f=0)
- QPtrList< **imagePoint** > ∗ **getPointList** ()
- void **highlightArea** (int currentArea)
- void **clearPointList** ()

Protected Methods

- virtual void **polishContent** (QPainter &p)
- virtual void **mouseMoveEvent** (QMouseEvent ∗mme)
- virtual void **mousePressEvent** (QMouseEvent ∗mme)

Private Methods

- bool **pointAllreadySelected** (QPoint p)
- void **removePoint** (QPoint p)

Private Attributes

- QPtrList< **imagePoint** > ∗ **_viewerPoints**
- int **_selectedPoints**
- int **_currentArea**
- QPixmap **pm**
- QPixmap **_image**
- int **x**
- int **y**
- bool **_doMagnify**
- int **_mlengthHalf**
- int **_zoomMlengthHalf**
- int **_zoomMlength**

D.5.1 Detailed Description

Image viewer class.

This class implements an image viewer for the images that are constructed out of the video frames captured at Ada/Expo.02.

Author:
Christoph Kiefer

Date:
6/6/2003

D.5.2 Constructor & Destructor Documentation

D.5.2.1 AdaViewer::AdaViewer (QWidget ∗ *parent* = 0, const char ∗ *name* = 0, WFlags *f* = 0)

Constructor.

Parameters:

parent Pointer to parent widget. Zero (default) for top-level widgets.

name Identifier internally used by Qt for identifying individual class instances.

f Widget-specific flags.

D.5.3 Member Function Documentation

D.5.3.1 void AdaViewer::clearPointList ()

Deletes all points on the list of image points.

D.5.3.2 QPtrList<imagePoint>∗ AdaViewer::getPointList ()

Returns the list of user selected points.

Returns:

A list of pointers to image points.

D.5.3.3 void AdaViewer::highlightArea (int *currentArea*)

One area of the displayed image is highlighted. The other 3 areas are shadowed.

Parameters:

currentArea The area of the image to highlight.

D.5.3.4 void AdaViewer::magnify (QMouseEvent ∗ *mme*) `[slot]`

Slot gets called when an area on the Vision Matrix should be magnified.

Parameters:

mme Mouse event. Holds position of mouse pointer.

D.5.3.5 virtual void AdaViewer::mouseMoveEvent (QMouseEvent ∗ *mme*) `[protected, virtual]`

This virtual method defines what happens if the user moves the mouse over the widget. If the magnifier checkbox is clicked, a small area around QMouseEvent::pos is magnified.

Parameters:

re Mouse event.

D.5.3.6 virtual void AdaViewer::mousePressEvent (QMouseEvent ∗ *mme*) `[protected, virtual]`

Catch mouse press events. Depending on mouse button, either a point is selected or dis selected from the Vision Matrix.

Parameters:
mme Mouse event.

D.5.3.7 bool AdaViewer::pointAllreadySelected (QPoint *p*) `[private]`

Goes through the list of image points and looks up if a given point was already selected by the user.

Parameters:
p Point to look for in the list.

Returns:
`true` if point is already in the list, \false otherwise.

D.5.3.8 virtual void AdaViewer::polishContent (QPainter & *p*) `[protected, virtual]`

Enhance image content before displaying it.

This virtual method can be overloaded in a derived class for changing/enhancing the image's content before it is displayed. The painter `p` is open when it is passed to this method and must still be opened when leaving the method. The method draws lines on the 3 areas which are not to highlight.

Parameters:
p Painter for drawing into the given image.

D.5.3.9 void AdaViewer::removePoint (QPoint *p*) `[private]`

Remove a point from the Vision Matrix.

Parameters:
The point to remove.

D.5.3.10 void AdaViewer::selectPoint (QPoint *point*) `[slot]`

This slot gets called whenever a user selects a point on the image.

Parameters:
point The point the user has clicked (x- & y-coordinate within image).

D.5.3.11 void AdaViewer::setImage (QImage *image*) `[slot]`

Set a new image. Takes a local copy of the image.

Parameters:
image New image to be displayed.

D.5.3.12 void AdaViewer::setMagnify (bool *state*) `[slot]`

Sets magnification on/off.

Parameters:

state `true` or `false`

The documentation for this class was generated from the following file:

- **adaviewer.h**

D.6 ApplicationWindow Class Reference

Class that implements the whole GUI around the application. .

`#include <application.h>`

Public Slots

- void **play** ()
- void **stop** ()
- void **fastForward** ()
- void **rewind** ()
- void **next** ()
- void **previous** ()
- void **updateTimeCodeFrame** (QImage image)
- void **updateTimeCodeLog** (QString timecode)
- void **setCurrentTimecode** (const QString &currentTimecode)
- void **setRefFrameTimestamp** ()
- void **setImage** (QImage image)

Public Methods

- **ApplicationWindow** (QWidget ∗parent=0, const char ∗name=0, WFlags f=0)

Protected Methods

- virtual void **closeEvent** (QCloseEvent ∗ce)
- virtual void **resizeEvent** (QResizeEvent ∗re)

Private Slots

- void **synchronize** ()
- void **deSynchronize** ()
- void **startCalibration** ()
- void **createCalibrationFile** ()
- void **convert** ()
- void **setTsaiImageWidth** (const QString &height)
- void **setTsaiImageHeight** (const QString &width)
- void **changeLock** ()
- void **encodeUnifiedView** ()

Private Methods

- void **setupGUI** ()

Private Attributes

- **AdaDVCamera** ∗ **_dvcam**
- QPushButton ∗ **_startCaptureButton**
- QPushButton ∗ **_stopCaptureButton**
- QPushButton ∗ **_nextFrameButton**
- QPushButton ∗ **_prevFrameButton**
- QPushButton ∗ **_ffButton**
- QPushButton ∗ **_rewindButton**
- QPushButton ∗ **_selectLogButton**
- QPushButton ∗ **_nextLogEntryButton**
- QPushButton ∗ **_prevLogEntryButton**
- QPushButton ∗ **_syncButton**
- QPushButton ∗ **_lockButton**
- QPushButton ∗ **_calibrateButton**
- QHBoxLayout ∗ **_mainLayout**
- QWidget ∗ **_viewerWidget**
- QWidget ∗ **_floor**
- QWidget ∗ **_widgetA**
- QWidget ∗ **_widgetB**
- QGroupBox ∗ **_gbViewer**
- QGroupBox ∗ **_gbFloor**
- QGroupBox ∗ **_gbSync**
- QGroupBox ∗ **_videoBox**
- QGroupBox ∗ **_floorBox**
- QGroupBox ∗ **_gbTsai**
- QGroupBox ∗ **_gbMisc**
- QGridLayout ∗ **_leftLayout**
- QGridLayout ∗ **_rightLayout**
- QGridLayout ∗ **_gridViewer**
- QGridLayout ∗ **_gridFloor**
- QButtonGroup ∗ **_buttonGroupV**
- QButtonGroup ∗ **_buttonGroupF**
- QButtonGroup ∗ **_buttonGroupS**
- QButtonGroup ∗ **_buttonGroupM**
- QHBoxLayout ∗ **_buttonGroupLayoutV**
- QHBoxLayout ∗ **_buttonGroupLayoutF**
- QHBoxLayout ∗ **_buttonGroupLayoutS**
- QHBoxLayout ∗ **_buttonGroupLayoutM**
- QLineEdit ∗ **_logHours**
- QLineEdit ∗ **_logMinutes**
- QLineEdit ∗ **_logSeconds**
- QLineEdit ∗ **_logMiliSeconds**
- QLineEdit ∗ **_frameHours**
- QLineEdit ∗ **_frameMinutes**
- QLineEdit ∗ **_frameSeconds**

- QLineEdit ∗ **_frameMiliSeconds**
- QLineEdit ∗ **_timestamp**
- QLineEdit ∗ **_tsaiWidth**
- QLineEdit ∗ **_tsaiHeight**
- QCheckBox ∗ **_saturationCBox**
- QHBox ∗ **_frameRecDateBox**
- QHBox ∗ **_logRecDateBox**
- QHBox ∗ **_syncBox**
- QHBox ∗ **_miscBox**
- QSpacerItem ∗ **_spacer**
- QSpacerItem ∗ **_spacer2**
- QValidator ∗ **_tsaiValidator**
- QLabel ∗ **_frameRecDate**
- QLabel ∗ **_logRecDate**
- QImage **_image**
- QImage **_currentImage**
- QImage **_tsaiImage**
- int **_tsaiImageWidth**
- int **_tsaiImageHeight**
- **AdaViewer** ∗ **_viewer**
- **AdaTsai** ∗ **_tsai**
- **FloorWidget** ∗ **_floorWidget**
- QString **_currentTimecode**
- int **_currentArea**
- **Scaler** ∗ **_targetScaler**
- QCheckBox ∗ **_magnifier**
- QCheckBox ∗ **_encodeCBox**
- QPushButton ∗ **_goButton**
- **VideoEncoder** ∗ **_encoder**
- **TsaiViewer** ∗ **_tsaiViewer**
- bool **_encodeUnified**

D.6.1 Detailed Description

Class that implements the whole GUI around the application. .

This class implements a GUI to use the classes **AdaDVCamera** (p. 43) and **FloorWidget** (p. 65). Various buttons and fields give the user the opportunity to control the DV camera and to start the camera calibration using **Tsai** (p. 84)'s method.

Author:
Christoph Kiefer

Date:
6/6/2003

D.6.2 Constructor & Destructor Documentation

D.6.2.1 ApplicationWindow::ApplicationWindow (QWidget ∗ *parent* = 0, const char ∗ *name* = 0, WFlags *f* = 0)

Constructor.

Parameters:

parent Pointer to parent widget. Zero (default) for top-level widgets.

name Identifier internally used by Qt for identifying individual class instances.

f Widget-specific flags.

D.6.3 Member Function Documentation

D.6.3.1 void ApplicationWindow::changeLock () `[private, slot]`

Changes the state of the lock button. If the video and the floor are unlocked moving forward or backward one frame in the video does not automatically move forward or backward the log and vise versa.

D.6.3.2 virtual void ApplicationWindow::closeEvent (QCloseEvent ∗ *ce*) `[protected, virtual]`

This virtual method defines what happens if the user closes the main widget. If the camera is still playing it is told to stop.

Parameters:

ce Close event.

D.6.3.3 void ApplicationWindow::convert () `[private, slot]`

Produces the unified view.

D.6.3.4 void ApplicationWindow::createCalibrationFile () `[private, slot]`

For each area of the image two corresponding files are created. One holding the pairs of points for that area, the other one consists of the calculated camera parameters (by **Tsai** (p. 84)).

D.6.3.5 void ApplicationWindow::deSynchronize () `[private, slot]`

Defines the action that is performed if the user presses the sync button when a synchronization of the video and the log has already been done.

D.6.3.6 void ApplicationWindow::encodeUnifiedView () `[private, slot]`

Starts encoding the unified view to a video file. Allocates a new **VideoEncoder** (p. 89).

D.6.3.7 void ApplicationWindow::fastForward () `[slot]`

Winds the camera fast forward.

D.6.3.8 void ApplicationWindow::next () `[slot]`

Displays the next video frame.

D.6.3.9 void ApplicationWindow::play () `[slot]`

Start playing the camera.

D.6.3.10 void ApplicationWindow::previous () `[slot]`

Displays the previous video frame.

D.6.3.11 virtual void ApplicationWindow::resizeEvent (QResizeEvent ∗ *re*) `[protected, virtual]`

This virtual method defines what happens if the user resizes the main widget.

Parameters:
re Resize event.

D.6.3.12 void ApplicationWindow::rewind () `[slot]`

Rewinds the camera.

D.6.3.13 void ApplicationWindow::setCurrentTimecode (const QString & *currentTimecode*) `[slot]`

Stores the timecode of the current frame to a variable.

Parameters:
currentTimecode The timecode.

D.6.3.14 void ApplicationWindow::setImage (QImage *image*) `[slot]`

The current QImage to display. To clarify: the current frame to display is synonym with the current image to display.

Parameters:
image The image.

D.6.3.15 void ApplicationWindow::setRefFrameTimestamp () `[slot]`

Sets the timestamp of the reference frame.

D.6.3.16 void ApplicationWindow::setTsaiImageHeight (const QString & *width*) `[private, slot]`

Does basically the same as **setTsaiImageWidth**() (p. 57) but sets the height instead of the width of the unified view.

Parameters:

width Gets width and calculates maximum height.

D.6.3.17 void ApplicationWindow::setTsaiImageWidth (const QString & *height*) `[private, slot]`

Sets the width of the unified view.

Parameters:

height Gets the height and calculates the maximum width to still respect the ratio of the Ada floor width and height.

D.6.3.18 void ApplicationWindow::setupGUI () `[private]`

Sets up all GUI elements.

D.6.3.19 void ApplicationWindow::startCalibration () `[private, slot]`

Starts the camera calibration in which the user must select pairs of image and world points in every of the four areas of the image.

D.6.3.20 void ApplicationWindow::stop () `[slot]`

Stop playing the camera.

D.6.3.21 void ApplicationWindow::synchronize () `[private, slot]`

Defines the action performed if the user clicks on the sync button.

D.6.3.22 void ApplicationWindow::updateTimeCodeFrame (QImage *image*) `[slot]`

Sets the timecode of the current frame in the appropriated field.

Parameters:

image The timecode of the current frame is attached to current image to display.

D.6.3.23 void ApplicationWindow::updateTimeCodeLog (QString *timecode*) `[slot]`

Sets the timecode of the current log entry in the appropriated field.

Parameters:

timecode The timecode.

The documentation for this class was generated from the following file:

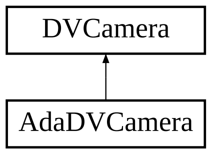
- **application.h**

D.7 DVCamera Class Reference

Control digital video camera connected to IEEE1394 interface.

`#include <dvcamera.h>`

Inheritance diagram for DVCamera::



Public Slots

- void **startCapture** ()
- void **stopCapture** ()
- void **pauseCapture** ()
- QImage **next** ()
- QImage **previous** ()
- void **stop** ()
- void **play** ()
- void **fastForward** ()
- void **rewind** ()
- void **pauseCamera** ()
- void **setTimerValue** (int value)

Signals

- void **captureStarted** ()
- void **captureStopped** ()
- void **ff** ()
- void **rw** ()
- void **frameProcessed** (QImage image)
- void **bufferEmpty** ()
- void **outputEmpty** ()
- void **outputNotEmpty** ()
- void **signalRefFrameTimecode** (const QString &refFrameTimecode)
- void **signalFrameTimecode** (const QString &frameTimecode)

Public Methods

- **DVCamera** ()
- bool **bufferQueueIsEmpty** ()
- bool **outputQueueIsEmpty** ()
- bool **isPlaying** ()
- void **setQImage** (QImage &image)

Static Public Methods

- int **avi_iso_handler** (raw1394handle_t handle, int channel, size_t length,quadlet_t ∗data)
- int **this_reset_handler** (raw1394handle_t handle, unsigned int generation)

Static Public Attributes

- QMap< raw1394handle_t, DVCamera ∗ > **map**
- int **currentRetries**
- int **maxRetries**

Protected Methods

- void **send_avc_command** (raw1394handle_t handle, nodeid_t node, quadlet_t command)
- void **startCamera** ()
- void **stopCamera** ()
- void **playSlowestReverse** ()
- virtual void **frameToQImage** (**Frame** ∗frame)
- virtual void **checkForMemoryAllocation** ()

Protected Attributes

- **IEEE1394IOHandler** ∗ **_handler**
- raw1394handle_t **_handle**
- unsigned char ∗ **_image_buffer**
- **Frame** ∗ **_current_frame**
- QImage **_image**
- QTimer ∗ **_timer**
- QPtrList< **Frame** > **_buffer_queue**
- QPtrList< **Frame** > **_output_queue**
- QMutex **_buffer_Mutex**
- QMutex **_output_Mutex**
- QWaitCondition **_pass**
- bool **_isPlaying**
- bool **_refFrameTimecodeSend**
- int **_value**

D.7.1 Detailed Description

Control digital video camera connected to IEEE1394 interface.

Class DVCamera provides methods to control a digital video camera that is connected to the IEEE1394 (FireWire, i.Link) serial bus. To control the camera, methods like **play**() (p. 62) send AV/C (Audio/Video Control) commands using a **_handle** (p. 64) obtained from **IEEE1394IOHandler::get_raw1394_handle**() (p. 76). The **_handle** (p. 64) provides a connection to the kernel side of raw1394. The implemented AVI/ISO (Audio Video Interleave) handler grabs raw audio/video data and packs it into **frames** (p. 73). These frames then share either the **_buffer_queue** (p. 64) or the **_output_queue** (p. 65). Because these queues are shared between this class and class **IEEE1394IOHandler** (p. 75) that calls repeatedly the static member function **avi_iso_handler**() (p. 60), mutexes (**_buffer_Mutex** (p. 64), **_output_Mutex** (p. 65)) are used to gain and release access to these queues. DVCamera does not handle any audio data.

Author:
Christoph Kiefer

Date:
6/6/2003

D.7.2 Constructor & Destructor Documentation

D.7.2.1 DVCamera::DVCamera ()

Constructor.

D.7.3 Member Function Documentation

D.7.3.1 int DVCamera::avi_iso_handler (raw1394handle_t *handle*, int *channel*, size_t *length*, quadlet_t ∗ *data*) `[static]`

Implements the AVI/ISO handler as a static member function. When the camera is playing this function is repeatedly called from **IEEE1394IOHandler::run**() (p. 77).

Parameters:

handle Provides a connection to kernel side of raw1394.

channel ISO channel used for transmitting the video data. Default is 63, the channel most DV cameras are using.

length Length of packet to process.

data The raw PAL/NTSC DV data packet.

D.7.3.2 void DVCamera::bufferEmpty () `[signal]`

Signal is emitted if **_buffer_queue** (p. 64) is empty.

D.7.3.3 bool DVCamera::bufferQueueIsEmpty ()

Returns `true` if the **_buffer_queue** (p. 64) is empty, `false` otherwise.

D.7.3.4 void DVCamera::captureStarted () `[signal]`

Signal is emitted at the end of method **startCapture**() (p. 64).

D.7.3.5 void DVCamera::captureStopped () `[signal]`

Signal is emitted at the end of method stopCapture.

D.7.3.6 virtual void DVCamera::checkForMemoryAllocation () `[protected, virtual]`

This virtual method can be overloaded in a derived class. It checks if the image buffer and the QImage which are needed in method **frameToQImage**() (p. 61) are already allocated. It gives the class the possibility to allocate only the needed amount of memory.

Reimplemented in **AdaDVCamera** (p. 44).

D.7.3.7 void DVCamera::fastForward () `[slot]`

First flushes **_buffer_queue** (p. 64) and **_output_queue** (p. 65) and tells the camera to wind fast forward. Calls **send_avc_command**() (p. 63). AV/C command operand is "VCR_-OPERAND_PLAY_FAST_FORWARD".

D.7.3.8 void DVCamera::ff () `[signal]`

Signal is emitted at the end of method **fastForward**() (p. 61).

D.7.3.9 void DVCamera::frameProcessed (QImage *image*) `[signal]`

Signal is emitted when next frame from @_output_queue has been processed.

Parameters:
image Image holding extracted frame data.

D.7.3.10 virtual void DVCamera::frameToQImage (Frame ∗ *frame*) `[protected, virtual]`

This virtual method can be overloaded in a derived class to change the way in which the frame data is copied to the QImage.

Parameters:
frame The frame to copy to a QImage.

Reimplemented in **AdaDVCamera** (p. 45).

D.7.3.11 bool DVCamera::isPlaying ()

Returns `true` if the camera is currently playing, `false` otherwise.

D.7.3.12 QImage DVCamera::next () `[slot]`

Takes next frame from output queue, calls **frameToQImage**() (p. 61) and emits a **frame-Processed**() (p. 61) signal.

D.7.3.13 void DVCamera::outputEmpty () `[signal]`

Signal is emitted if **_output_queue** (p. 65) is empty.

D.7.3.14 void DVCamera::outputNotEmpty () `[signal]`

Signal is emitted if **_output_queue** (p. 65) is not empty.

D.7.3.15 bool DVCamera::outputQueueIsEmpty ()

Returns `true` if the **_output_queue** (p. 65) is empty, `false` otherwise.

D.7.3.16 void DVCamera::pauseCamera () `[slot]`

Pauses the camera. Calls **send_avc_command**() (p. 63). AV/C command operand is "VCR_OPERAND_PLAY_FORWARD_PAUSE".

D.7.3.17 void DVCamera::pauseCapture () `[slot]`

Tells **IEEE1394IOHandler** (p. 75) to stop polling the interface. Also calls **pause-Camera**() (p. 62).

D.7.3.18 void DVCamera::play () `[slot]`

Slot to call **startCapture**() (p. 64).

D.7.3.19 void DVCamera::playSlowestReverse () `[protected]`

Calls **send_avc_command**() (p. 63). AV/C command operand is "VCR_OPERAND_-PLAY_SLOWEST_REVERSE".

D.7.3.20 QImage DVCamera::previous () `[slot]`

Takes the most recently processed frame from the tail of the **_buffer_queue** (p. 64), puts it back to the front of the **_output_queue** (p. 65) and processes like **next**() (p. 62).

D.7.3.21 void DVCamera::rewind () `[slot]`

Behaves actually in the same way as **fastForward**() (p. 61) but tells the camera to rewind. Calls **send_avc_command**() (p. 63). AV/C command operand is "VCR_OPERAND_-WIND_REWIND".

D.7.3.22 void DVCamera::rw () `[signal]`

Signal is emitted at the end of method **rewind**() (p. 62).

D.7.3.23 void DVCamera::send_avc_command (raw1394handle_t *handle*, nodeid_t *node*, quadlet_t *command*) `[protected]`

Method to send AV/C commands to the DV camera. It will call IEEE1394IOHandler::cooked1394_write().

Parameters:

handle Provides a connection to kernel side of raw1394.

node Physical device id of DV camera. Default is 1.

command AV/C command.

D.7.3.24 void DVCamera::setQImage (QImage & *image*)

Set image.

Parameters:

image Image where extracted frame data should be stored.

D.7.3.25 void DVCamera::setTimerValue (int *value*) `[slot]`

Sets the value of the timer. The timer continuously calls **next**() (p. 62) that grabs frames from the **_output_queue** (p. 65).

Parameters:

value Timer value in milliseconds.

D.7.3.26 void DVCamera::signalFrameTimecode (const QString & *frameTimecode*) `[signal]`

Signal is emitted whenever a frame is processed.

Parameters:

frameTimecode The timecode of the frame.

D.7.3.27 void DVCamera::signalRefFrameTimecode (const QString & *refFrameTimecode*) `[signal]`

Signal is emitted when first frame from DV tape is processed.

Parameters:

refFrameTimecode The timecode of the first frame.

D.7.3.28 void DVCamera::startCamera () `[protected]`

Sends AV/C command "VCR_OPERAND_PLAY_FORWARD" to camera.

D.7.3.29 void DVCamera::startCapture () `[slot]`

Tells **IEEE1394IOHandler** (p. 75) to start polling the interface. Also calls startcamera().

D.7.3.30 void DVCamera::stop () `[slot]`

Slot to call **stopCapture**() (p. 64).

D.7.3.31 void DVCamera::stopCamera () `[protected]`

Calls **send_avc_command**() (p. 63). AV/C command operand is "VCR_OPERAND_-WIND_STOP".

D.7.3.32 void DVCamera::stopCapture () `[slot]`

Tells **IEEE1394IOHandler** (p. 75) to stop polling the interface. Also calls **stopCamera**() (p. 64) stopCamera.

D.7.3.33 int DVCamera::this_reset_handler (raw1394handle_t *handle*, unsigned int *generation*) `[static]`

The handler that is called when a bus reset message is encountered.

Parameters:

handle Provides a connection to kernel side of raw1394.

generation Current configuration of the port.

D.7.4 Member Data Documentation

D.7.4.1 QMutex DVCamera::_buffer_Mutex `[protected]`

Mutex to protect the critical resource **_buffer_queue** (p. 64).

D.7.4.2 QPtrList<Frame > DVCamera::_buffer_queue `[protected]`

Frames are buffered on the this queue before they go to the **_output_queue** (p. 65).

D.7.4.3 raw1394handle_t DVCamera::_handle `[protected]`

Provides a connection to kernel side of raw1394.

D.7.4.4 IEEE1394IOHandler∗ DVCamera::_handler `[protected]`

IEEE1394IOHandler (p. 75).

D.7.4.5 QMutex DVCamera::_output_Mutex `[protected]`

Mutex to protect the critical resource **_output_queue** (p. 65).

D.7.4.6 QPtrList<Frame > DVCamera::_output_queue `[protected]`

Frames to display are taken from the this queue.

D.7.4.7 QTimer∗ DVCamera::_timer `[protected]`

The timer emits signal timeout to initiated processing of the next frame from the **_output_-queue** (p. 65) and is started in method **play**() (p. 62) and stopped in method **stop**() (p. 64).

The documentation for this class was generated from the following file:

- **dvcamera.h**

D.8 FloorWidget Class Reference

Class representing the Ada floor.

```
#include <floorwidget.h>
```

Public Slots

- void **nextLogEntry** ()
- void **previousLogEntry** ()
- void **setRefFrameTimestamp** (const QString &refFrameTimestamp)
- void **setRefFrameTimecode** (const QString &refFrameTimecode)
- void **synchronize** (const QString &frameTimecode)
- void **changeLockState** ()
- QPointArray **definePolygon** (int tile, **Scaler** ∗scaler)

Signals

- void **pixmapCreated** (QString timestamp)
- void **backQueueIsEmpty** ()
- void **floorPointSelected** (QPoint p)

Public Methods

- **FloorWidget** (QWidget ∗parent=0, const char ∗name=0)
- QPtrList< **worldPoint** > ∗ **getPointList** ()
- void **signalResizeHandler** (QResizeEvent ∗re)
- void **clearPointList** ()
- double **getFloorWidth** ()
- double **getFloorHeight** ()
- double **getMinX** ()
- double **getMaxY** ()
- bool **refFrameTimestampIsSet** ()

Protected Methods

- virtual void **paintEvent** (QPaintEvent $*$pe)
- virtual void **mousePressEvent** (QMouseEvent $*$me)

Private Slots

- void **selectPoint** (const QPoint &p)
- void **selectnewLog** ()
- void **changeSaturation** ()

Private Methods

- void **setLogFile** (const QString &filename)
- void **loadNextLogLine** (string logline)
- void **loadFloorTopologyFile** ()
- void **setPoints** (const int index, const int na, const int nb, double $*$cornerPoints, const int offset)
- void **createPixmap** ()
- void **setTileOutlines** ()
- QString **getCurrentTimestamp** ()
- bool **pointAllreadySelected** (double xw, double yw)
- void **removePoint** (const QPoint &p)
- QPointArray **definePolygon** (int tile)

Private Attributes

- **Tile tiles** [360]
- QPixmap $*$ **_buffer**
- QString **logFilename**
- ifstream **logStream**
- string **logLine**
- string **_currentLine**
- string **_nextLine**
- int **_logsRead**
- QColor **_back_queue** [MAX_BACK_QUEUE_LENGTH][360]
- char **_ts_queue** [MAX_BACK_QUEUE_LENGTH][20]
- int **_current_queue_index**
- int **_next_free_queue_index**
- bool **_saturate**
- long long **_firstTimestamp**
- long long **_refFrameTimestamp**
- long long **_refFrameTimecode**
- long long **_timestampToSynchronize**
- QPtrList$<$ **worldPoint** $>$ $*$ **_floorPoints**
- int **_selectedPoints**
- long long **_dif1**
- long long **_dif2**
- bool **_synchronized**

- bool **_locked**
- double **_minX**
- double **_maxX**
- double **_minY**
- double **_maxY**
- double **_floorWidth**
- double **_floorHeight**
- **Scaler** $*$ **_floorScaler**
- int **_xDist**
- int **_correctionValue**

D.8.1 Detailed Description

Class representing the Ada floor.

The class implements methods to draw the Ada floor that consists of 360 pentagonal tiles. Therefore a log file needs to be read and processed and the user also has the possibility to select corner points of one tile. There is also a method that searches a log entry by the means of a given timestamp.

Author:
Christoph Kiefer

Date:
6/6/2003

D.8.2 Constructor & Destructor Documentation

D.8.2.1 FloorWidget::FloorWidget (QWidget $*$ *parent* = 0, const char $*$ *name* = 0)

Constructor.

Parameters:
parent Pointer to parent widget. Zero (default) for top-level widgets.

name Identifier internally used by Qt for identifying individual class instances.

f Widget-specific flags.

D.8.3 Member Function Documentation

D.8.3.1 void FloorWidget::backQueueIsEmpty () `[signal]`

Signal is emitted if the floor log back queue is empty which means further pressing the previous button has no effects.

D.8.3.2 void FloorWidget::changeLockState () `[slot]`

Changes the state of the widget to be locked or unlocked. Refer to **Application-Window::changeLock**() (p. 55) for further explanation.

D.8.3.3 void FloorWidget::changeSaturation () `[private, slot]`

Changes the color saturation of each tile of the Ada floor.

D.8.3.4 void FloorWidget::clearPointList ()

Deletes all points on the list of world points.

D.8.3.5 void FloorWidget::createPixmap () `[private]`

Paints all 360 tiles and all selected user points to a pixmap buffer.

D.8.3.6 QPointArray FloorWidget::definePolygon (int *tile*) `[private]`

Constructs an array which holds the points that define the pentagonal tile. The points are scaled down by the **_floorScaler** (p. 72).

Parameters:

tile The number of the tile to construct its pentagonal shape.

Returns:

The array holding the coordinates of the corners of the tile.

D.8.3.7 QPointArray FloorWidget::definePolygon (int *tile*, Scaler ∗ *scaler*) `[slot]`

Constructs an array which holds the points that define the hexagonal tile. The points are scaled by a scaler.

Parameters:

tile The number of the tile to construct its hexagonal shape.

scaler Scales points.

Returns:

The array holding the coordinates of the corners of the tile.

D.8.3.8 void FloorWidget::floorPointSelected (QPoint *p*) `[signal]`

Signal is emitted as soon as a point on the floor is selected.

Parameters:

p The selected point on the Ada floor.

D.8.3.9 QString FloorWidget::getCurrentTimestamp () `[private]`

Returns the timestamp of the current displayed floor log entry.

Returns:

The timestamp of the current floor log entry.

D.8.3.10 double FloorWidget::getFloorHeight ()

Returns the height of the Ada floor.

Returns:
The Ada floor height.

D.8.3.11 double FloorWidget::getFloorWidth ()

Returns the width of the Ada floor.

Returns:
The Ada floor width.

D.8.3.12 double FloorWidget::getMaxY ()

Return the minimal y-coordinate in meters of the Ada floor.

Returns:
The minimal y-coordinate of the Ada floor.

D.8.3.13 double FloorWidget::getMinX ()

Return the minimal x-coordinate in meters of the Ada floor.

Returns:
The minimal x-coordinate of the Ada floor.

D.8.3.14 QPtrList<worldPoint>* FloorWidget::getPointList ()

Returns the list of user selected points.

Returns:
A list of pointers to world points.

D.8.3.15 void FloorWidget::loadFloorTopologyFile () `[private]`

Loads the floor topology file. This file contains all center coordinates of the tiles and each tile's neighbors as well as some other parameters that are not used in this application.

D.8.3.16 void FloorWidget::loadNextLogLine (string *logline*) `[private]`

Loads the next line in the log file into memory, that is fills up the **_back_queue** (p. 72) and the **_ts_queue** (p. 72).

Parameters:
logline The line to load into memory.

D.8.3.17 virtual void FloorWidget::mousePressEvent (QMouseEvent $*$ *me*) `[protected, virtual]`

Virtual method to handle mouse press events that occurs if the user wants to select a point on the floor.

Parameters:
me The catched mouse event.

D.8.3.18 void FloorWidget::nextLogEntry () `[slot]`

Goes to the next log entry.

D.8.3.19 virtual void FloorWidget::paintEvent (QPaintEvent $*$ *pe*) `[protected, virtual]`

This virtual method handles paint events which occur in the case of a resize event.

Parameters:
pe The catched paint event.

D.8.3.20 void FloorWidget::pixmapCreated (QString *timestamp*) `[signal]`

Signal is emitted as soon as the floor is painted that is after **createPixmap**() (p. 68).

Parameters:
timestamp The timestamp of the painted floor log entry.

D.8.3.21 bool FloorWidget::pointAllreadySelected (double *xw*, double *yw*) `[private]`

Goes through the list of world points and looks up if a given point was already selected by the user.

Parameters:
xw X-coordinate of world point.
yw Y-coordinate of world point.

Returns:
`true` if point is already in the list, \false otherwise.

D.8.3.22 void FloorWidget::previousLogEntry () `[slot]`

Goes to the previous log entry.

D.8.3.23 bool FloorWidget::refFrameTimestampIsSet ()

Return `true` if the timestamp of the reference frame is set, `false` otherwise. This is important because only if the timestamp is set, the floor log can be synchronized with the video.

Returns:

\true if timestamp of reference frame is set, `false` otherwise.

D.8.3.24 void FloorWidget::removePoint (const QPoint & *p*) `[private]`

Removes a point from the list of world points.

Parameters:

p The point to remove.

D.8.3.25 void FloorWidget::selectnewLog () `[private, slot]`

Opens a file dialog to select a log file.

D.8.3.26 void FloorWidget::selectPoint (const QPoint & *p*) `[private, slot]`

Selects a point on the Ada floor by first finding out to which tile the user selected point belongs and afterward storing the point in the list of world points.

Parameters:

p The point the user selected by clicking on the widget.

D.8.3.27 void FloorWidget::setLogFile (const QString & *filename*) `[private]`

Sets up necessary variables to log the log file.

Parameters:

filename The chosen log file.

D.8.3.28 void FloorWidget::setPoints (const int *index*, const int *na*, const int *nb*, double $*$ *cornerPoints*, const int *offset*) `[private]`

Calculates the corner points of a tile.

Parameters:

index The index of the tile whose corner points should be set.

na Index of one neighbor of the tile.

nb Index of another neighbor of the tile.

cornerpoints A pointer where the corner points of the tile should be stored.

offset Offset into the array of corner points.

D.8.3.29 void FloorWidget::setRefFrameTimecode (const QString & *refFrameTimecode*) `[slot]`

Sets the timecode of the reference frame.

Parameters:

refFrameTimecode

D.8.3.30 void FloorWidget::setRefFrameTimestamp (const QString & *refFrameTimestamp*) `[slot]`

Sets the timestamp of the reference frame.

Parameters:
refFrameTimestamp The timestamp of the reference frame.

D.8.3.31 void FloorWidget::setTileOutlines () `[private]`

Checks out which neighbors a tile has and makes a corresponding call to **setPoints**() (p. 71).

D.8.3.32 void FloorWidget::signalResizeHandler (QResizeEvent ∗ *re*)

Handels resize events.

Parameters:
re The catched resize event.

D.8.3.33 void FloorWidget::synchronize (const QString & *frameTimecode*) `[slot]`

Synchronizes the floor log with a given timecode of a video frame. Notice: Although a timecode is given, internally timestamps are used for synchronization.

Parameters:
frameTimecode The timecode of a video frame which is to synchronize with the log.

D.8.4 Member Data Documentation

D.8.4.1 QColor FloorWidget::_back_queue[MAX_BACK_QUEUE_-LENGTH][360] `[private]`

Data structure to store a certain amount of log entries. It is only necessary to store the color of each tile.

D.8.4.2 Scaler∗ FloorWidget::_floorScaler `[private]`

Object that handles the scaling of world points to image points and vise versa.

D.8.4.3 char FloorWidget::_ts_queue[MAX_BACK_QUEUE_LENGTH][20] `[private]`

A separate data structure to store timestamps of log entries.

The documentation for this class was generated from the following file:

- **floorwidget.h**

D.9 Frame Class Reference

Code for handling raw DV frame data.

```
#include <frame.h>
```

Public Methods

- **Frame** ()
- ∼**Frame** ()
- bool **GetSSYBPack** (int packNum, Pack &pack) const
- bool **GetVAUXPack** (int packNum, Pack &pack) const
- bool **GetAAUXPack** (int packNum, Pack &pack) const
- bool **GetTimeCode** (TimeCode &timeCode) const
- string **GetTimeCode** () const
- bool **GetRecordingDate** (struct tm &recDate) const
- string **GetRecordingDate** (void) const
- int **GetFrameSize** (void) const
- float **GetFrameRate** () const
- bool **IsPAL** (void) const
- bool **IsNewRecording** (void) const
- bool **IsComplete** (void) const
- void **ExtractHeader** (void)
- void **SetPreferredQuality** ()
- void **ExtractRGB** (void ∗rgb)
- int **ExtractPreviewRGB** (void ∗rgb)
- bool **IsWide** (void) const
- int **GetWidth** ()
- int **GetHeight** ()
- void **SetRecordingDate** (time_t ∗datetime, int frame)
- void **SetTimeCode** (int frame)
- void **Deinterlace** (void ∗image, int bpp)

Public Attributes

- unsigned char **data** [144000]
- int **bytesInFrame**
- dv_decoder_t ∗ **decoder**

D.9.1 Detailed Description

Code for handling raw DV frame data.

Class Frame contains methods for handling the raw DV frame **data** (p. 74). A PAL frame usually occupies 144000 bytes of data. Methods like **GetTimeCode**() (p. 74), **GetFrameRate**() (p. 74), **ExtractHeader**() (p. 74) or **ExtractRGB**() (p. 74) do special analysis of that data in one or the other way. The object therefore uses a digital video **decoder** (p. 75) provided by libdv-0.99 library (`http://sourceforge.net/projects/libdv/`).

Author:
Christoph Kiefer

Date:
6/6/2003

D.9.2 Constructor & Destructor Documentation

D.9.2.1 Frame::Frame ()

Constructor

D.9.3 Member Function Documentation

D.9.3.1 void Frame::ExtractHeader (void)

Extracts various frame header information as timecode or recording date. The information are directly decoded into **data** (p. 74).

D.9.3.2 void Frame::ExtractRGB (void ∗ *rgb*)

Extracts RGB values from frame **data** (p. 74).

Parameters:
rgb The buffer which afterward holds the extracted RGB values.

D.9.3.3 float Frame::GetFrameRate () const

Returns the actual frame rate. For PAL this is 25Hz, for NTSC about 30Hz.

Returns:
The frame rate.

D.9.3.4 string Frame::GetTimeCode () const

Returns the timecode of the frame.

Returns:
The String representing the timecode of the frame.

D.9.4 Member Data Documentation

D.9.4.1 unsigned char Frame::data[144000]

Enough space to hold a PAL frame.

D.9.4.2 dv_decoder_t∗ Frame::decoder

The decoder to decode the frame **data** (p. 74).

The documentation for this class was generated from the following file:

- **frame.h**

D.10 IEEE1394IOHandler Class Reference

Provides basic functionality to talk to the IEEE1394 interface.

```
#include <IEEE1394IOHandler.h>
```

Public Methods

- **IEEE1394IOHandler** (int channel, int card)
- ∼**IEEE1394IOHandler** ()
- virtual void **run** ()
- void **open_1394_driver** ()
- void **close_1394_driver** ()
- void **set_1394_iso_handler** (iso_handler_t handler)
- void **set_1394_reset_handler** (bus_reset_handler_t reset_handler)
- void **ask_for_raw1394_handle** ()
- raw1394handle_t **get_raw1394_handle** ()
- void **updateGeneration** (unsigned int generation)
- void **startPolling** ()
- void **stopPolling** ()
- bool **isPolling** ()
- void **write_raw1394** (raw1394handle_t handle, nodeid_t node, nodeaddr_t addr, size_t length, quadlet_t ∗data)

Private Attributes

- iso_handler_t **_iso_handler**
- bus_reset_handler_t **_reset_handler**
- bool **_handlerIsSet**
- int **_channel**
- int **_card**
- raw1394handle_t **_handle**
- bool **_polling**
- bool **_openDriver**
- pollfd **_pfd** [1]

D.10.1 Detailed Description

Provides basic functionality to talk to the IEEE1394 interface.

This class offers methods to talk to the IEEE1394 Interface. It therefore implements methods to open and close the interface, to set an ISO handler and to start and stop polling the interface. It makes use of Linux libraw1394-0.9.0 library. Due to performance reasons IEEE1394IOHandler inherits QThread to faster poll the FireWire interface.

Author:
Christoph Kiefer

Date:
6/6/2003

D.10.2 Constructor & Destructor Documentation

D.10.2.1 IEEE1394IOHandler::IEEE1394IOHandler (int *channel*, int *card*)

Constructor

Parameters:

channel The channel the connected device is using.

card The interface (port, card, bus) number.

D.10.2.2 IEEE1394IOHandler::∼IEEE1394IOHandler ()

Destructor

D.10.3 Member Function Documentation

D.10.3.1 void IEEE1394IOHandler::ask_for_raw1394_handle ()

Calls raw1394_new_handle() to get a handle that can control one IEEE1394 interface.

D.10.3.2 void IEEE1394IOHandler::close_1394_driver ()

Stops receiving from the ISO channel and destroys the **_handle** (p. 78)

D.10.3.3 raw1394handle_t IEEE1394IOHandler::get_raw1394_handle ()

Returns the **_handle** (p. 78) to the caller.

Returns:
The handle with the connection to the kernel side of raw1394.

D.10.3.4 bool IEEE1394IOHandler::isPolling ()

Returns `true` if thread is still polling the IEEE1394 interface, `false` otherwise.

Returns:
Status of **_polling** (p. 78).

D.10.3.5 void IEEE1394IOHandler::open_1394_driver ()

Uses various functions from libraw1394-0.9.0 library, for example raw1394_set_port(), raw1394_start_iso_rcv() or raw1394_set_iso_handler(). After this method returns the IEEE1394 interface is ready for transactions.

D.10.3.6 virtual void IEEE1394IOHandler::run () `[virtual]`

This virtual method is inherited from QThread. As long as **_polling** (p. 78) is `true` the created thread polls the IEEE1394 interface by repeatedly calling raw1394_loop_iterate().

D.10.3.7 void IEEE1394IOHandler::set_1394_iso_handler (iso_handler_t *handler*)

Set the **_iso_handler** (p. 78) that will be called when an iso packet arrives.

Parameters:
handler AVI ISO handler

D.10.3.8 void IEEE1394IOHandler::set_1394_reset_handler (bus_reset_handler_t *reset_handler*)

Set the **_reset_handler** (p. 78) that will be called when a bus reset message is encountered.

Parameters:
reset_handler Bus reset handler.

D.10.3.9 void IEEE1394IOHandler::startPolling ()

If necessary calls **open_1394_driver**() (p. 77). Calls start() to initiate thread execution.

D.10.3.10 void IEEE1394IOHandler::stopPolling ()

Sets **_polling** (p. 78) to `false`.

D.10.3.11 void IEEE1394IOHandler::updateGeneration (unsigned int *generation*)

Updates the generation number of the port.

Parameters:
handle Structure providing a connection to the kernel side of raw1394.

generation The new generation number to use for the handle's port.

D.10.3.12 void IEEE1394IOHandler::write_raw1394 (raw1394handle_t *handle*, nodeid_t *node*, nodeaddr_t *addr*, size_t *length*, quadlet_t ∗ *data*)

Calls same function from libraw1394-0.9.0 library which does the complete transaction of the data. It will call raw1394_loop_iterate() as often as necessary.

Parameters:

handle Structure providing a connection to the kernel side of raw1394.

node Physical device id of DV camera. Default is 1.

addr FCP Register address.

length Length of command (4 byte or one quadlet).

data The command to send to the camera/

D.10.4 Member Data Documentation

D.10.4.1 raw1394handle_t IEEE1394IOHandler::_handle `[private]`

Structure providing a connection to the kernel side of raw1394.

D.10.4.2 iso_handler_t IEEE1394IOHandler::_iso_handler `[private]`

The handler to process the packets. It is called repeatedly by raw1394_loop_iterate().

D.10.4.3 struct pollfd IEEE1394IOHandler::_pfd[1] `[private]`

Structure to store the file descriptor of the **_handle** (p. 78). Used to poll the descriptor to see if packets are available.

D.10.4.4 bool IEEE1394IOHandler::_polling `[private]`

If `true` thread is still polling the IEEE1394 interface.

D.10.4.5 bus_reset_handler_t IEEE1394IOHandler::_reset_handler `[private]`

Handler that is called whenever a bus reset occurs (optional).

The documentation for this class was generated from the following file:

- **IEEE1394IOHandler.h**

D.11 imagePoint Struct Reference

Struct to hold an image point.

```
#include <points.h>
```

Public Attributes

- int **x**
- int **y**
- int **number**

D.11.1 Detailed Description

Struct to hold an image point.

D.11.2 Member Data Documentation

D.11.2.1 int imagePoint::number

Number of image point.

D.11.2.2 int imagePoint::x

x-coordinate of image point.

D.11.2.3 int imagePoint::y

y-coordinate of image point.

The documentation for this struct was generated from the following file:

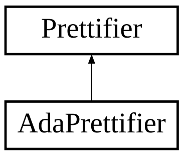
- **points.h**

D.12 Prettifier Class Reference

Abstract class to read image data from an image buffer in a user defined way and to store the image data in a QImage.

`#include <prettifier.h>`

Inheritance diagram for Prettifier::



Public Methods

- **Prettifier** ()
- virtual void **rearrange** (QImage &image, unsigned char ∗data_buffer)=0

D.12.1 Detailed Description

Abstract class to read image data from an image buffer in a user defined way and to store the image data in a QImage.

Author:
Christoph Kiefer

Date:
6/6/2003

D.12.2 Constructor & Destructor Documentation

D.12.2.1 Prettifier::Prettifier ()

Constructor

D.12.3 Member Function Documentation

D.12.3.1 virtual void Prettifier::rearrange (QImage & *image*, unsigned char ∗ *data_buffer*) `[pure virtual]`

A derived class can overload this virtual method to define how to extract the image data from the image buffer. Method setPixel() of QImage may be used to store the image data at the proper position in the image.

Implemented in **AdaPrettifier** (p. 46).

The documentation for this class was generated from the following file:

- **prettifier.h**

D.13 Scaler Class Reference

Class to scale points between the world and an image (or widget).

```
#include <scaler.h>
```

Public Methods

- **Scaler** (double width, double height, double minX, double maxY)
- void **setScaleFactor** (double widgetWidth, double widgetHeight)
- double **getScaleFactor** ()
- virtual int **wx2ix** (double wx)
- virtual int **wy2iy** (double wy)
- virtual double **ix2wx** (int ix)
- virtual double **iy2wy** (int iy)

Private Attributes

- double **_width**
- double **_height**
- double **_ratio**
- double **_scaleFactor**
- double **_minX**
- double **_maxY**

D.13.1 Detailed Description

Class to scale points between the world and an image (or widget).

The class scales a world point (maybe in mm) to an image point (in pixels) and vise versa. It therefore uses a scale factor that is calculated in method **setScaleFactor**() (p. 82).

Author:
Christoph Kiefer

Date:
6/6/2003

D.13.2 Constructor & Destructor Documentation

D.13.2.1 Scaler::Scaler (double *width*, double *height*, double *minX*, double *maxY*)

Constructor Computes the ratio of given width and height. This ratio stays unchanged.

Parameters:
width The width of a constant distance, for instance the width of the Ada/Expo.02 floor.
height The height of a constant distance.
minX The minimum x-coordinate.
maxY The maximum y-coordinate.

D.13.3 Member Function Documentation

D.13.3.1 double Scaler::getScaleFactor ()

Returns the scale factor.

Returns:
The current scale factor.

D.13.3.2 virtual double Scaler::ix2wx (int *ix*) `[virtual]`

Virtual method than can be overloaded in a derived class. It computes the world x-coordinate of an x-coordinate in an image.

Parameters:
ix x-coordinate in an image (in pixels).

D.13.3.3 virtual double Scaler::iy2wy (int *iy*) `[virtual]`

Virtual method than can be overloaded in a derived class. It computes the world y-coordinate of an y-coordinate in an image.

Parameters:
iy y-coordinate in an image (in pixels).

D.13.3.4 void Scaler::setScaleFactor (double *widgetWidth*, double *widgetHeight*)

Sets the scale factor according to a given width and height of an image or widget.

Parameters:
widgetWidth The width of the widget where a point should be scaled in.
widgetHeight The height of the widget where a point should be scaled in.

D.13.3.5 virtual int Scaler::wx2ix (double *wx*) `[virtual]`

Virtual method than can be overloaded in a derived class. It computes the image x-coordinate of an x-coordinate in the world.

Parameters:
wx x-coordinate in the world (in mm).

D.13.3.6 virtual int Scaler::wy2iy (double *wy*) `[virtual]`

Virtual method than can be overloaded in a derived class. It computes the image y-coordinate of an y-coordinate in the world.

Parameters:
wy y-coordinate in the world (in mm).

The documentation for this class was generated from the following file:

- **scaler.h**

D.14 Tile Class Reference

Class representing one hexagonal floor tile at Ada/Expo.02.

```
#include <tile.h>
```

Public Methods

- **Tile** ()
- **Tile** (double xCenter, double yCenter, const int rh, const int ru, const int lu, const int lh, const int ld, const int rd)
- int **getNeighbour** (const neighbours n) const
- double **getXCenterWorld** ()
- double **getYCenterWorld** ()

Public Attributes

- double ∗ **_cornerPointsWorld**
- QColor **color**

Private Attributes

- double **_xCenterWorld**
- double **_yCenterWorld**
- int **_rh**
- int **_ru**
- int **_lu**
- int **_lh**
- int **_ld**
- int **_rd**

D.14.1 Detailed Description

Class representing one hexagonal floor tile at Ada/Expo.02.

The floor at Ada/Expo.02 was consisting of 360 individual hexagonal floor tiles which were able to change its colors. In that way the floor could produce some kind of patterns to interact with visitors at the exhibition.

Author:
Christoph Kiefer

Date:
6/6/2003

D.14.2 Constructor & Destructor Documentation

D.14.2.1 Tile::Tile ()

Constructor

D.14.2.2 Tile::Tile (double *xCenter*, double *yCenter*, const int *rh*, const int *ru*, const int *lu*, const int *lh*, const int *ld*, const int *rd*)

Constructor

Parameters:

xCenter	x-coordinate of the center of tile inside the room.
yCenter	y-coordinate of the center of tile inside the room.
rh	Horizontal right neighbor tile. -1 if no neighbor is present, otherwise its index.
ru	Upper right neighbor tile. -1 if no neighbor is present, otherwise its index.
lu	Upper left neighbor tile. -1 if no neighbor is present, otherwise its index.
lh	Horizontal left neighbor tile. -1 if no neighbor is present, otherwise its index.
ld	Lower left neighbor tile. -1 if no neighbor is present, otherwise its index.
rd	Lower right neighbor tile. -1 if no neighbor is present, otherwise its index.

D.14.3 Member Function Documentation

D.14.3.1 int Tile::getNeighbour (const neighbours *n*) const

Tells caller if tile has neighbor 'n'.

Returns:
1 if caller has neighbor n, -1 otherwise.

D.14.3.2 double Tile::getXCenterWorld ()

Returns x-coordinates of center to the caller.

Returns:
The x-coordinate of the center.

D.14.3.3 double Tile::getYCenterWorld ()

Returns y-coordinates of center to the caller.

Returns:
The y-coordinate of the center.

The documentation for this class was generated from the following file:

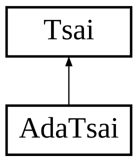
- **tile.h**

D.15 Tsai Class Reference

Abstract wrapper around class TsaiCal.

```
#include <tsai.h>
```

Inheritance diagram for Tsai::



Public Slots

- virtual void **startCalibration** ()=0
- virtual void **createCalibrationFile** (int fileNumber, QPtrList< **imagePoint** > ∗i-Points, QPtrList< **worldPoint** > ∗wPoints)=0
- virtual void **convert** (double wWorld, double hWorld, int w, int h)=0

Public Methods

- **Tsai** (int tsaiInstances, QImage &oldImage, QImage &tsaiImage)

Protected Attributes

- ofstream **_camCDStream**
- QPtrList< TsaiCal > ∗ **_tsai**
- QImage **_oldImage**
- QImage **_tsaiImage**

D.15.1 Detailed Description

Abstract wrapper around class TsaiCal.

The class is actually a wrapper around class TsaiCal but augmented with some sugar to separate the calibration from the main application.

Author:
Christoph Kiefer

Date:
6/6/2003

D.15.2 Constructor & Destructor Documentation

D.15.2.1 Tsai::Tsai (int *tsaiInstances*, QImage & *oldImage*, QImage & *tsaiImage*)

Constructor

Parameters:

tsaiInstances Number of instances of class TsaiCal.

oldImage Image out of which a unified view should be computed.

tsaiImage The image which will hold the unified view.

D.15.3 Member Function Documentation

D.15.3.1 virtual void Tsai::convert (double *wWorld*, double *hWorld*, int *w*, int *h*) `[pure virtual, slot]`

Pure virtual function that must be implemented by a derived class. Computes the final unified view.

Parameters:

wWorld x-coordinate in the world (in mm) of point w

hWorld y-coordinate in in the world (in mm) of point h

w x-coordinate in the image (that image that hold the unified view), in pixels

h y-coordinate in the image (that image that hold the unified view), in pixels

Implemented in **AdaTsai** (p. 47).

D.15.3.2 virtual void Tsai::createCalibrationFile (int *fileNumber*, QPtrList< imagePoint > ∗ *iPoints*, QPtrList< worldPoint > ∗ *wPoints*) `[pure virtual, slot]`

Pure virtual function that must be implemented by a derived class. Writes the calibration files.

Parameters:

fileNumber Number or index of the calibration files that get written.

iPoints A list of image points.

wPoints A list of world points.

Implemented in **AdaTsai** (p. 48).

D.15.3.3 virtual void Tsai::startCalibration () `[pure virtual, slot]`

Pure virtual function that must be implemented by a derived class. Steps that needs to be done before the calibration may start should be implemented in this method.

Implemented in **AdaTsai** (p. 48).

The documentation for this class was generated from the following file:

- **tsai.h**

D.16 TsaiViewer Class Reference

The TsaiViewer class displays the unified view.

```
#include <tsaiviewer.h>
```

Public Slots

- void **setImage** (QImage image)

Public Methods

- **TsaiViewer** (QWidget ∗parent=0, const char ∗name=0, WFlags f=0, **FloorWidget** ∗floorWidget=0, **Scaler** ∗scaler=0)

Protected Methods

- void **polishContent** (QPainter &p)
- void **wheelEvent** (QWheelEvent ∗we)
- void **mousePressEvent** (QMouseEvent ∗me)
- void **mouseReleaseEvent** (QMouseEvent ∗me)
- void **mouseMoveEvent** (QMouseEvent ∗me)

Private Slots

- void **showGrid** ()

Private Attributes

- **FloorWidget** ∗ **_floorWidget**
- **Scaler** ∗ **_scaler**
- QPixmap ∗ **_buffer**
- QBitmap **_mask**
- double **_width**
- double **_height**
- bool **_mousePressed**
- int **_xPos**
- int **_yPos**
- QImage **_image**

D.16.1 Detailed Description

The TsaiViewer class displays the unified view.

This class displays the unified view in a separate Qt widget. By clicking the mouse, the floor grid is projected over the unified view. The grid can be moved around on the unified view. The floor grid can also be enlarged or scaled down by turning the mouse wheel.

Author:
Christoph Kiefer

Date:
3/7/2003

D.16.2 Constructor & Destructor Documentation

D.16.2.1 TsaiViewer::TsaiViewer (QWidget ∗ *parent* = 0, const char ∗ *name* = 0, WFlags *f* = 0, FloorWidget ∗ *floorWidget* = 0, Scaler ∗ *scaler* = 0)

Constructor.

Parameters:

parent Pointer to parent widget. Zero (default) for top-level widgets.

name Identifier internally used by Qt for identifying individual class instances.

f Widget-specific flags.

floorWidget Holds information about floor tiles. Offers methods to handle them.

scaler Scales points to fit in this widget.

D.16.3 Member Function Documentation

D.16.3.1 void TsaiViewer::mouseMoveEvent (QMouseEvent ∗ *me*) `[protected]`

Catch mouse move events. The grid is moved as long as the mouse is pressed down and moved on the image.

Parameters:
Mouse event.

D.16.3.2 void TsaiViewer::mousePressEvent (QMouseEvent ∗ *me*) `[protected]`

Catch mouse press events. If mouse is pressed down, the grid can be moved.

Parameters:
me Mouse event.

D.16.3.3 void TsaiViewer::mouseReleaseEvent (QMouseEvent ∗ *me*) `[protected]`

Catch mouse release events. If mouse is released the grid cannot be moved around on the displayed anymore. If left mouse button is pressed, the grid will be displayed. Pressing the right mouse button removes the grid from the image.

Parameters:
me Mouse Event.

D.16.3.4 void TsaiViewer::polishContent (QPainter & *p*) `[protected]`

Enhance image content before displaying it.

Parameters:
p Painter for drawing into the given image.

D.16.3.5 void TsaiViewer::setImage (QImage *image*) `[slot]`

Sets new image. Stores image width and height locally before calling Viewer::set-Image().

Parameters:
image New image to be displayed.

D.16.3.6 void TsaiViewer::showGrid () `[private, slot]`

Paints the floor grid onto the unified view.

D.16.3.7 void TsaiViewer::wheelEvent (QWheelEvent ∗ *we*) `[protected]`

Catch mouse wheel events. The displayed grid is either enlarged or scaled down.

Parameters:
we Wheel Event.

The documentation for this class was generated from the following file:

- **tsaiviewer.h**

D.17 VideoEncoder Class Reference

Class to encode QImages into a movie stream.

```
#include <videoencoder.h>
```

Public Slots

- void **openVideoStream** ()
- void **closeVideoStream** ()
- void **encode** (QImage image)

Public Methods

- **VideoEncoder** (CodecID id, const char ∗of, int width, int height, const char ∗filename)
- ∼**VideoEncoder** ()

Private Slots

- int **avpicture_alloc** (AVPicture ∗picture, int pix_fmt, int width, int height)
- void **avpicture_free** (AVPicture ∗picture)

Private Attributes

- AVCodecContext ∗ **codecContext**
- AVCodec ∗ **codec**
- AVPicture ∗ **yuvImage**
- AVPicture ∗ **rgbImage**
- AVFrame ∗ **frame**
- ofstream **movie**
- unsigned char ∗ **outbuf**
- unsigned char ∗ **yuvBuffer**
- unsigned char ∗ **rgbBuffer**
- int **pos**
- int **outSize**
- int **_width**
- int **_height**
- bool **_codecIsOpen**
- const char ∗ **_filename**

D.17.1 Detailed Description

Class to encode QImages into a movie stream.

The class encapsulates functionality from libavcodec library that also belongs to FFMPEG Multimedia System (see `http://ffmpeg.sourceforge.net/`). A video encoder gets initialized. A QImage is converted from RGB24 to YUV420 before it is encoded and written to the video file.

Author:
Christoph Kiefer

Date:
30/6/2003

D.17.2 Constructor & Destructor Documentation

D.17.2.1 VideoEncoder::VideoEncoder (CodecID *id*, const char ∗ *of*, int *width*, int *height*, const char ∗ *filename*)

Constructor.

Parameters:
id ID of the video encoder.

an of Name of the output format, e.g. "avi".

Parameters:
width Width of output frame.
height Height of output frame.
filename Name of the output video file.

D.17.2.2 VideoEncoder::∼VideoEncoder ()

Destructor.

D.17.3 Member Function Documentation

D.17.3.1 int VideoEncoder::avpicture_alloc (AVPicture ∗ *picture*, int *pix_fmt*, int *width*, int *height*) `[private, slot]`

Allocates an new AVPicture. Is currently not used.

Parameters:
picture Pointer to an AVPicture whose data is to be allocated.
pix_fmt Pixel format (e.g. PIX_FMT_RGB24).
width Width of picture.
height Height of picture.

Returns:
0 if successfully allocated, otherwise -1.

D.17.3.2 void VideoEncoder::avpicture_free (AVPicture ∗ *picture*) `[private, slot]`

Frees resources of an AVPicture.Currently not used.

Parameters:

picture Pointer to an AVPicture whose data is to be freed.

D.17.3.3 void VideoEncoder::closeVideoStream () `[slot]`

Writes trailer data to video file and closes it. Frees allocated resources.

D.17.3.4 void VideoEncoder::encode (QImage *image*) `[slot]`

The QImage is converted first from RGB24 to YUV420. The new image is then encoded using the video encoder. The the encoded image is written to the video file.

D.17.3.5 void VideoEncoder::openVideoStream () `[slot]`

Opens the video codec and the output video file.

The documentation for this class was generated from the following file:

- **videoencoder.h**

D.18 worldPoint Struct Reference

Struct to hold a world point.

```
#include <points.h>
```

Public Attributes

- int **tile**
- int **corner**
- int **number**
- double **xw**
- double **yw**

D.18.1 Detailed Description

Struct to hold a world point.

D.18.2 Member Data Documentation

D.18.2.1 int worldPoint::corner

Number of the corner of the tile the point designates.

D.18.2.2 int worldPoint::number

Number of world point.

D.18.2.3 int worldPoint::tile

Tile (p. 82) number the point lies on.

D.18.2.4 double worldPoint::xw

x-coordinate of world point.

D.18.2.5 double worldPoint::yw

y-coordinate of world point.

The documentation for this struct was generated from the following file:

- **points.h**

D.19 IEEE1394AVC.h File Reference

Defines AV/C commands that can be be sent to the DV camera.

Defines

- #define **FCP_COMMAND_ADDR** 0xFFFFF0000B00

  Address of function control protocol register. AV/C commands are written to this register.

- #define **FCP_RESPONSE_ADDR** 0xFFFFF0000D00
- #define **AVC_CTYPE_CONTROL** 0x00000000

  Type of command.

- #define **AVC_CTYPE_STATUS** 0x01000000
- #define **AVC_CTYPE_SPECIFIC_INQUIRY** 0x02000000
- #define **AVC_CTYPE_NOTIFY** 0x03000000
- #define **AVC_CTYPE_GENERAL_INQUIRY** 0x04000000
- #define **AVC_RESPONSE_NOT_IMPLEMENTED** 0x08000000
- #define **AVC_RESPONSE_ACCEPTED** 0x09000000
- #define **AVC_RESPONSE_REJECTED** 0x0A000000
- #define **AVC_RESPONSE_IN_TRANSITION** 0x0B000000
- #define **AVC_RESPONSE_IMPLEMENTED** 0x0C000000
- #define **AVC_RESPONSE_STABLE** 0x0C000000
- #define **AVC_RESPONSE_CHANGED** 0x0D000000
- #define **AVC_RESPONSE_INTERIM** 0x0F000000
- #define **AVC_SUBUNIT_TYPE_VIDEO_MONITOR** (0 << 19)
- #define **AVC_SUBUNIT_TYPE_DISC_RECORDER** (3 << 19)
- #define **AVC_SUBUNIT_TYPE_TAPE_RECORDER** (4 << 19)

The device that is connected to the IEEE1394 bus.

- #define **AVC_SUBUNIT_TYPE_TUNER** (5 << 19)
- #define **AVC_SUBUNIT_TYPE_VIDEO_CAMERA** (7 << 19)
- #define **AVC_SUBUNIT_TYPE_VENDOR_UNIQUE** (0x1C << 19)
- #define **AVC_SUBUNIT_ID_0** (0 << 16)
- #define **AVC_SUBUNIT_ID_1** (1 << 16)
- #define **AVC_SUBUNIT_ID_2** (2 << 16)
- #define **AVC_SUBUNIT_ID_3** (3 << 16)
- #define **AVC_SUBUNIT_ID_4** (4 << 16)
- #define **AVC_COMMAND_CHANNEL_USAGE** 0x00001200
- #define **AVC_COMMAND_CONNECT** 0x00002400
- #define **AVC_COMMAND_CONNECT_AV** 0x00002000
- #define **AVC_COMMAND_CONNECTIONS** 0x00002200
- #define **AVC_COMMAND_DIGITAL_INPUT** 0x00001100
- #define **AVC_COMMAND_DIGITAL_OUTPUT** 0x00001000
- #define **AVC_COMMAND_DISCONNECT** 0x00002500
- #define **AVC_COMMAND_DISCONNECT_AV** 0x00002100
- #define **AVC_COMMAND_INPUT_PLUG_SIGNAL_FORMAT** 0x00001900
- #define **AVC_COMMAND_OUTPUT_PLUG_SIGNAL_FORMAT** 0x00001800
- #define **AVC_COMMAND_SUBUNIT_INFO** 0x00003100
- #define **AVC_COMMAND_UNIT_INFO** 0x00003000
- #define **AVC_COMMAND_OPEN_DESCRIPTOR** 0x00000800
- #define **AVC_COMMAND_READ_DESCRIPTOR** 0x00000900
- #define **AVC_COMMAND_WRITE_DESCRIPTOR** 0x00000A00
- #define **AVC_COMMAND_SEARCH_DESCRIPTOR** 0x00000B00
- #define **AVC_COMMAND_OBJECT_NUMBER_SELECT** 0x00000D00
- #define **AVC_COMMAND_POWER** 0x0000B200
- #define **AVC_COMMAND_RESERVE** 0x00000100
- #define **AVC_COMMAND_PLUG_INFO** 0x00000200
- #define **AVC_COMMAND_VENDOR_DEPENDENT** 0x00000000
- #define **AVC_OPERAND_DESCRIPTOR_TYPE_SUBUNIT_IDENTIFIER_-DESCRIPTOR** 0x00
- #define **AVC_OPERAND_DESCRIPTOR_TYPE_OBJECT_LIST_-DESCRIPTOR_ID** 0x10
- #define **AVC_OPERAND_DESCRIPTOR_TYPE_OBJECT_LIST_-DESCRIPTOR_TYPE** 0x11
- #define **AVC_OPERAND_DESCRIPTOR_TYPE_OBJECT_ENTRY_-DESCRIPTOR_POSITION** 0x20
- #define **AVC_OPERAND_DESCRIPTOR_TYPE_OBJECT_ENTRY_-DESCRIPTOR_ID** 0x21
- #define **AVC_OPERAND_DESCRIPTOR_SUBFUNCTION_CLOSE** 0x00
- #define **AVC_OPERAND_DESCRIPTOR_SUBFUNCTION_READ_OPEN** 0x01
- #define **AVC_OPERAND_DESCRIPTOR_SUBFUNCTION_WRITE_OPEN** 0x03
- #define **VCR_COMMAND_ANALOG_AUDIO_OUTPUT_MODE** 0x000007000
- #define **VCR_COMMAND_AREA_MODE** 0x000007200
- #define **VCR_COMMAND_ABSOLUTE_TRACK_NUMBER** 0x000005200
- #define **VCR_COMMAND_AUDIO_MODE** 0x000007100

- #define **VCR_COMMAND_BACKWARD** 0x000005600
- #define **VCR_COMMAND_BINARY_GROUP** 0x000005A00
- #define **VCR_COMMAND_EDIT_MODE** 0x000004000
- #define **VCR_COMMAND_FORWARD** 0x000005500
- #define **VCR_COMMAND_INPUT_SIGNAL_MODE** 0x000007900
- #define **VCR_COMMAND_LOAD_MEDIUM** 0x00000C100
- #define **VCR_COMMAND_MARKER** 0x00000CA00
- #define **VCR_COMMAND_MEDIUM_INFO** 0x00000DA00
- #define **VCR_COMMAND_OPEN_MIC** 0x000006000
- #define **VCR_COMMAND_OUTPUT_SIGNAL_MODE** 0x000007800
- #define **VCR_COMMAND_PLAY** 0x00000C300

  The command to start playing the DV camera.

- #define **VCR_COMMAND_PRESET** 0x000004500
- #define **VCR_COMMAND_READ_MIC** 0x000006100
- #define **VCR_COMMAND_RECORD** 0x00000C200
- #define **VCR_COMMAND_RECORDING_DATE** 0x000005300
- #define **VCR_COMMAND_RECORDING_SPEED** 0x00000DB00
- #define **VCR_COMMAND_RECORDING_TIME** 0x000005400
- #define **VCR_COMMAND_RELATIVE_TIME_COUNTER** 0x000005700
- #define **VCR_COMMAND_SEARCH_MODE** 0x000005000
- #define **VCR_COMMAND_SMPTE_EBU_RECORDING_TIME** 0x000005C00
- #define **VCR_COMMAND_SMPTE_EBU_TIME_CODE** 0x000005900
- #define **VCR_COMMAND_TAPE_PLAYBACK_FORMAT** 0x00000D300
- #define **VCR_COMMAND_TAPE_RECORDING_FORMAT** 0x00000D200
- #define **VCR_COMMAND_TIME_CODE** 0x000005100
- #define **VCR_COMMAND_TRANSPORT_STATE** 0x00000D000
- #define **VCR_COMMAND_WIND** 0x00000C400
- #define **VCR_COMMAND_WRITE_MIC** 0x000006200
- #define **VCR_OPERAND_LOAD_MEDIUM_EJECT** 0x60
- #define **VCR_OPERAND_LOAD_MEDIUM_OPEN_TRAY** 0x31
- #define **VCR_OPERAND_LOAD_MEDIUM_CLOSE_TRAY** 0x32
- #define **VCR_OPERAND_PLAY_NEXT_FRAME** 0x30
- #define **VCR_OPERAND_PLAY_SLOWEST_FORWARD** 0x31
- #define **VCR_OPERAND_PLAY_FAST_FORWARD_1** 0x39
- #define **VCR_OPERAND_PLAY_FAST_FORWARD_2** 0x3A
- #define **VCR_OPERAND_PLAY_FAST_FORWARD_3** 0x3B
- #define **VCR_OPERAND_PLAY_FAST_FORWARD_4** 0x3C
- #define **VCR_OPERAND_PLAY_FAST_FORWARD_5** 0x3D
- #define **VCR_OPERAND_PLAY_FAST_FORWARD_6** 0x3E
- #define **VCR_OPERAND_PLAY_FASTEST_FORWARD** 0x3F
- #define **VCR_OPERAND_PLAY_PREVIOUS_FRAME** 0x40
- #define **VCR_OPERAND_PLAY_SLOWEST_REVERSE** 0x41
- #define **VCR_OPERAND_PLAY_FAST_REVERSE_1** 0x49
- #define **VCR_OPERAND_PLAY_FAST_REVERSE_2** 0x4A
- #define **VCR_OPERAND_PLAY_FAST_REVERSE_3** 0x4B
- #define **VCR_OPERAND_PLAY_FAST_REVERSE_4** 0x4C
- #define **VCR_OPERAND_PLAY_FAST_REVERSE_5** 0x4D
- #define **VCR_OPERAND_PLAY_FAST_REVERSE_6** 0x4E
- #define **VCR_OPERAND_PLAY_FASTEST_REVERSE** 0x4F

- #define **VCR_OPERAND_PLAY_FORWARD** 0x75

  The operand of the command play.

- #define **VCR_OPERAND_PLAY_FORWARD_PAUSE** 0x7D
- #define **VCR_OPERAND_RECORD_RECORD** 0x75
- #define **VCR_OPERAND_RECORD_PAUSE** 0x7D
- #define **VCR_OPERAND_TRANSPORT_STATE** 0x7F
- #define **VCR_RESPONSE_TRANSPORT_STATE_LOAD_MEDIUM** 0x0000C100
- #define **VCR_RESPONSE_TRANSPORT_STATE_RECORD** 0x0000C200
- #define **VCR_RESPONSE_TRANSPORT_STATE_PLAY** 0x0000C300
- #define **VCR_RESPONSE_TRANSPORT_STATE_WIND** 0x0000C400
- #define **VCR_OPERAND_WIND_HIGH_SPEED_REWIND** 0x45
- #define **VCR_OPERAND_WIND_STOP** 0x60
- #define **VCR_OPERAND_WIND_REWIND** 0x65
- #define **VCR_OPERAND_WIND_FAST_FORWARD** 0x75
- #define **VCR_OPERAND_RELATIVE_TIME_COUNTER_CONTROL** 0x20
- #define **VCR_OPERAND_RELATIVE_TIME_COUNTER_STATUS** 0x71
- #define **VCR_OPERAND_TIME_CODE_CONTROL** 0x20
- #define **VCR_OPERAND_TIME_CODE_STATUS** 0x71
- #define **VCR_OPERAND_TRANSPORT_STATE** 0x7F
- #define **VCR_OPERAND_RECORDING_TIME_STATUS** 0x71
- #define **TUNER_COMMAND_DIRECT_SELECT_INFORMATION_TYPE** 0xC8
- #define **TUNER_COMMAND_DIRECT_SELECT_DATA** 0xCB
- #define **TUNER_COMMAND_CA_ENABLE** 0xCC
- #define **TUNER_COMMAND_TUNER_STATUS** 0xCD
- #define **TUNER_COMMAND_DIRECT_SELECT_INFORMATION_TYPE** 0xC8
- #define **TUNER_COMMAND_DIRECT_SELECT_DATA** 0xCB
- #define **TUNER_COMMAND_CA_ENABLE** 0xCC
- #define **TUNER_COMMAND_TUNER_STATUS** 0xCD
- #define **CTLVCR0** AVC_CTYPE_CONTROL | AVC_SUBUNIT_TYPE_TAPE_-RECORDER | AVC_SUBUNIT_ID_0

  Simplification because there will always be sent the same type of commands to the same device.

D.19.1 Detailed Description

Defines AV/C commands that can be be sent to the DV camera.

Author:
Christoph Kiefer

Date:
6/6/2003

D.20 points.h File Reference

Defines world and image points.

Compounds

- struct **imagePoint**

  Struct to hold an image point.

- struct **worldPoint**

  Struct to hold a world point.

D.20.1 Detailed Description

Defines world and image points.

The file defines two structs. One for holding image points and another struct for world points. World points are those used in class **FloorWidget** (p. 65) whereas image points are used in class **AdaViewer** (p. 48).

Author:
Christoph Kiefer

Date:
6/6/2003

Appendix E

Project Description

E.1 Introduction

ETH Zurich participated in the Swiss national exhibition Expo.02 with the *Ada* project, an artificial life form developed by the Institute for Neuroinformatics. This life form is able to interact with its visitors through a room equipped with sensors and actuators.

One of these sensor types is the so-called *Vision Matrix*, a grid of originally 8 ceiling-mounted black&white-cameras. In August 2002, the black&white-cameras have been replaced by 4 to 6 pan/tilt color cameras. The Vision Matrix's main purpose is the observation of the visitors from a bird's eye perspective. The video data captured from the Vision Matrix allows tracking of the visitors while they are interacting with Ada. This vision based approach was originally thought as ground truth for the tracking results obtained by the pressure sensitive floor tiles. However, from PCCV's perspective, vision will be the major issue and the floor tiles used as ground truth.

E.2 Task Description

The aim of this semester thesis is the development of a method for the semi-automatic calibration of the Vision Matrix. The calibration of the four cameras is one major precondition for further applications of vision based tracking algorithms. One prototype[25] was allready developed. The task in this semester thesis is to automate the process of the calibration as far as possible. The problem consists of the following four major steps:

1. Reading in video data via *IEEE 1394* aka "FireWire". The video data is stored on Mini DV tapes. A first step should be to read in this data via the IEEE 1394 bus using existing API's. Existing examples concerning this task are applications like `dvcont` or `dvgrab`.

2. Reading in and visualizing the log files of the Ada floor. The already developed prototype can be used to complete this step.

3. Calibrating the video data. The four individual video streams have to be projected into one world coordinate system to produce one *unified view*. The world coordinate system is defined by the floor topology. "Tsai's method"[24] should be used to further define the necessary parameters of the calibration. This method needs pairs of points which consist of a world point and a corresponding image point. To choose

the right pairs of points, the log files of the floor and the video stream have to be synchronized. The process of synchronization should be automatic whenever possible and is achieved by comparing timestamps.

4. Writing out the resulting unified view as MPEG-2 video.

The following genereal conditions are given aside these textual requirements:

1. Class reuseability. The design of the main interfaces of the components should be aligned with the supervisor's work to ensure code reusability.
2. Runtime environment is Linux (>= Kernel 2.4.18) and the GUI Toolkit `Qt`[27].

E.3 Requirements by PCCV

in accordance with the "guidelines for semester and diploma theses with PCCV", the group for *Perceptual Computing and Computer Vision* demands the following points fulfilled for successfully accomplish a semester thesis.

- Meetings with the advisor on a regular base. These meetings are thought to discuss problems, to review intermediate results and to plan the next steps of the thesis.
- Fully functional programs which comply with the requirements stated above.
- Written documentation of the programs as well as source code documentation are part of the task and will be evaluated in the end of the thesis. The written documentation should primarily describe the concept and design of the implementation, explain the applied algorithms and provide a manual for the implemented software. The source code documentation is thought as a help for re-using the code and thus focus on the description of interfaces.
- Two bound copies and one loose copy of the written documentation. The written part of a semester thesis should have an extent of 30 to 60 pages. All printed parts of the thesis have to be made with LaTeX.
- An oral presentation has to be given at the end of the thesis. The duration of the presentation should not exceed 20 minutes. The focus of the presentation is the description of the student's own work, e.g. developed algorithms, implemented software, etc.
- A CD-ROM that contains all files required to rebuild the software as well as the documentation. In addition, a PDF version of the documentation and – if possible – a compiled and executable version of the software should also be on the CD.

Advisor
Martin Spengler
IFW B26.2
`spengler@inf.ethz.ch`
Tel. 632 09 64

Bibliography

[1] Kynan Eng et al. *Ada: Buildings as Organisms.* `www.ini.unizh.ch/~tobi/papers/ada-gamesetandmatch.pdf`.

[2] *IEEE 1394 for Linux.* `http://www.linux1394.org/`.

[3] *Audio/Video Control (AV/C).* `http://www.ict.tuwien.ac.at/ieee1394/avc/avc-en.htm#%5BAVC%2099a%5D/`.

[4] Andreas Bombe. *Libraw1394 Short Documentation.* `http://www.linux1394.org/doc/libraw1394/book1.html`, 2001.

[5] *Libraw1394 Project.* `http://sourceforge.net/projects/libraw1394/`.

[6] *Quasar DV Codec Project.* `http://sourceforge.net/projects/libdv/`.

[7] Charles Krasic and Erik Walthinsen. *Quasar DV Codec: libdv.* `http://sourceforge.net/projects/libdv/`, 2001.

[8] *Quicktime for Linux.* `http://heroinewarrior.com/quicktime.php3`.

[9] Arne Schirmacher. *Digital Video for Linux.* `http://kino.schirmacher.de/`.

[10] *GNU/Linux 1394 AV/C Library.* `http://sourceforge.net/projects/libavc1394/.`

[11] *1394-based DC Control Library.* `http://sourceforge.net/projects/libdc1394/`.

[12] *IEEE 1394 Standart.* `http://www-ivs.cs.uni-magdeburg.de/bs/lehre/wise9900/proro/vortrag/ieee/IEEE.htm`.

[13] *IEEE Organisation.* `http://www.ieee.org/portal/index.jsp`.

[14] Leslie Shapiro *Firewire - The Consumer Electronics Connection.* `http://www.extremetech.com/article2/0,3973,87693,00.asp`, 2002.

[15] *IEEE 1394 (AKA “FireWire” & “iLink”).* `https://secure2.vivid-design.com.au/jaycar/images_uploaded/firewire.pdf`, 2002.

[16] Soren Thing Andersen. *IEEE 1394 for Linux.*
`http://www.linux1394.org/doc/overview.html`, December 1999.

[17] Chad N. Tindel, Brian D. Pietsch. *IEEE 1394 and Linux.*
`http://www.csc.calpoly.edu/~ctindel/550/firewire.html`, Spring 2000.

[18] *FireWire Overview.*
`http://developer.apple.com/techpubs/macosx/Darwin/IOKit/DeviceInterfaces/FireWire/WorkingWFireWireDI/FWDevInterfaces/chapter_1_section_3.html`.

[19] *International Electrotechnical Commission.*
`http://www.iec.ch/`.

[20] Stefan Rubner. *Tech Brief: Firewire.*
`http://eu.computers.toshiba-europe.com/cgi-bin/ToshibaCSG/download_whitepaper.jsp?z=71&service=EU&WHITEPAPER_ID=0000000b7e`.

[21] Tobias Oebrink. *Payload format design continued.*
`http://www.it.kth.se/~nv91-tob/Report/Lic/Apr1999/packetization.html`, 1999.

[22] *DV format encoding.*
`http://www.zvon.org/tmRFC/RFC3189/Output/chapter2.html`.

[23] Chris Needham. *Tsai camera calibration software.*
`http://www.comp.leeds.ac.uk/chrisn/Tsai/index.html`, February 2003.

[24] Tsai, Roger Y. *An Efficient and Accurate Camera Calbration Technique for 3D Machine Vision.* Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 1986, pages 364-374.

[25] Andy Hao ZHOU. *Multi-Camera Calibration for the Ada/Expo.02 Vision Matrix*, 2003.

[26] *Guidelines for semester and diploma thesis of the PCCV group.*
`http://www.vision.ethz.ch/dasa/guidelines.pdf`.

[27] *Qt, a C++ Application Framework.*
`http://www.trolltech.com/products/index.html`.

[28] *Dvgrab 1.2*
`http://kino.schirmacher.de/article/view/58/1/7`.

[29] *Linux DV Camera console control program.*
`http://www.spectsoft.com/idi/dvcont/`.

[30] *FFMPEG Multimedia System.*
`http://ffmpeg.sourceforge.net/`.