matrixIxirtam
|
matrixi: an open project for camera synchronisation check

Sebastien COUDERT

June 22, 2011

Contents

Chapter 1

matrixIxirtam

1.1 description

- aim: **check multiple camera synchronisation**.
- solution: simply based on imaging a number that is changing in time.
- constrain: **fast and reliable processing**. easy implementation.

1.1.1 no number as characters

no OCR: too complex and too slow. parallel to conversion from binary to ASCII (and vice versa) = lot of CPU time.

1.1.2 number as matrix

digital number = bits. use **16 light spots to create 16 bit number**. 16 light spots layout on a **4x4 matrix** (i.e. 2D array). image processing is simple: get bits on image:

- high level = 1 = true
- low level = 0 = false

matrix layout as:
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
For example, number 123 (i.e. 0000 0000 0111 1011 as binary) will show as follow in **image axis convention**:
1 1 0 1
1 1 1 0
0 0 0 0
0 0 0 0
easy implementation in C++ or GHDL language.

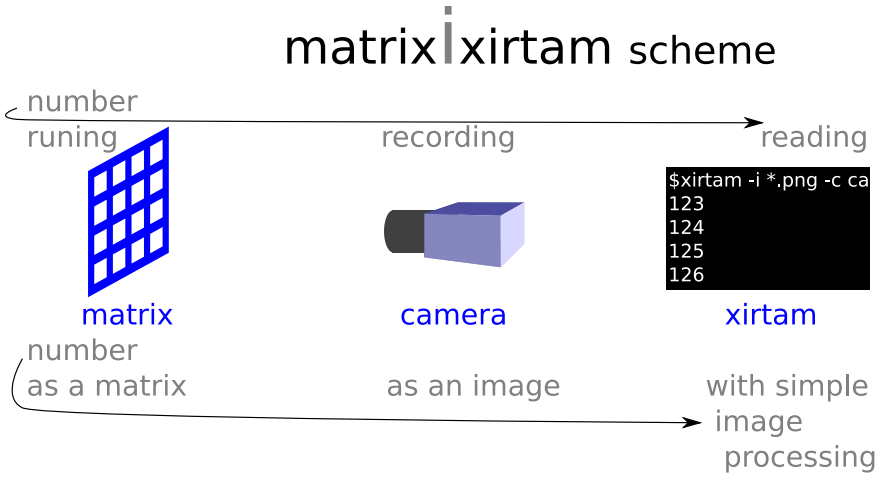

Figure 1.1: matrixIxirtam overview

1.1.3 project name

Project is nicknamed *matrixi* for easy prononciation. The full name is *matrixIxirtam* to stand for a given number that is converted to a matrix then it is convert back to number. One way is *matrix* the reverse way is *xirtam* (i.e. **reverse order** characters of matrix). It is a kind of mirroring: i as a capital letter stands for | which represent the **mirror** (i.e. matrixIxirtam). This name is easy to write as text (e.g. Linux terminal or C++ code) or render as a figure (e.g. XWindows display or LaTeX).

1.2 component

1.2.1 description

The components of the project layout on the figure 1.1.

1. The number is displayed as a light spot matrix using a device called *matrix* **component** in the project.
2. The image of the matrix is recorded using one (or multiple) camera but which is not part of the project.
3. The images are processed using the *xirtam* **component**, that decodes the number.

To comment on each step again:

1. *matrix* component is creating a number incremented in time (either software or hardware delay). For example, the number begins at 123 (i.e. 0000 0000 0111 1011 as binary or bit matrix) for first image, then after the delay it is 124 (i.e. 0000 0000 0111 1110) for second image and so on.
2. multiple cameras may be able to record images of the same *matrix* component to record same times (i.e. same number as a matrix; e.g. 124 for all second images of cameras).

3. *xirtam* component is reading the numbers (e.g. 123,124,125,126,...) and may compare it either from preceeding or following one (e.g. 123,124,125) and in between cameras at the same image index (e.g. 124), to check for synchronisation.

note: fast image processing (real time). no complex OCR.

1.2.2 matrix

write number as a matrix. 16 light spots. different ways of making the matrix image. examples

- display on a computer screen (see section 2.1 $CImg.matrix$)
- LED display controled by a microcontroler (see section 2.2 $LED.matrix$)

use depending on accuracy and speed.

1.2.3 xirtam

read number from the recorded image using the matrix image. different ways of converting the matrix image to the corresponding number. examples

- sequence of previously recorded images (see section 4.1 $CImg.xirtam$)
- real time reading (see section 4.2 $OpenCV.xirtam$)

Chapter 2

matrix

2.1 CImg.matrix

based on CImg library. display the matrix image on the computer screen that is runnning the program. stand alone component, but display images of the running number with a millisecond accuracy. see doxygen documentation

- user
- developper

2.2 LED.matrix

actually under implementation. 16 LED for number as a matrix (i.e. 16 bit number incremented at the end each period time). 03 LED for axes markers (always on). (01 LED for exposure) not stand alone component: it need to be controled by an other device such as $AVR.matrix$ or $ARM.matrix$.

2.2.1 LED.matrix single sided

On the figure 2.1, the picture of the actual LED.matrix layout shows, on the right side in the foreground, the LED.matrix component plugged in the connectors of the Arduino Mega (left side and background).

2.2.2 LED.matrix double sided

2.3 AVR.matrix

future implementation on an AVR microcontroler using a LED.matrix. e.g. Arduino Mega.

2.4 ARM.matrix

future implementation on an ARM microcontroler using a LED.matrix e.g. LeafLabs Maple.

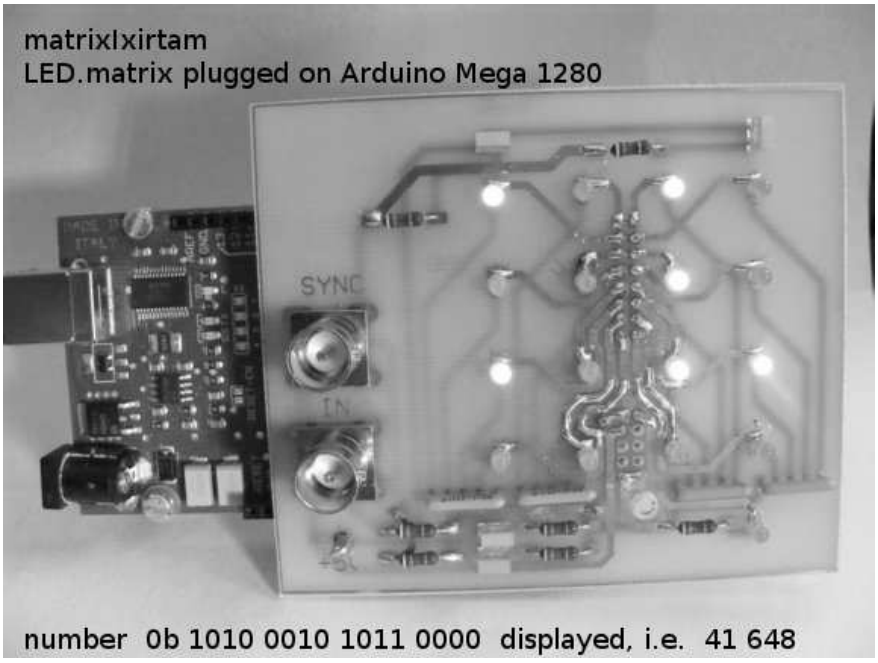

Figure 2.1: LED.matrix picture

Chapter 3

image recording

3.1 independant

record images with thrid party software.

3.2 part of xirtam

record images with matrixIxirtam project software. future implementation (see for example section 4.2 *OpenCV.xirtam*)

Chapter 4

xirtam

3 main steps in *xirtam* component:

1. first, calibration procedure to get position within the image of the matrix on the field of view, and also values for both minimum and maximal gray levels that corresponds to low and high level of the bit. It require once the user interaction through a GUI (e.g. XWindows) on a single image per camera. This may be saved in a file for future use if neither the camera nor the matrix component are moving.
2. then, read number using simple image processing from the recorded images.This may be run in batch mode using a calibration file.
3. finally, compare numbers to a reference and/or in between cameras.

4.1 CImg.xirtam

based on CImg library. read standard image format. specifications:

- handle multiple cameras in only one program call.
- run either in GUI or batch modes.

see doxygen documentation

- user
- developper

4.2 OpenCV.xirtam

future implementation using OpenCV/GStreamer. may: record (through GStreamer) and process in real time.

4.3 Elphel.xirtam

future implementation using either embedded FPGA and/or ARM. may: record and process in real time, outputting embedded Linux date and corresponding number for each camera.