Image Processing

for

Beam Diagnostics

on a

PC under Windows

using the

Portable CAS

B. Franksen, B. Kuner
(BESSY)

The Idea

- use video images of the beam for real-time (10 Hz) diagnostics
- analyze images to get quantified information about beam position and quality (size, orientation, focus)
- use a PC with a frame-grabber card to save money and because VME cards are not really available
- take the portable CAS to create a remotely (CA) controllable and readable beam diagnostic tool

No Problem...

...i thought: get your M$ Visual Studio running, install the frame-grabber libraries, plug in a CCD camera and start hacking...

but:

- what kind of image analysis should be done, actually?
- how to program the CA server tool?

Real-Time Image Analysis

beam profile should roughly look like an ellipse, but

- how do we find its center, orientation, outline, ...?
- what to do if it is *not* shaped like an ellipse?
- particularly, how can we do this *really fast*?
  - standard CCD resolutions are about 440.000 pixels / image
- we also want to have projections on the coordinate axes and
- the image to be displayed in real-time on the computer monitor, optionally with artificial coloring

Solution: Use Statistics

- interpret image as 2-dimensional probability density
- first moments (mean value) give position / center
- second moments (2x2 covariance matrix) give (square of) standard deviation in X and Y direction
- can be computed efficiently:
  - each pixel must be processed only once
  - during image traversal need only basic integer arithmetic (fortunately, M$ compiler supports 64 bit integers)
- projections can be computed in the same pass

Moreover...

- size and direction of main axes of the (so called) *ellipse of standard deviation* are easily obtained
  - by a singular value decomposition, find the rotation angle that diagonalizes the covariance matrix
    - this is always possible since cov(x,y) = cov(y,x), i.e. the matrix is symmetric
  - the diagonal elements of the rotated matrix then contains (square of) lengths of the axes
- floating point math operations necessary only *after* image traversal and only on the 6 numbers resulting from the traversal

Writing a CA server tool

- at the first look, the API to the portable CAS looks nice, clean and easy (even object-oriented)
- but for serious applications it is too low-level (provides only naked PVs)
- you want your PVs to have the usual attributes (like EPICS records), so that (e.g.) clients like MEDM get the graphics and control data for the channels
- soon you find yourself doing nothing else, but fighting a horrible beast: the *GDD*

GDD (Generic Data Descriptor)

- also known as the *Gruesome Developer Devourer:* you'll never want to get close to this thing
- CAS API hides a lot of complexity by using GDD as the data container
- scalar or array GDDs are *almost* manageable, but to provide graphic or control attributes, you need *container* GDDs (ugly, slow & buggy)
- at the moment, GDD limitations make it impossible to provide a correct and complete implementation of all CA features with the portable CAS

Solution: Wrap it up

- write a C++ class library on top of CAS API
- required features
  - hide *any* reference to GDDs
  - provide all standard attributes (as far as possible)
  - usable by the average programmer
  - support scalar & array PVs
  - hide the server routines (creation, destruction, name & PV instance management, ...)

The XCas Library

- started from an existing library by Kay-Uwe Kasimir (many ideas still in there, almost no code)
- user base class is called `Record`, because
  - the XCas `Records` are indeed similar to (simplified) EPICS records, e.g.
  - the attributes are accessible by CA with the usual EPICS record field names (HOPR, DRVH, ...)
- derived classes are `NumRecord<T>` (numerical data types), `StringRecord`, `EnumRecord`
- no distinction between input and output types

XCas Features

- easy to use:

```
NumRecord<double> num_rec("MYAPP:XX");
num_rec = 5.6;      // same as num_rec.setVal(5.6)
num_rec.setHopr(100);
double d = num_rec; // same as d = num_rec.getVal()
```

- automatic event posting on value or status change
- CA put notification is done by deriving from callback functor classes, templatized by an arbitrary user class
  - type safe (no casts)
  - callbacks can directly call user class methods