ISBN: 978-3-95450-241-7 ISSN: 2673-5350 doi:10.18429/JACoW-IBIC2022-MOP38

BEAM PROFILE MONITORING AND DISTRIBUTED ANALYSIS USING THE RabbitMQ MESSAGE BROKER

D. Proft*, K. Desch, D. Elsner, F. Frommberger, S. Kronenberg, A. Spreitzer, M. Switka
Physikalisches Institut, University of Bonn, Germany

Abstract

The ELSA facility utilizes several digital cameras for beam profile measurements on luminous screens and synchrotron radiation monitors. Currently a multitude of devices with analog signal output are being replaced in favor of digital outputs, preferably with data transfer via Ethernet. The increased network traffic for streaming, analyzing, and distribution of processed data to control system and machine operators is managed through a supplementary camera network in which distributed computing is performed by the `RabbitMQ` message broker. This allows performant and platform-independent image acquisition from multiple cameras, real time profile analysis, and supports programming interfaces for `C++` and `Python`. The setup and performance of the implementation are presented.

INTRODUCTION

As in most facilities the beam images at ELSA [1] are obtained through several different observation techniques via synchrotron radiation or luminous screens based on fluorescence, intensified phosphorescence or transition radiation. Thereby numerous camera systems from various manufacturers are in use, whose data acquisiation and transfer techniques are based on different interfaces. In practice, this makes it difficult to grant reliable and permanent access to all imaging systems for operators or control system algorithms, especially when hardware from different decades with individual digitizers, software, and software platforms are required. In the following we present an approach of unifying the various image streams to provide more reliable and uniformly accessible beam image data for beam observation, profile analysis, beam parameter measurements, and hence, machine optimization.

IMAGING HARDWARE

Cameras With Analog Output

The prevalent form of imaging has been based on cameras with analog output signals (e.g. *composite video*), which served its purpose for basic adjustment procedures, such as beam alignment. However, the analog signals are usually transmitted over long coaxial lines to distant video multiplexers and digitizers and are prone to interference with a multitude of disturbing signals from the accelerator environment. In those setups the requirements for quantitative beam profile measurements with adequate precision are rarely met due to visual defects from transmission or imperfect exposure settings, as shown in Fig. 1 a). In addition, multiple digitizer cards installed at different work stations suffer from aging and incompatibility with modern operating systems. Some images are not digitized at all and broadcasted to a screen in the control room, lacking the advanced capability for processing by computer vision algorithms.

Cameras with Digital Output

The usage of cameras with digital signal output overcomes the above mentioned issues and the *GigE Vision* communication interface allows for cameras to be conveniently connected through a network of Ethernet switches allowing bi-directional communication between camera and accelerator control system (e.g. to set shutter time or frame rate). In this configuration power can be provided over the Ethernet cable itself (PoE), implying convenient hardware connectivity up to 100 m away from a network switch (compare with Fig. 1 b)). This is especially useful for temporarily installed monitoring cameras, which only require an Ethernet cable (e.g. on a reel) to function. As explicated below, cameras with *GenICam* interface allow good software connectivity.

Some camera systems often operate through enclosed, proprietary software environments running on dedicated computers, such as a streak camera (see Fig. 1 c). Usually, data transfer and device control inferfaces via TCP/IP protocol are availble and the computers can be used as image servers, whose data stream is manageable through prevalent programming environments such as *Python* or *C++*. This way any computer operating digitizer cards (*framegrabber*) can be integrated into the camera network.

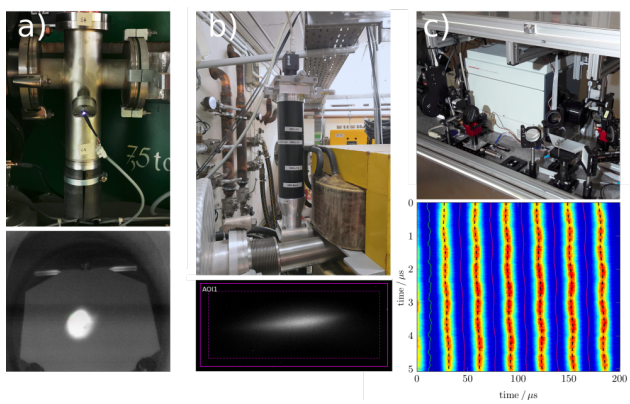

Figure 1: Examplary hardware and corresponding beam images: a) chromox screen monitor with saturated image and visual defects from analog signal, b) synchrotron radiation monitor equipped with Ethernet camera, c) streak camera as bunch train monitor with analyzed image (tune measurement).

* proft@physik.uni-bonn.de

ISBN: 978-3-95450-241-7 ISSN: 2673-5350 doi:10.18429/JACoW-IBIC2022-MOP38

Camera Network

To protect the accelerator control system and its process network from data traffic overload when *GigE Vision* cameras and other devices are connected, a separated network infrastrucuture of multiple gigabit Ethernet switches with power over Ethernet (PoE) capability are installed throughout the facility, where long distances are bridged via switches with fibre transmission capability. The accelerator process network is connected to the camera network through a gateway computer whose outgoing data traffic contains only selected information, such as size- and rate-reduced image streams or numeric fit results.

Magnification Calibration

The individual optical system of a beam monitor determines the image magnification and hence, the calibration ratio of pixel to object size, which can be as simple as a single factor. In the case of luminous screens, which are typically mounted under an observation angle of 45°, in most cases the warped image yields an image magnification as function of one transverse position (compare with Fig. 1 a)). In addition, digitizers may change the resolution of the original broadcast for lines or columns separately.

FGrabbit FRAMEWORK

`FGrabbit` is a distributed image acquisition and analysis framework for beam profile monitoring developed for ELSA. It supersedes the multitude of individually developed solutions — internally called *Framegrabber* — for the different monitoring stations spread around the accelerator facility. The development goal was to build one common software environment for all monitoring stations where only the interface to the corresponding hardware (camera) is device dependent. Emphasis was laid on a *distributed* system for network and CPU load balancing.

Message Broker & Concept

The fundamental building block is the `RabbitMQ`[1] message broker software, that is utilized to distribute *messages* to a number of software tools for further processing. These messages contain either raw or calibrated image data with sizes of up to 10 MiB per frame, or analysis results in the form of e.g. parameters of typically two-dimensional profiles.

Several instances of `RabbitMQ` running on different machines (called *nodes*) form a cluster and allow message distribution between each other. This way, transparent access to camera data is possible from every node. Furthermore, each node takes care of one or more cameras and usually the complete processing chain involved: processing of raw images to calibrated images, preparation of fits to user-selectable profiles and execution of the corresponding fit algorithms. Limiting the processing chain to only one machine reduces the network traffic drastically, as raw image data remains on the same machine. Nevertheless, multiple machines can take part, if e.g. more processing power is required.

The processing scheme for each node is shown in Fig. 2. The different steps are explained in detail in the following.

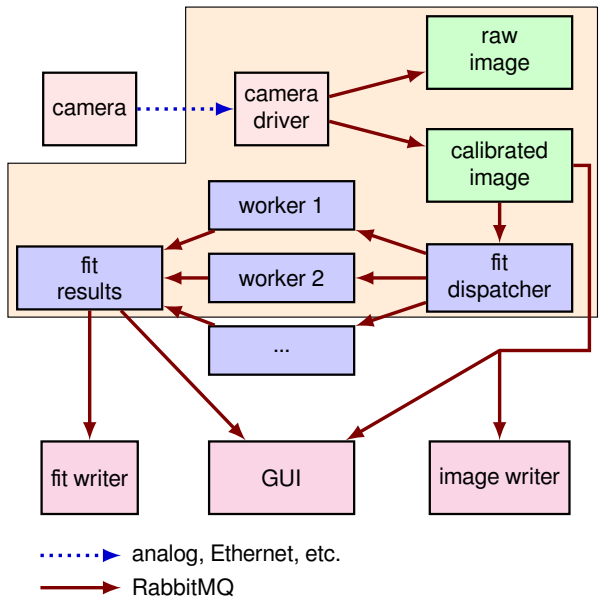

Figure 2: Schematic sketch of processing chain usually taking place on a single node (orange). The chain consists of image acquisition (red), provisioning of raw and calibrated images (green) and fitting (blue). User applications typically access the data from any node (magenta).

Image Acquisition

For the wide variety of cameras and image acquisition systems individual *camera drivers* were implemented. Most of the existing analog video capture cards from many different vendors are incompatible to modern PC hardware. The pre-existing image acquisition software was extended to stream the raw images via a compressed TCP/IP connection to a server application acting as camera driver for these devices. *video4linux* (e.g. USB and HDMI capture devices) and *GigE Vision* devices are natively included in the drivers — the latter also via the *harvesters* `Python` library (see below).

Image Calibration

A distinct calibration needs to be carried out for each of the stations. Hence, a regular checkerboard pattern (with up to 15 × 15 fields) with edge length of up to typically 5 mm is temporarily installed as screen and captured by the imaging system including the specific camera. To account for the tilt angle of 45° that the screen is mounted in relation to the camera (see Fig. 3), one transverse dimension of the calibration pattern is stretched by a factor of $\sqrt{2}$. For the calibration procedure a performant, reliable computer-vision based approach is used, where a sequence of raw camera images is captured, the corresponding positions of the inner edges of the checkerboard pattern are derived, averaged and

[1] Open source message broker: `https://www.rabbitmq.com`

ISBN: 978-3-95450-241-7 ISSN: 2673-5350 doi:10.18429/JACoW-IBIC2022-MOP38

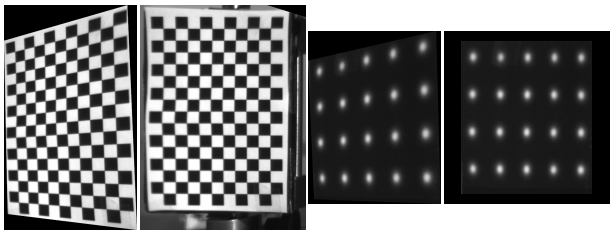

Figure 3: Left two images: Checkerboard pattern used for calibration as raw image from camera (left) and resulting calibrated image (right). Right two images: Test pattern with 5 × 4 Gaussian profiles as raw image (left) and calibrated image (right)

then mapped to the respective real world coordinates. With *openCV*'s 3D reconstruction and camera calibration capabilities [2] the corresponding 3x3 homography transformation matrix M_h and translation matrix T is derived and stored as a calibration data set within FGrabbit's configuration subsystem. Image coordinates $\vec{x}'$ can now be projected to real world coordinates $\vec{x}''$ using

$$\vec{x}'' = T \cdot M_h \cdot \vec{x}' \quad . \tag{1}$$

After applying the calibration, the raw camera images are automatically de-warped (interpolated) into calibrated images and published to RabbitMQ. All further image processing and analysis is carried out using the de-warped images (see Fig. 2).

With a test setup and a very rough calibration, by utilizing a paper printout of a checkerboard, measurements of the beam profile position and beam sizes with a relative deviance below 2% could be achieved. Even better results are expected as soon as the checkerboard pattern is replaced by a thin laser engraved PVC plate with the proper dimensions of the respective screen.

Processing / Fitting

From the de-warped image subimages can be cropped out according to the position and dimensions of *areas of interest* (AOIs), which can be easily defined via the graphical user interface (GUI). A *fit job* is created from the cut out segment and complemented by metadata (e.g. timestamps, see below) describing the original image as well as the function to be fitted to the data. The *fit job* is then dispatched to a *fit queue* from which several *fit workers* consume and eventually fit functions to the image data. The fit results (typically parameters μ_i, σ_i, rotation angle, intensity and background level of a 2D Gaussian profile) are finally published back to a RabbitMQ *exchange*[2].

A GUI displays the (calibrated) images, AOIs and fit results to the user, or dedicated tools can be used to e.g. write fit results to a file or transfer the results to to the accelerators control system.

[2] In the AMQ protocol messages are sent to *exchanges* and then routed to different queues for message delivery.

Configuration Subsystem

All configuration parameters of the cameras (e.g. frame rate, exposure time, external trigger configuration), the calibration data and the fit properties (e.g. AOI position, fit function) are stored in a central key-value database that is accessible via RabbitMQ. With appropriate callback mechanisms being implemented in the *C++* and Python library, parameter updates are broadcasted to the client applications. Remote procedure calls (RPC) allow for communication between the different applications as well as execution of common tasks (e.g. setup of new cameras, update of calibration data sets).

Deployment

All applications and libraries are deployed using a single Docker [3] image. With that approach, new RabbitMQ nodes can be joined into the cluster with ease and run isolated from the operating system. The nodes can either run the whole FGrabbit tool chain including *camera drivers*, *fit dispatcher* and *fit workers* or just run *fit workers* to increase computing power.

Python Interface

In addition to the framework's full-featured C++ interface, a Python interface allows for a quick and easy access to FGrabbit in projects by researchers and students. The Python interface is based on the RabbitMQ client library *Pika*[3]. Divided into different types of *exchanges*, the FGrabbit-features are handled by *exchange*-specific classes with a specialized set of functions. A brief overview of the functional scope is given in the following:

- The **image-data**-class includes functions for publishing images - both raw and calibrated. Each image contains a *header* with camera metadata, a timestamp and a serial number for identification and later assignment in the event of backlogs during time consuming tasks such as *fit jobs*. Likewise it is possible to subscribe to any camera *exchange*. One can choose to receive a raw stream or the already calibrated image.

- The **fit-data**-class allows users to access live fit data for a selected camera and AOI. Up to now *fit jobs* are handled by the reliable C++ fit library presented below. However, the architecture of the FGrabbit-framework enables Python scripts to handle *fit jobs* and publish fit results. This possibility can find use in application-dependent unique image analysis without the need to extend the C++-library.

- The **config-data**-class interfaces with the *configuration subsystem* described above. One essential function of this class is to request a complete key-value set regarding a specified *topic* (e.g. general, camera, calibration data, fit properties). Along with a subscription feature that allows for callback mechanisms upon configuration

[3] Open source AMQP 0-9-1 library: https://github.com/pika/pika

ISBN: 978-3-95450-241-7 ISSN: 2673-5350 doi:10.18429/JACoW-IBIC2022-MOP38

updates, an object can be configured initially and kept up to date. Likewise, an update function implements a way to control devices and set properties for AOIs, fits, etc. Within the config-data-class, additional RPCs can be executed for common tasks (e.g. adding cameras).

- The **log-data**-class attaches to the global programming language independent logging *exchange*. Besides the possibility to send log messages, there is also a subscription method to receive all or filtered log messages. This is used e.g. for a log viewer realized with the `cutelog` [4] GUI to investigate the overall health on the whole system.

With the briefly described features above and following the naming scheme in Fig. 2, `Python` scripts can embody *camera drivers*, *fit writers*, *image writers* and *fit workers*. In particular, the ability to have `Python` *camera drivers* simplifies image acquisition thanks to freely available `Python` libraries like the one described in the following section.

GenICam-Harvesters Library Bi-directional communication with e.g. *GigE Vision* devices can be established via the generic programming interface for computer vision cameras *GenICam* [5]. The *Harvesters*[4] `Python` library aims to simplify the image acquisition process via this interface. It allows both for easy acquisition through *GenTL* producers (the transport layer interface, here hidden for the consumer) and uncomplicated feature node manipulation to remote control devices. During the development of `FGrabbit`, *Harvesters* has proven to ba a reliable and simplified alternative to implementations of the *GenICam* interface by closed source software development kits (SDKs) from various manufactures.

IMPROVEMENT OF FIT PERFORMANCE

In addition to the optimization of the workflow using the new `FGrabbit`, the performance and accuracy of the fit algorithm was also improved. In the scope of [6] a `C++` library for the analysis of beam images at various camera observed beam monitoring stations was developed. The library is based on the *Gnu Scientific Library* [7] and *Ceres* [8]. This library implements two-dimensional fits for several functions suited for profile analysis. Special emphasis was put on the analysis of saturated and rotated profiles. Hence two fit functions were established for this cases. The first implements a Gaussian profile with a saturation compensation (by implementing a flattop into the function and thus neglecting the saturated area). The second implements a rotation angle, which clearly increases the computing effort, but also increases the fit accuracy for slightly rotated profiles noticeably. The performance of the new fit functions and the reached fit accuracy was extensively examined using a generator, which emulates typical images with known profile parameters. 5000 of these images were fitted by the new algorithm and the deviation between simulated and fitted profile parameters for all images were analyzed. The results of this statistical analysis are shown in Table 1. The relative deviation of the widths of the beam spot is in all cases in the order of a tenth of a percent or lower, the relative deviation of the intensity is around 1%. With the usage of the rotated Gaussian fit function relative deviations in the order of these displayed in Table 1 were reached also for rotated profiles.

Table 1: Relative Deviation of the Fit Results from the Simulated Values for 5000 Analyzed Images and The Results Presented as: $\mu \pm \sigma$, Referring to Mean and Standard Deviation of a Laplace Distribution

	Gaussian		Gaussian + *flattop*	
	not-sat.	sat.	not-sat.	sat.
$\Delta I_0/\%$	0.8 ± 0.4	-13.3	0.8 ± 0.4	1.0 ± 0.4
$\Delta\sigma_v/\%$	0.1 ± 0.5	5.9	0.0 ± 0.4	0.1 ± 0.8
$\Delta\sigma_h/\%$	0.1 ± 0.5	7.1	0.0 ± 0.4	0.1 ± 0.3

CONCLUSION

The `FGrabbit` framework, which is being developed at ELSA, provides a solution for processing a multitude of varying camera signals for beam profile analysis and other tasks. An improved library for two-dimensional fits and automated image calibration is included. The image acquisition and data processing work load is distributed over arbitrarily many worker computers in a separated camera network and managed by the `RabbitMQ` message-broker. The unified data can be easily accessed via interfaces in `C++` and `Python`, as well as through the accelerator control system.

REFERENCES

[1] W. Hillert *et al.*, "Beam and spin dynamics in the fast ramping storage ring ELSA", *EPJ Web Conf.*, vol. 134, p. 05002, Jan. 2017. doi:10.1051/epjconf/201713405002

[2] G. Bradski, "The OpenCV Library", Dr. Dobb's Journal of Software Tools, 2000.

[3] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment", *Linux J.*, vol. 2014, no. 239, 2014.

[4] A. Bus, "Cutelog – GUI for logging", 2019, https://github.com/busimus/cutelog

[5] European Machine Vision Association. "GenICam Standard", https://www.emva.org/standards-technology/genicam/

[6] S. Kronenberg, "Entwicklung einer Softwarebibliothek zur automatisierten Analyse von Strahlabbildern an ELSA", Bachelor thesis, University of Bonn, Bonn, Germany, 2020.

[7] M. Galassi *et al.*, GNU Scientific Library Reference Manual (Release 2.7), 2021, http://www.gnu.org/software/gsl

[8] S. Agarwal, K. Mierle & Ceres Solver Team, "Ceres Solver", 2022, https://github.com/ceres-solver/ceres-solver

[4] Developed by the GenICam collaboration: https://github.com/genicam/harvesters