BIAS Documentation

Introduction

BIAS is the program responsible for image acquisition and display, enabling asynchronous inspection of results and setting regions of interest for further automated analysis (in ***lyse***). BIAS is responsible for camera communication and raw data processing, implementing a number of drivers for different makes of camera. This document explains both how to use BIAS in regular operation, and how to extend it to support hardware which requires drivers not currently implemented.

Installation

Unzip bias2.zip to the desired directory. LabVIEW's user.lib may be a good choice.

The four included VIP files are required packages, which can be installed using the free VI package manager (VIPM community edition), available from http://jki.net/vipm

With VIPM open, select File->Open Package File(s), select the four VIP files, click OK, then select "Add To Library & Install", and follow the prompts. You may be required to, install several required OpenG community packages, which will be downloaded automatically.

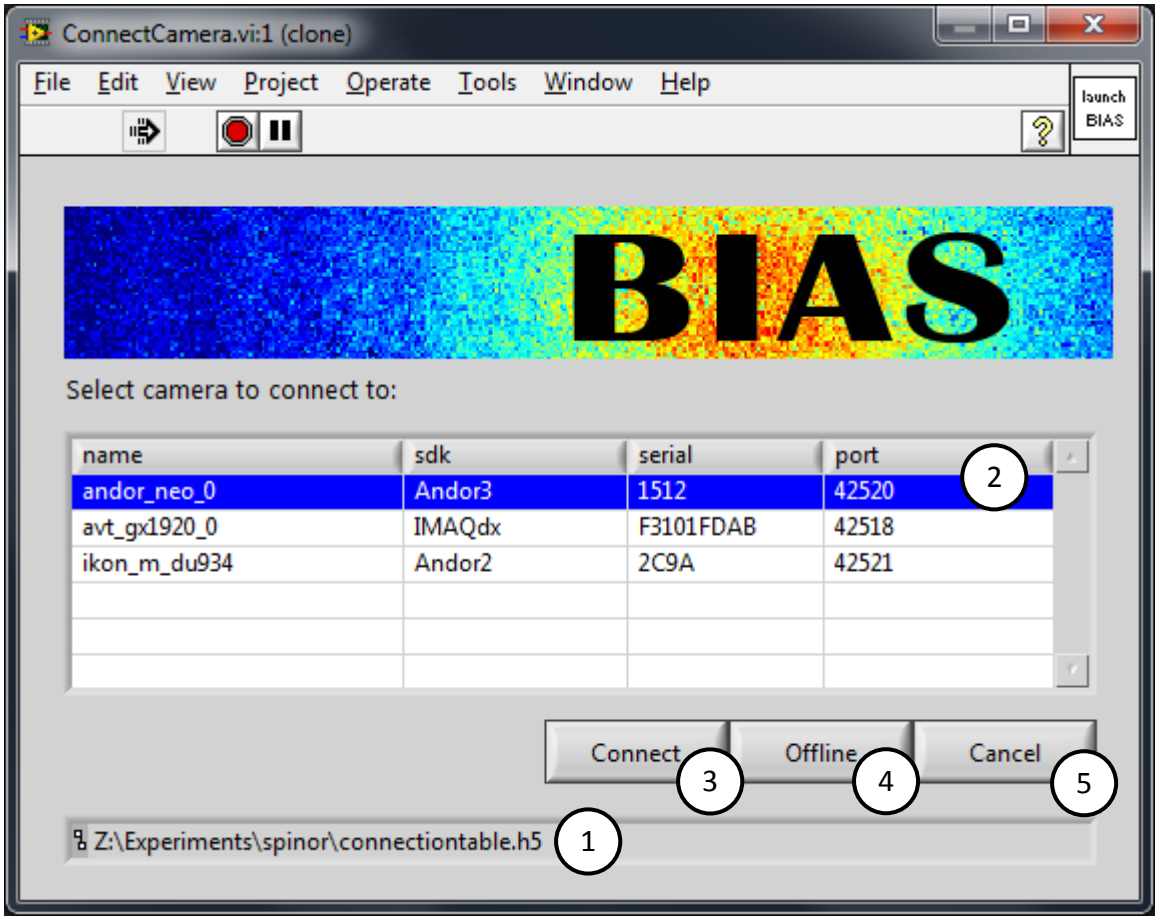
Optionally copy the *-errors.txt files to the directory user.lib\errors (e.g. C:\Program Files\National Instruments\LabVIEW 2012\user.lib\errors, create the directory if it doesn't already exist) to help translate error messages.

Getting started

1. Configure your computer's ***LabConfig*** (see some other document)
2. Open BIAS.vi in LabVIEW and click Run
3. Select camera to connect to, click "Connect"
4. Click Capture>Sensor to ensure camera is communicating correctly
5. Click Zoom>All to see entire area
6. Select an ROI and Capture>Selection to check sub-regions function
7. Click "Check connectivity" in the BLACS tab corresponding to this camera
8. Run a test script acquiring two frames on hardware trigger.

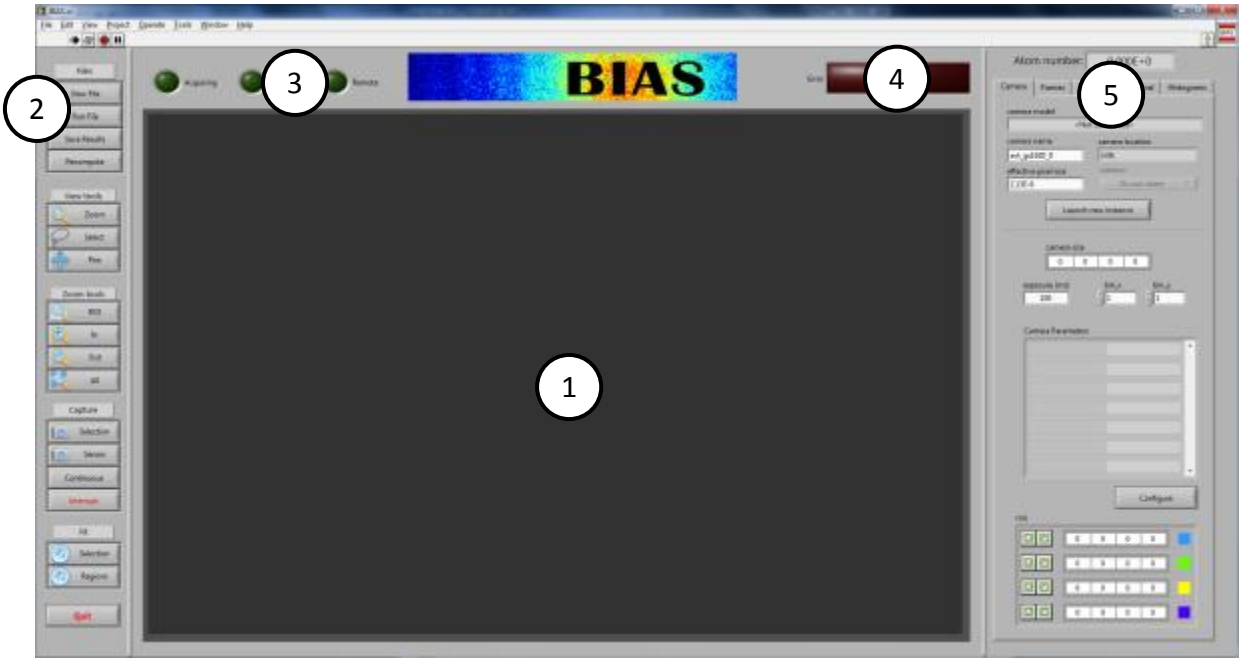
Connecting to a camera

When you start BIAS, it will spawn the camera connection window, shown below:



1. Location of your system's ***connection table***, as determined from your ***lab config*** file.
2. List of camera devices found in the connection table and its associated details:
    - "name" is the device's name in ***labscript***
    - "sdk" defines the camera class to load to communicate with the hardware (see ***adding cameras to BIAS***).
    - "serial" is a unique (hexadecimal) number used by the camera class to identify the hardware. Usually the manufacturer's serial number.
    - "port" is the TCP port used to communicate with the ***BLACS tab*** for this camera.
3. Clicking "connect" instantiates the camera class of the selected device, and instructs it to establish a connection to the hardware. Some devices may be particularly slow to initialise, so this can take up to a minute.
4. Clicking "offline" loads the settings of the selected camera but doesn't connect to the hardware. This lets you browse previously acquired data for that camera.
5. Clicking "cancel" causes BIAS to start without any camera settings or connecting to any hardware. To view previously acquired data these details must be entered manually (specifically "camera name" on the "Camera" tab).

Main BIAS interface



1. Primary display window
   With “View tools” set to “Select”, the current region of interest (ROI) can be set by clicking and dragging in the window. The ROI is displayed in green, and resize handles appear when you mouse over it (which distinguishes it from other regions). You can move it around by dragging the cross in the centre of the ROI and resize the boundaries. Clicking once on the display outside the ROI will remove it.
2. Button bar
   - “Files” has controls for loading and saving files.
     - “View File” opens an H5 file and any images stored in it, **for this camera**, for viewing.
     - “Run File” opens the H5 file as above, but parses any camera instructions, programs the camera and begins an acquisition.
     - “Save Results” saves the currently opened file with all settings, images, computations and fit results in it. **Files are not saved automatically to prevent overwriting good data**.
   - “View tools” and “Zoom tools” enable manipulation of the primary display
   - “Capture” provides *manual control* of the camera:
     - “Selection” software triggers the camera to recapture only the *currently selected ROI*.
     - “Senor” captures the camera’s entire acquisition region.
     - “Continuous” is a toggle, that when set retriggers the camera for a new acquisition as soon as the previous one has been completed (“movie mode”)
     - “Interrupt” sends tells the camera driver to abort the current acquisition. This is useful if the camera misses a hardware trigger or gets stuck in acquisition mode.

- "Fit" calls the fit function on either the currently set ROI ("Selection") or on all ROIs defined in the Camera tab. See also ***calculating with pybywire*** for how to modify the fit routines.
- "Quit" is an asynchronous call that aborts all running processes and stops the program. **This will lose any unsaved data**. Use only if camera driver hangs or state machine gets stuck and you need to end the process.

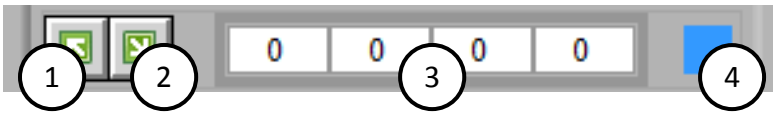
3. Status indicators
    - "Acquiring": BIAS is waiting to receive image data from the camera. This indicates that the camera has been programmed for acquisition and is waiting for a trigger, or is in the process of sending data back to the computer.
    - "Busy": BIAS is presently working on something, such as computation, file I/O, programming hardware, or waiting for results. Any tasks given to BIAS while busy will be queued up for later processing.
    - "Remote": BIAS received instructions from BLACS and will automatically configure the camera, download the images, run computations, and save resulting data.
4. Recent error indicator

    BIAS is designed to be run without user intervention, but enable the data to be inspected as it comes in. Therefore if an error occurs, *it should not stop the program running*. Instead, time-stamped error messages are listed in the "Log" tab and written to the log file "ImagingSystem.log" in the BIAS directory. To notify the operator that something has gone wrong at some point, this error indicator will turn red, indicating the log needs to be inspected. For example, if the camera misses a trigger because of bad sequence timing, this should not prevent the next sequence (which may have fixed timing) from executing.
5. Tabs

    Help to organise the variety of options and features BIAS includes.
    - "Camera": Camera settings and ROIs (see ***regions of interest***)
    - "Frames": Details about the H5 file, and lists of captured data. Select a frame to display it. The controls below the frames lists are visualisation setting, including min/max and palette. The controls at the bottom are settings for Optical Depth calculations.
    - "Fit": Settings and results of fit routines. If "Auto fit" is enabled, a fit is performed on each new dataset as it comes in. The fit ROI is extracted and passed to python (see ***calculating with pybywire***) for fitting. The results are from the *most recent fit*, which may not be the displayed image. This is to enable the fit to be inspected while waiting for new data to come in. When the results do correspond to the current data, the indicator "current" is lit.
    - "Log": Time-stamped errors and warnings. Click the "most recent error" indicator at the bottom to reset the "Recent error" indicator (see item 4).
    - "Internal": Settings used internally by BIAS, and results of calculations to be saved to files. These are generally not intended to be modified directly, but exist for debug purposes.
    - "Histograms": If the "Histograms" option on the "Frames" tab is enabled, each raw frame is histogrammed and displayed in a separate graph on this tab. This is helpful in diagnosing camera saturation, dark noise levels, and so on. However enabling this feature may be very slow for cameras with a large number of pixels.

Regions of interest

Multiple ROIs can be specified in BIAS, from the image capture region ("Capture selection" button, or "Internal" tab), to the fit region ("Fit" tab), to arbitrarily many regions for later analysis in lyse (bottom of "Camera" tab). Regions are modified using the controls shown below:



1. Set current ROI to show the region specified here
2. Set the values here using the current ROI
3. RECT containing the coordinates of the region in pixels. In order: left, top, right, bottom
4. Draw a box showing this region on the display in this colour. Click to change, set to transparent (T) to not draw.

Sequence procedure

BIAS follows the process below when capturing images as part of an experimental sequence under control from BLACS.

Receives filename from BLACS	When BLACS transitions to buffered mode, it sends BIAS the name of the experiment shot file.
Open H5 file, read camera properties	BIAS opens the H5 file and reads properties related to the camera it has connected to, and the sequence of shots to take.
Programs camera	Configures camera for acquisition based on values in the H5 file.
Set camera into hardware mode	The camera is set to wait for a hardware trigger. To prevent hanging and to provide time for readout, a timeout is set to 1min past the end of the experiment (as estimated by *labscript*).
Wait for data	BIAS waits for the camera to trigger, then extracts image data.
Report success/failure to BLACS	If BIAS successfully retrieves the expected number of frames, it returns success to BLACS; otherwise it reports failure and stops.
Run "compute" routines	Uses *pybywire* to perform computations in python (currently only optical depth), producing "Computed Frames" for viewing.
Run "fit" if Auto-fit	If "Auto-fit" is specified, the region specified on the "Fit" tab is fitted using *pybywire*.
Display results	If a computed frame was created (i.e. an optical depth image), it is displayed.

Display is parallel to computation, so frames can be inspected at any time. In particular, when running a sequence of experiments, the images from the last shot can be examined while the next shot is running.

Calculating with pybywire

Calculations, such as of optical depth and 2D Gaussian fitting, are scripted in python and run from LabVIEW using *pybywire*. The *pybywire* package includes a python server, and LabVIEW VIs for interfacing with it. If a calculation is run and the *pybywire* server is not already running, a prompt is displayed which can start the server for you.

Any function contained in subvis/calc/pbw_functions.py can be called from *pybywire*, and other python modules can be imported and called. For example 2D Gaussian fit routines are stored in fit_gaussian_2d.py and imported by pbw_functions.py, making them accessible to LabVIEW.

Functions must return a dict, which is then passed back to LabVIEW and entries can be extracted. The fit routine returns a dict of fit parameters, which are displayed on the "Fit" tab, and all get written to the H5 file, regardless of the type of fit being computed.

If an error occurs during execution, this is fed back to LabVIEW and stored in the error log, but will not interrupt program flow.

Adding cameras to BIAS

When you create a camera device in ***labscript*** you specify the "SDK", which appears in the ConnectCamera dialog when you start BIAS. This is the name of the class instantiated by BIAS to control the camera, and corresponds to a directory in the "classes" subfolder of the BIAS directory. Object orientation means behaviour can be overridden where desirable and provides a calling scheme.

The classes provided with BIAS are:

- **IMAQdx**
  National Instrument's IMage AQuisition (deluxe) driver enables any camera compatible with Measurement Automation eXplorer (NI-MAX) to be controlled. This covers most cameras that do not require a custom SDK. Currently written specifically for GigE cameras
- **Photonfocus**
  Photonfocus GigE cameras are compatible with NI-MAX, but the acquisition attributes have different names, which break the usual convention. Hence these cameras must be programmed differently, but connection, query and download are unchanged. This class therefore inherits the IMAQdx class, and overwrites only the programming function, which demonstrates the advantage of an object orientated approach.
- **Andor3**
  Andor's sCMOS cameras require a custom SDK dubbed SDK3, a LabVIEW version of which is available from your distributer upon request. After installing the vendor drivers, users are

required to install the at3mod package (http://at3mod.sf.net) over the top, to fix broken functionality in the official drivers.

- **Andor2**
  All of Andor's other products (at the time of writing), including CMOS, CCD and EMCCD cameras use an entirely different custom SDK called SDK2 (confusing, yes). The Andor2 class abstracts this SDK to a simpler interface. Comparing Andor2 to Andor3 demonstrates why class abstraction is so useful since the SDKs for a single manufacturer bear no similarity to each other!

If your camera is supported by one of these drivers, just add it to your connection table with the correct serial number.

If you have a camera that requires a driver not listed here, you can add an SDK to BIAS by creating a sub-class of CameraBase. Some knowledge of LVOOP (LabVIEW Object Oriented Programming) is assumed.

1. Open the CameraBase.lvclass
2. File->New... and select Class
3. Enter the name for your SDK
4. Right-click on the lvclass, select Properties, go to Inheritance
5. Click "Change Inheritance..." and select CameraBase and "Inherit from Selected"
6. Right-click on the lvclass, select New->VI for Override...
7. These are the functions you can override. Anything with asterisks ***must*** be overridden with new a VI to implement the required behaviour for your custom SDK so that it can interface with BIAS.

Check the documentation on the CameraBase VIs for information on what functionality they need to implement.

Potential problems

- Missed frames/retriggering
  Individual cameras have different requirements for back-to-back timing sequences. You will have to experiment with the minimum-possible inter-exposure time, because cameras that receive a TTL trigger before they are ready usually simply ignore it. This causes lost frames and an error at timeout.
  Sometimes the minimum time is in the specifications, but it will depend on acquisition area, readout speed, and so on. Often there is a "reset" time after readout before the camera is retriggerable.
  Interline CCD are fastest and can "double-back"; expose the next frame while reading the previous. High efficiency CCDs are slowest because of their serialised readout process.

- Timeout
  If your camera trigger is near the end of your experiment, you may experience timeout without getting data - particularly if your camera has a slow read-out cycle (see above). Also, since the camera clock starts ticking as soon as it's put in hardware mode, if your entire

experimental sequence takes a while to program (e.g. slow serial devices) that will also count unfairly towards the “timeout”
A constant (60s) gets added to the total time of the experimental run in the “Acquire” stage of the state-machine, which is used as the timeout time. Increase this constant if you experience difficulty with this (make sure it’s not a retrigger.

- Out of memory
  Cameras with a large number of pixels return a substantial amount of data, which then takes up a lot of memory in intermediary data structures. If you experience out-of-memory errors, particularly when capturing a single frame from a particular camera, try capturing a sub-region. Use roi_capture on the “Internal” tab to set the region if necessary.