
motmot Documentation

Release 0

Andrew Straw

June 26, 2010

CONTENTS

1	Overview	3
1.1	The name motmot	3
1.2	Packages within motmot	3
1.3	Mailing list	4
1.4	Related Software	4
2	Download and installation instructions	5
2.1	Quick install: FView application on Windows	5
3	Full install information	7
3.1	Supported operating systems	7
3.2	Download	7
3.3	Installation	7
3.4	Download direct from the source code repository	8
4	Gallery of applications built on motmot packages	9
4.1	Open source	9
4.2	Closed source	12
5	Frequently Asked Questions	13
5.1	What cameras are supported?	13
5.2	What frame rates, image sizes, bit depths are possible?	13
5.3	Which way is up? (Why are my images flipped or rotated?)	13
6	Writing FView plugins	15
6.1	Overview	15
6.2	Register your FView plugin	15
6.3	Tutorials	15
7	Camera trigger device with precise timing and analog input	25
7.1	camtrig – Camera trigger device firmware	25
8	Motmot development	31
8.1	Code repository	31
8.2	Buildbot	31
8.3	Email list	31
9	Packages reference	33
9.1	libcamiface - camera interface C API	33
9.2	cam_iface – Python wrapper of libcamiface	37

9.3	<code>motmot.fview_ext_trig</code> - camera trigger device	38
9.4	<code>fview</code> - Extensible realtime image viewing, saving, and analysis app	39
9.5	<code>flytrax</code> - <code>fview</code> plugin for tracking 2D points in realtime	40
9.6	<code>pygarrayimage</code> - fast display of numpy arrays to OpenGL/pyglet	40
9.7	<code>motmot.FlyMovieFormat</code> - simple, uncompressed movie storage format	41
9.8	<code>pygxinput</code> - use <code>XInput</code> devices in pyglet	45
9.9	<code>motmot.wxglvideo</code> - CPU friendly display of uncompressed video using <code>wxPython</code>	47
9.10	<code>motmot.wxvideo</code> - display of uncompressed video using <code>wxPython</code>	48
9.11	<code>motmot.FastImage</code> - SIMD image processing	49
9.12	<code>motmot.imops</code> - manipulate image codings	50
10	Citations	51
10.1	Scientific papers which used the <code>motmot</code> suite of tools	51
11	Indices and tables	53
	Module Index	55
	Index	57

Motmot is software for the realtime collection and analysis of uncompressed digital images from a variety of sources. The target audience ranges from anyone needing a simple GUI to record video to the developer wanting to implement a new realtime vision algorithm within an open, plugin-based environment. A primary goal is high throughput and low latency for use in realtime applications. We routinely use these libraries to process data from cameras with data rates up to 60 MB per second (with processing power to spare) on commodity hardware.

The program **fview** is a high-level application for viewing camera output, adjusting camera parameters, streaming video to disk, and providing an interface for that realtime processing routines can plugin to. Much of the rest of motmot is a modular set infrastructure components for FView or plugins that utilize FView.

The primary language is [Python](#), but the low-level camera access library (*libcamiface*) is in C. Lots of heavy lifting happens using [numpy](#), C libraries utilizing SIMD instructions, or custom C code.

There is also firmware for a \$30 USB device to *trigger frame acquisition and sample analog inputs with precise timing*.

Motmot is open source software under the BSD license. See the [Download and installation instructions](#) page, including information about [Full install information](#), for more information.

Figure: the relationships of important motmot components. Python and numpy are required for almost all modules, and are not shown. Shaded rectangular nodes are GUI components, while rounded nodes are libraries in C, C++, or Python. Dotted arrows indicate an optional dependency (plugin relationship) rather than a hard dependency.

This software was developed by [Andrew Straw](#) within the [Dickinson Lab](#) at [Caltech](#) to facilitate experiments in the neurobiology, biomechanics and aerodynamics of the fruit fly, *Drosophila melanogaster*.

OVERVIEW

1.1 The name motmot

Where does the name motmot come from? In short, it grew out of a previous project named “kookaburra”, and motmot is a bird related to the kookaburra (both are related to the kingfisher). See [wikipedia on motmots](#) for more information.

1.2 Packages within motmot

High level GUI app

- `fview` - Application with plugin architecture to enable writing new realtime analyses by creating your own `process_frame()` function.

Core camera infrastructure

- `libcamiface` - camera interface C API
- `cam_iface` - Python wrapper of libcamiface
- `fview_ext_trig` - software and firmware for *precisely timed trigger generation with synchronized analog input*

Core display infrastructure

- `motmot.wxglvideo` - wxPython OpenGL interface for video
- `pygarrayimage` - transfer Python objects supporting the array interface to OpenGL textures
- `motmot.wxvideo` - wxPython interface for video
- `motmot.imops` - manipulate image codings

Analysis infrastructure

- FastImage - Pyrex based wrapper of Intel’s Integrated Performance Primitives (IPP) Library
- `motmot.FlyMovieFormat` - Code for manipulating .fmm movies. Includes Python (read/write) and MATLAB® (read-only) bindings.
- `realtime_image_analysis` - Implements background subtraction and 2D feature extraction using FastImage

FView plugins

- `fview_ext_trig` - software and firmware for *precisely timed trigger generation with synchronized analog input*
- `flytrax` - fview plugin for tracking 2D points in realtime and saving data and small images

- *fview_periodic_trigger* - example fview plugin to trigger an external device at a fixed interval
- *fview_change_trigger* - example fview plugin to trigger an external device based on image change
- *fview_c_callback* - example fview plugin that calls pure C code
- *fview_live_histogram* - example fview plugin that calls pure Python code
- *trackem* - multiple point realtime tracker

Miscellaneous

- *motmot_utils* - Facilitate versioning and configuring of motmot packages
- *posix_sched* - Python extension module to boost priority in POSIX systems
- *pygxinput* - use XInput devices with pyglet

Deprecated packages

- *wxvalidatedtext* - validated integer/float text entry field for wxPython
- *wxwrap* - wrapper to allow use of multiple wxPython versions

1.3 Mailing list

To stay up to date, ask questions, and share information, [join the motmot email list](#). The [archives](#) are also online.

1.4 Related Software

1.4.1 Similar open source libraries

See the [Augmented Reality Toolkit](#), <http://muonics.net/>, [unicap](#), [OpenCV](#), [pyvision](#), [camunits](#) (formerly [libcam](#)), and [Micro Manager](#) for lots of interesting and fun stuff.

1.4.2 GenICam™ and GigEVision™

Another project with similar goals to motmot/camiface is GenICam™ <http://www.genicam.com/>. Primary differences between camiface and GenICam™ include the following: 1) camiface has been developed by a single individual to support a limited number of camera features from a limited number of cameras and is necessarily narrower in scope than an API meant to encompass every available feature on every available camera. 2) camiface operates using existing drivers rather than creating a new implementation of the driver layer.

One implementation of GenICam™ appears to be Basler's Pylon. http://www.baslerweb.com/beitraege/beitrag_en_53074.html

For a description of GigEVision™ see <http://www.machinevisiononline.org/public/articles/index.cfm?cat=167>

For an discussion of these libraries from an open source perspective, see [this thread on the libdc1394-devel mailing list](#).

1.4.3 Similar closed source libraries/applications

- [Streams 5](#) by IO Industries
- [StreamPix](#) and [the Hermes API](#) by Norpix
- The MATLAB [Image Acquisition Toolbox](#).

DOWNLOAD AND INSTALLATION INSTRUCTIONS

2.1 Quick install: FView application on Windows

Kristin Branson (Janelia Farm Research Campus, HHMI) has kindly provided an [installer for a Windows version of FView](#). This installs FView.exe, which can be used to record uncompressed .fmf movies. The FlyTrax plugin is also included, allowing realtime tracking of individual flies. This downloader was built with the latest [libcamiface](#) installer, and we recommend Pt. Grey cameras for this release. (Their inexpensive Firefly MV USB camera is a good way to get started.)

FULL INSTALL INFORMATION

3.1 Supported operating systems

Installation of applications built from Motmot components, such as the movie acquisition program **fvview** or the Fly-MovieFormat player **playfmmf**, currently requires installation of several constituent modules. Due to this modular, multi-component nature of Motmot, use of a package manager facilitates easy installation.

Packages for the native package management system of Ubuntu Linux (release “Lucid Lynx”, version 10.04) are provided, making this is the the easiest way to get started and the best tested operating system. The provided Ubuntu packages manage these dependencies, allowing automatic installation and updates, while on other systems, these dependencies must be handled manually.

The components are also tested on Windows and Mac OS X. Source code, Windows binaries, and for *libcamiface*, Mac binaries, are provided. See above for a *FView installer for Windows*.

To summarize:

- Ubuntu Linux (10.04): **best supported**, native package management. See *below*.
- Windows 32 bit: binary packages provided for most components (see above *for FView installer*)
- Mac OS X: binary packages provided for libcamiface, build the rest from source
- Other operating systems: build from source

3.2 Download

3.2.1 libcamiface

Source and binaries packages of *libcamiface* can be downloaded from its [release page](#).

3.2.2 Python modules

With the exception of libcamiface (above), Motmot is comprised of standard Python packages. These are distributed on the [Python Package Index \(a.k.a. PyPI\)](#).

3.3 Installation

See the *libcamiface* page for platform-specific installation instructions on libcamiface.

The rest of motmot should be installed with standard Python idioms. For example, if you want to install FView, you can use the following command:

```
easy_install motmot.fview
```

This will download and install FView and all its Python dependencies.

3.3.1 Ubuntu packages

I maintain an Ubuntu repository to enable myself and my collaborators to use these packages with minimal installation difficulties. Following these instructions is, by far, the easiest way to get started with motmot, especially if you're not experienced with C and Python installation issues.

These instructions are for Ubuntu 10.04 (Lucid Lynx). Pre-built binaries are not provided for other distributions.

1. Start the Terminal (Applications->Accessories->Terminal).
2. In the terminal window, type the following. This will add Andrew Straw's repositories (the PPA is for purely open source software, and the `debs.astraw.com` site packages non-open software) to your list of repositories:

```
sudo add-apt-repository ppa:astraw/ppa
sudo wget --output-document=/etc/apt/sources.list.d/astraw.list http://debs.astraw.com/sources.list.
```

You will be prompted for your password – this is normal.

3. Accept Andrew Straw's keyring. Still from in the terminal window, type:

```
sudo apt-get update && sudo apt-get install astraw-keyring && sudo apt-get update
```

After lots of downloading, you will eventually be asked to accept the `astraw-keyring` package even though it cannot be authenticated (“WARNING: The following packages cannot be authenticated! `astraw-keyring` Install these packages without verification [y/N]?”). This is normal; type “y” (for yes) to trust Andrew Straw to install software on your computer.

4. Start Synaptic (System->Administration->Synaptic Package Manager).
5. Install `fview` by clicking the “Search” button and typing “python-motmot-fview” in the Search field. Then click the “Search” button. After a couple seconds, this will bring up a list of packages matching your search string.
6. In the small empty square next to “python-motmot-fview”, click once and select “Mark for installation”. Click on the “Apply” button.
7. If you are using a firewire camera, add your user to the “video” group, and change the owner of the `raw1394` device to this group. In the terminal window again, type:

```
sudo adduser $USER video
sudo chown .video /dev/raw1394
```

Once you did this, you will need to log out and log in again for the new group membership to take effect.

8. Start `fview` (Applications->Sound & Video->`fview`).

3.4 Download direct from the source code repository

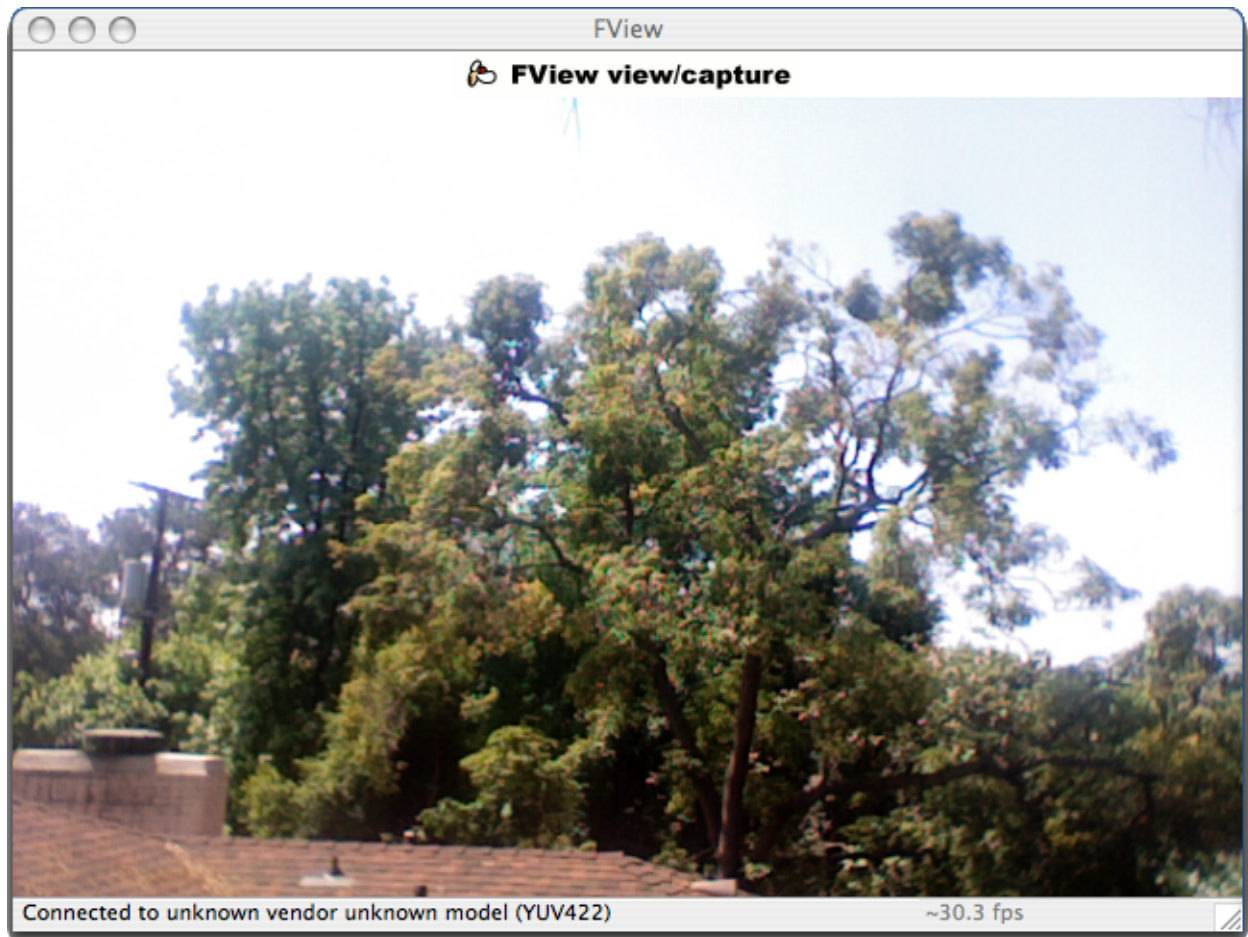
See the [Motmot development](#) page for details on how to download the latest version control repository.

GALLERY OF APPLICATIONS BUILT ON MOTMOT PACKAGES

4.1 Open source

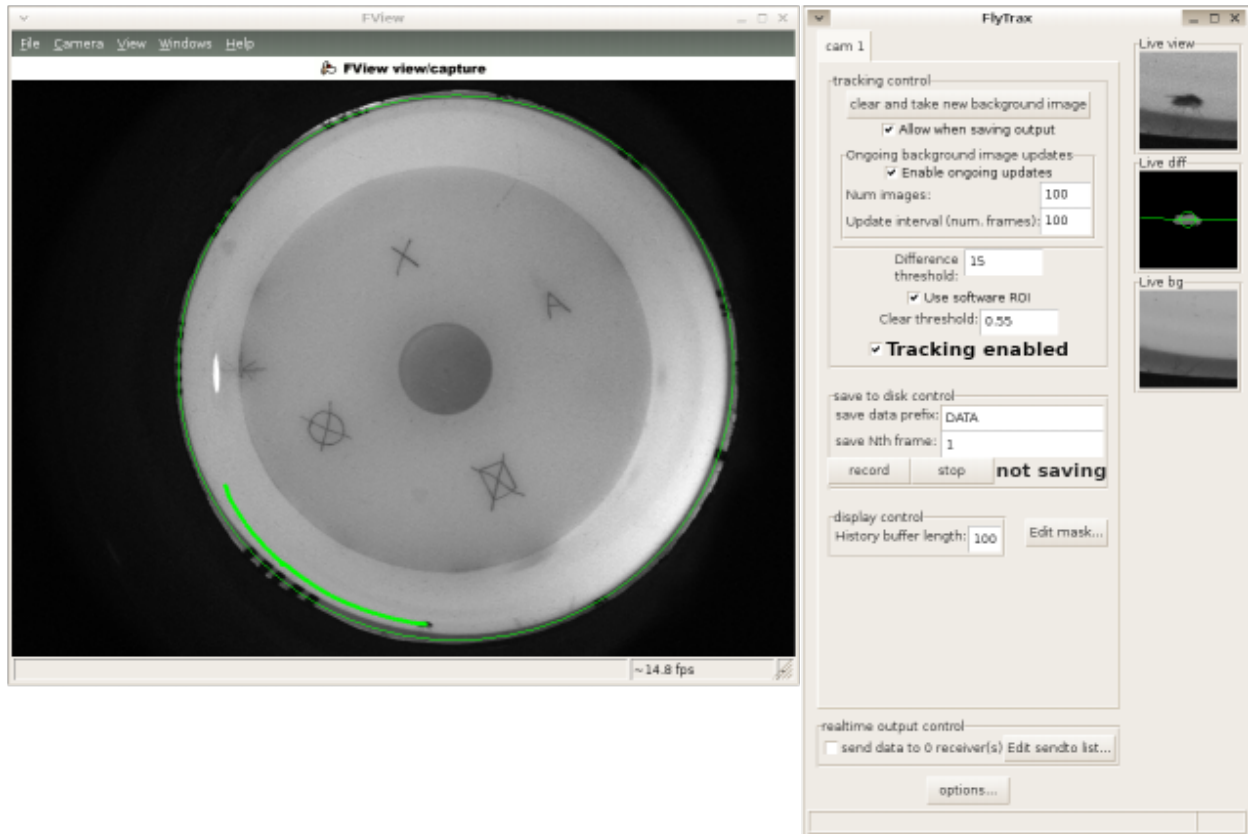
4.1.1 FView - Extensible realtime image viewing, saving, and analysis app

The `fview` application makes use of motmot components to create a realtime camera viewer program that can record videos to disk. It includes a convenient plugin mechanism by which realtime video analysis can be performed. FView runs on Linux, Mac OS X, and Windows. This application is essentially a GUI frontend for `cam_iface` built using the wx toolkit. You may be interested in visiting the [screenshot directory](#). It may optionally use our inexpensive, custom integrated *Camera trigger device with precise timing and analog input*. Here is a screenshot of fview running on Mac OS X using an iSight camera:



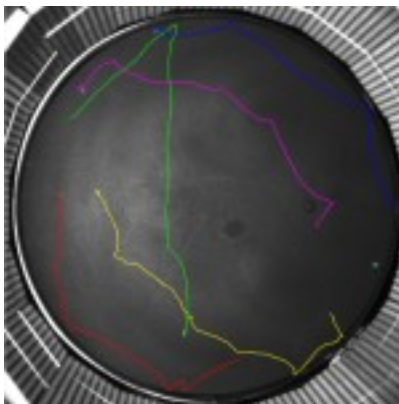
fview and flytrax screenshot

Here is another screenshot of fview and the [flytrax](#) realtime image analysis plugin. Note that the plugin is draws on the main display and to its own GUI window, which allows the user to interact with the online processing.



- `flytrax` is a realtime tracking system to track moving image features (a walking fly) using a simple but fast background-subtraction algorithm. Flytrax is freely available, open-source software distributed as part of the motmot package and functions as a plugin for `fview`.

4.1.2 ctrax - the Caltech Multiple Fly Tracker



- `ctrax` is multiple fly tracking software to enable quantitative analysis of fly behavior. From recorded videos, it tracks flies using a background/foreground model to track positions and orientations of individual flies over time. Identities of flies are maintained for long durations, with special care given to events such as close contact between flies and rapid movements. ctrax is free, open-source software.

4.1.3 trackem - Realtime multi-point tracker

- `trackem` tracks multiple bright or dark points and is used in our [insect inspired flight control testbed](#)

4.2 Closed source

4.2.1 Flydra - Realtime multi-camera tracking of multiple flying animals

- [Flydra](#) is a realtime tracking system for performing marker-less tracking of moving image features, such as flying insects or birds. This multi-camera system uses, at its core, the motmot packages running simultaneously on multiple computers connected to multiple cameras to enable high spatial and temporal resolution in a large tracking volume with standard commercial components.

Contact Andrew Straw astraw@caltech.edu to add your project to this list.

FREQUENTLY ASKED QUESTIONS

5.1 What cameras are supported?

See *What cameras are supported?*.

5.2 What frame rates, image sizes, bit depths are possible?

See *What frame rates, image sizes, bit depths are possible?*.

5.3 Which way is up? (Why are my images flipped or rotated?)

Motmot always keeps the camera pixel data in the same order and layout as provided from the source. This source is often live, uncompressed images direct from the camera sensor (rather than .avi files of any sort). With that in mind, some manufacturers read out their chips from top to bottom, and others from bottom to top. Thus, raw pixel coordinates are defined by the camera manufacturers – 0,0 is the first pixel to arrive, but it can be at any of the four camera corners. (Additionally, for some cameras, it is hard to discern a “correct” mounting orientation, with the top and bottom of camera enclosures being close to perfectly mirror symmetric.)

Therefore, the image processing components of Motmot operate in raw pixel coordinates. However, the *view* of the images in `motmot.wxvideo.wxvideo.DynamicImageCanvas` and `motmot.wxglvideo.simple_overlay.DynamicImageCanvas` may be changed by calling those classes’ `set_rotate_180()` and `set_flip_LR()`. The **fview** application calls these methods as appropriate based on the user’s selection in the View menu.

WRITING FVIEW PLUGINS

6.1 Overview

The `fview` program provides a simple extensibility mechanism. In outline, the steps required are:

- Create a subclass of `fview.traited_plugin.HasTraits_FViewPlugin`.
- Implement your GUI interaction using `traits`.
- Implement your realtime processing logic in your class's `process_frame()` method. Note that this code will be run in a separate thread of execution from the GUI, so be careful to avoid share memory structures without locking. The `buf` argument is a numpy array (or else supports the numpy array interface).
- Optionally, handle the various options allowed by FView.
- Finally, register your FView plugin.

To test your plugin, you can use the `fview` command directly, or you may use `fview_fmfr_replay` to test your plugin on a saved video recording.

6.2 Register your FView plugin

In your `setup.py`, use `setuptools` and add a `motmot.fview.plugins` key to `entry_points`. For the above example, this would be:

```
entry_points = {
    'motmot.fview.plugins': 'fview_ext_trig = motmot.fview_ext_trig.fview_ext_trig:FviewExtTrig',
}
```

6.3 Tutorials

6.3.1 FView plugin tutorial: live histogram of image intensity values

This tutorial illustrates the steps necessary to compute and display a live histogram of image intensity values.

A working copy of this code can be found at http://github.com/motmot/fview_histogram/

Prerequisites

- [Chaco](#) installed (see the [Chaco docs](#) for more information).

Step 1 - Build the file layout

Your plugin will be a standard Python package. Create the following directories (base should be your base directory, such as “fview_histogram”):

```
base
base/motmot
base/motmot/fview_histogram
```

And create the following empty files:

```
base/motmot/__init__.py
base/motmot/fview_histogram/__init__.py
```

Step 2 - Create setup.py

Now, edit `base/setup.py` to contain the following (modify as necessary):

```
from setuptools import setup, find_packages
import sys, os

setup(name='motmot.fview_histogram',
      description='live histogram plugin for FView',
      version='0.0.1',
      packages = find_packages(),
      author='Andrew Straw',
      author_email='strawman@astraw.com',
      url='http://code.astraw.com/projects/motmot',
      entry_points = {
        'motmot.fview.plugins': 'fview_histogram = motmot.fview_histogram.fview_histogram:FviewHistogram',
      },
      )
```

This is a standard [setuptools](#) file for distributing and installing Python packages that tells Python which files to install and some associated meta-data.

The `packages = find_packages()` line tells setuptools to look for standard Python packages (directories with an `__init__.py` file). Because you just created these directories and files in Step 1, setuptools automatically knows these directories contain the files to be installed.

The `entry_points` line tells setuptools that we want to register a plugin. Our plugin is registered under the `motmot.fview.plugins` key. FView inquires for any plugins under this key. This particular plugin is called `fview_histogram`. It is defined in the module `motmot.fview_histogram.fview_histogram` and the class `FviewHistogram`, which we create below.

Step 3 - Create a do-nothing plugin

Now we're going to create the module `motmot.fview_histogram.fview_histogram` with our new class `FviewHistogram`. Open a new file named:

base/motmot/fview_histogram/fview_histogram.py

The contents of this file:

```
import motmot.fview.traited_plugin as traited_plugin

class FviewHistogram(traited_plugin.HasTraits_FViewPlugin):
    plugin_name = 'image histogram'

    def camera_starting_notification(self, cam_id,
                                    pixel_format=None,
                                    max_width=None,
                                    max_height=None):

        # This function gets called from FView when a camera is
        # initialized.

        return

    def process_frame(self, cam_id, buf, buf_offset, timestamp, framenum):
        draw_points = []
        draw_linesegs = []

        # This function gets called from FView immediately after
        # acquisition of each frame. Implement your image processing
        # logic here.

        return draw_points, draw_linesegs
```

Step 4 - Create the plugin logic

From here, we're going to fill in the relevant parts with the code that our plugin executes. The most important function is `process_frame()`. This function is called by FView immediately after every frame is acquired from the camera. The arguments to the function contain information about this latest frame, and the return values are used to plot points and line segments on the main FView display. For the live histogram, we don't need to display anything, so we just return empty lists.

The contents of this file:

```
import warnings, time
import enthought.traits.api as traits
import motmot.fview.traited_plugin as traited_plugin
import numpy as np
from enthought.traits.ui.api import View, Item, Group
from enthought.chaco.chaco_plot_editor import ChacoPlotItem

# For a tutorial on Chaco and Traits, see
# http://code.enthought.com/projects/chaco/docs/html/user_manual/tutorial_2.html

class FviewHistogram(traited_plugin.HasTraits_FViewPlugin):
    plugin_name = 'image histogram'
    update_interval_msec = traits.Int(100)

    # The following traits are "transient" -- do not attempt to make
    # state persist across multiple runs of the application.
```

```
intensity = traits.Array(dtype=np.float,transient=True)
data = traits.Array(dtype=np.float,transient=True)

pixel_format = traits.String(None,transient=True)

last_update_time = traits.Float(-np.inf,transient=True)

# Define the view using Traits UI

traits_view = View(
    Group(
        ChacoPlotItem('intensity','data',
            x_label = 'intensity',
            y_label = '',
            show_label=False,
            y_auto=True,
            resizable=True,
            title = 'Image intensity histogram',
        ),
    ),
    resizable=True,
)

def camera_starting_notification(self,cam_id,
                                pixel_format=None,
                                max_width=None,
                                max_height=None):
    self.pixel_format = pixel_format
    self.intensity = np.linspace(0,255,50) # 50 bins from 0-255

def process_frame(self,cam_id,buf,buf_offset,timestamp,framenum):
    draw_points = []
    draw_linesegs = []

    if self.frame.IsShown():
        now = time.time()

        # Throttle the computation and display to happen only
        # occasionally. The display of the histogram, especially,
        # is computationally intensive.

        if (now - self.last_update_time)*1000.0 > self.update_interval_msec:
            npbuf = np.array(buf)
            if self.pixel_format == 'MONO8':
                self.data, edges = np.histogram(npbuf,
                                                bins=self.intensity,
                                                new=True)
            else:
                warnings.warn("histogram for %s format not implemented"%(
                    self.pixel_format,))
            self.last_update_time = now

    return draw_points, draw_linesegs
```

Afterward: optional improvements

As implemented above, the live plot of the histogram is automatically updated whenever the `self.data` attribute changes. While this is convenient, there's a problem with this. The `process_frame()` method is called in the camera acquisition and processing thread, and any computationally expensive process will slow down this thread. Drawing auto-scaled plots certainly qualifies as computationally intensive. Therefore, it would be better to calculate the histogram values as done here, but then to send them to another thread for display in the GUI. This would allow the image acquisition thread to operate unimpeded, but would require multi-threaded programming, which is beyond the scope of this tutorial.

6.3.2 FView plugin tutorial: periodic triggering of an external device

This tutorial illustrates the steps necessary to trigger an external device at a constant, relatively slow rate. Specifically, we will generate a pulse on the *CamTrig device* every N frames.

A working copy of this can be found at http://github.com/motmot/fview_periodic_trigger/

Prerequisites

- read and understand *FView plugin tutorial: live histogram of image intensity values*
- the *CamTrig hardware device* plugged in and functioning
- `fview_ext_trig` (software for the CamTrig) installed and functioning

Introduction

From *FView plugin tutorial: live histogram of image intensity values*, you should have a working knowledge of how to create a Python package directory structure and a `setup.py` file with the appropriate entry point to make an FView plugin. Therefore, this tutorial will focus only on the unique aspects of periodic triggering.

Create the plugin logic

Now we're going to create the module `motmot.fview_periodic_trigger.fview_periodic_trigger` with our new class `FviewPeriodicTrigger`. Open a new file named:

```
base/motmot/fview_periodic_trigger/fview_periodic_trigger.py
```

The contents of this file:

```
from __future__ import with_statement, division

import pkg_resources
import enthought.traits.api as traits

import motmot.fview.traited_plugin as traited_plugin
import motmot.fview_ext_trig.ttrigger as ttrigger

from enthought.traits.ui.api import View, Item, Group

class FviewPeriodicTrigger(traited_plugin.HasTraits_FViewPlugin):
    plugin_name = 'periodic trigger'
    trigger_device = traits.Instance(ttrigger.DeviceModel)
```

```

Nth_frame = traits.Int(100)

traits_view = View(Group(Item(name='Nth_frame'))))

def set_all_fview_plugins(self, plugins):
    """Get reference to 'FView external trigger' plugin"""

    # This method is called by FView to let plugins know about
    # each other.

    for plugin in plugins:
        if plugin.get_plugin_name()=='FView external trigger':
            self.trigger_device = plugin.trigger_device
    if self.trigger_device is None:
        raise RuntimeError('this plugin requires "FView external trigger"')

def process_frame(self, cam_id, buf, buf_offset, timestamp, framenum):
    if framenum%self.Nth_frame == 0:
        if self.trigger_device is not None:

            # fire pulse on EXT_TRIG1
            self.trigger_device.ext_trig1 = True

            # toggle LED
            self.trigger_device.led1 = not self.trigger_device.led1

    draw_points = []
    draw_linesegs = []
    return draw_points, draw_linesegs

```

The initial several lines are standard import statements. Note that this plugin uses Enthought's `Traits` module to facilitate event handling and GUI elements.

Next, we define the class `FviewPeriodicTrigger`, which is derived from the `traited_plugin.HasTraits_FViewPlugin` base class. This base class implements most of the functionality required for FView plugins, so we just have to implement or override a few things.

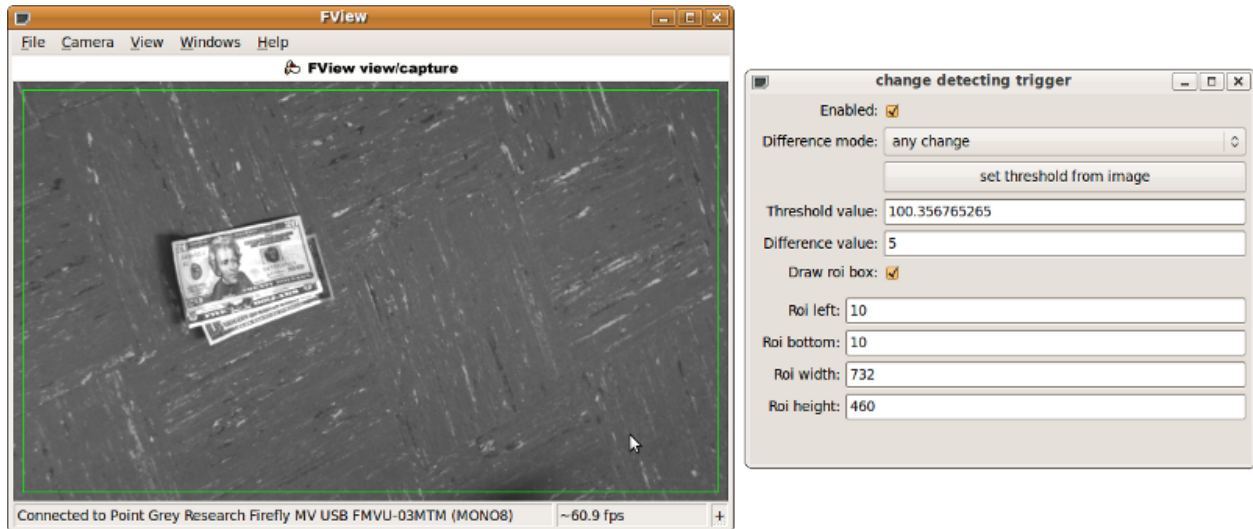
First, in our class, we give our plugin a name, in this case `periodic trigger`. Next, we tell traits that we're going to have a `trigger_device` variable and an `Nth_frame` variable. The `trigger_device` variable is an instance of the `motmot.fview_ext_trig.ttrigger.DeviceModel`, and is used to interact with the Motmot CamTrig hardware. The `Nth_frame` variable defines how frequently we will pulse the external trigger pin (EXT_TRIG1) and toggle the LED.

The `set_all_fview_plugins` method is required because we need to find the CamTrig trigger device. We do this by checking each of the plugins registered with FView to see if it is the 'FView external trigger' plugin. This plugin will have an attribute called `trigger_device`, which we want to keep a reference to.

Finally, the business end of this plugin, like most FView plugins, is the `process_frame` method. This method gets called on every frame and can be used to do realtime image analysis. We're keeping things simple in this tutorial, however, and only testing the framecount and pulsing the external trigger and toggling the LED if it's a multiple of the `Nth_frame` variable. The return value of `process_frame` are any points and line segments that FView should draw over the main display. In our case, we don't want to draw anything, so we return a couple of empty lists.

6.3.3 FView plugin tutorial: change-based triggering of an external device

This tutorial illustrates the steps necessary to trigger an external device based on some change in the image. Specifically, we will generate a pulse on the *CamTrig device* when luminance changes more than a specified amount.



A working copy of this code can be found at http://github.com/motmot/fview_change_trigger/

Prerequisites

- read and understand *FView plugin tutorial: live histogram of image intensity values*
- the *CamTrig hardware device* plugged in and functioning
- `fview_ext_trig` (software for the CamTrig) installed and functioning

Introduction

From *FView plugin tutorial: live histogram of image intensity values*, you should have a working knowledge of how to create a Python package directory structure and a `setup.py` file with the appropriate entry point to make an FView plugin. Therefore, this tutorial will focus only on the unique aspects of change-based triggering.

Create the plugin logic

Now we're going to create the module `motmot.fview_change_trigger.fview_change_trigger` with our new class `FviewChangeTrigger`. Open a new file named:

```
base/motmot/fview_change_trigger/fview_change_trigger.py
```

The contents of this file:

```
from __future__ import with_statement, division

import pkg_resources
import warnings, threading
import enthought.traits.api as traits

import motmot.fview.traited_plugin as traited_plugin
import motmot.fview_ext_trig.ttrigger as ttrigger
import numpy as np

from enthought.traits.ui.api import View, Item, Group
```

```

class FviewChangeTrigger(traited_plugin.HasTraits_FViewPlugin):
    plugin_name = 'change detecting trigger'

    trigger_device = traits.Instance(tttrigger.DeviceModel)
    enabled = traits.Bool(False)

    capture_background = traits.Button
    _capture_background_notify = traits.Bool(False) # pass value to realtime thread

    difference_mode = traits.Trait('darker', 'lighter', 'any change' )

    draw_roi_box = traits.Bool(False)

    roi_left = traits.Int(-1)
    roi_bottom = traits.Int(-1)
    roi_width = traits.Int(-1)
    roi_height = traits.Int(-1)

    threshold_value = traits.Float
    difference_value = traits.Float(5)

    # Store some values about the camera
    camera_cam_id = traits.String(transient=True)
    camera_max_width = traits.Int(transient=True)
    camera_max_height = traits.Int(transient=True)

    traits_view = View(Group(Item(name='enabled'),
                             Item(name='difference_mode'),
                             Item(name='capture_background',
                                   label='set threshold from image',
                                   show_label=False),
                             Item(name='threshold_value'),
                             Item(name='difference_value'),
                             Item(name='draw_roi_box'),
                             Group(Item(name='roi_left'),
                                   Item(name='roi_bottom'),
                                   Item(name='roi_width'),
                                   Item(name='roi_height'),
                                   )))

    def __init__(self, *args, **kwargs):
        super(FviewChangeTrigger, self).__init__(*args, **kwargs)

    def set_all_fview_plugins(self, plugins):
        """Get reference to 'FView external trigger' plugin"""

        # This method is called by FView to let plugins know about
        # each other.

        for plugin in plugins:
            if plugin.get_plugin_name()=='FView external trigger':
                self.trigger_device = plugin.trigger_device
            if self.trigger_device is None:
                raise RuntimeError('this plugin requires "FView external trigger"')

    def _capture_background_fired(self):
        self._capture_background_notify = True

```

```

def camera_starting_notification(self, cam_id,
                                pixel_format=None,
                                max_width=None,
                                max_height=None):

    if self.camera_cam_id != '':
        warnings.warn('FviewChangeTrigger only supports one camera')
        return
    self.camera_cam_id = cam_id
    self.camera_max_width = max_width
    self.camera_max_height = max_height

    # default margin ( in pixels )
    margin = 10
    if self.roi_left == -1:
        self.roi_left = margin
    if self.roi_bottom == -1:
        self.roi_bottom = margin
    if self.roi_width == -1:
        self.roi_width = self.camera_max_width - self.roi_left - margin
    if self.roi_height == -1:
        self.roi_height = self.camera_max_height - self.roi_bottom - margin

def process_frame(self, cam_id, buf, buf_offset, timestamp, framenum):
    draw_points = []
    draw_line_segs = []

    if cam_id != self.camera_cam_id:
        return draw_points, draw_line_segs

    l = self.roi_left
    r = l + self.roi_width
    b = self.roi_bottom
    t = b + self.roi_height

    if self.draw_roi_box:
        draw_line_segs.extend([ (l, b, l, t),
                                (l, t, r, t),
                                (r, t, r, b),
                                (r, b, l, b) ])

    npbuf = np.asarray(buf) # make sure it's a numpy array
    assert buf_offset == (0, 0)
    roi_buf = npbuf[b:t, l:r]

    if self._capture_background_notify:
        self._capture_background_notify = False
        self.threshold_value = np.mean(roi_buf)

    # turn off LED from any previous runs
    self.trigger_device.led1 = False

    if self.enabled:
        current_value = np.mean(roi_buf)
        fire_trigger = False
        if self.difference_mode == 'darker':
            if (self.threshold_value - current_value) > self.difference_value:
                fire_trigger = True

```

```
elif self.difference_mode == 'lighter':
    if (current_value - self.threshold_value) > self.difference_value:
        fire_trigger = True
elif self.difference_mode == 'any change':
    if abs(current_value - self.threshold_value) > self.difference_value:
        fire_trigger = True
else:
    raise ValueError('unknown difference_mode')

if fire_trigger:
    # fire pulse on EXT_TRIG1
    self.trigger_device.ext_trig1 = True

    # toggle LED
    self.trigger_device.led1 = True

return draw_points, draw_linesegs
```

A description of the above should go here.

CAMERA TRIGGER DEVICE WITH PRECISE TIMING AND ANALOG INPUT

7.1 camtrig – Camera trigger device firmware

camtrig - firmware for precisely timed trigger generation with synchronized analog input

7.1.1 Why

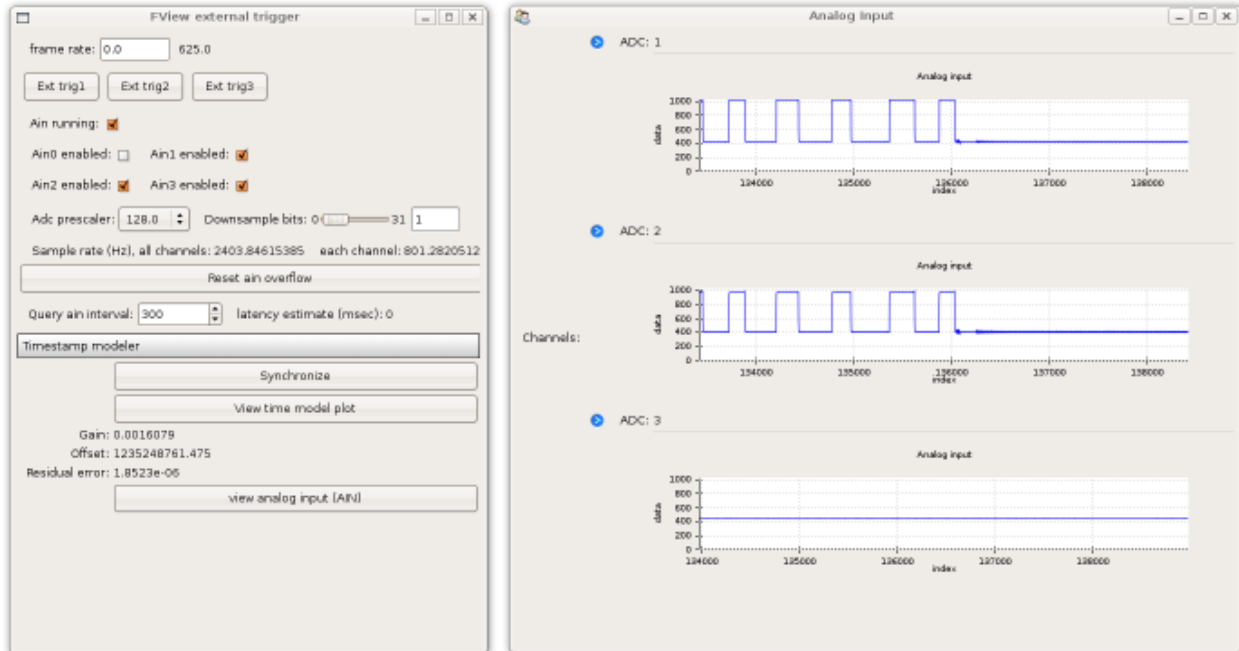
Triggering your camera is useful to synchronize it with other devices. These other devices can be other cameras, so that images are taken simultaneously, or a computer, so that the images can be correlated with other activity.

7.1.2 What



Camtrig is firmware for the \$30 [AT90USBKEY](#) USB development board that:

1. generates trigger pulses to feed into the external trigger input of digital video cameras. The pulses are generated with a 16-bit hardware timer using an 8 MHz crystal oscillator to produce very regular timing.
2. communicates synchronization information with software running on a PC. By repeatedly querying for timestamps from the USB device, the PC is able to make a model of the gain and offset of the two clocks with computed precision.
3. acquires analog voltage streams. The AT90USBKEY has a multiplexed 10-bit analog-to-digital converter (ADC), which can sample from 0.0 to 3.3 volts and operates up to 9.6 KHz using `motmot.fview_ext_trig`.
4. produces digital pulses to trigger other hardware.
5. provides a GUI plugin to `fview` that includes a display like a strip-chart recorder. This plugin is `motmot.fview_ext_trig`.



7.1.3 How

The device is accessed using the Python `motmot.fview_ext_trig` package.

Camtrig is built with GCC-AVR using the [LUFA](#) library for the AT90USBKEY. To load the firmware onto the device, use `dfu-programmer` or `FLIP` to transfer the hex file `camtrig.hex` to the device in Device Firmware Upload (DFU) mode.

7.1.4 Where

The source code for the camera trigger device is kept in the `fview_ext_trig` git repository under the `CamTrigUSB` directory. Alternatively, download the released Python code at the [PyPI](#) page and the firmware, `camtrig.hex`, from [github](#).

7.1.5 More information

Motmot camera trigger – firmware build and install instructions

This directory contains the source code for the `motmot` camera trigger. As this firmware is built using the [LUFA](#) library, that library is included in this source code tree. The `motmot` camera trigger is in `Projects/MotmotCamTrig`. The rest of the code here is a direct copy of the [LUFA](#) repository.

Building

Note: the `.hex` file (encoded firmware machine code) is already precompiled as `camtrig.hex` (using `avr-gcc 4.3.2` on Ubuntu Jaunty). Thus, you don't have to compile this code unless you want to change something.

in linux To build the MotmotCamtrig firmware (camtrig.hex):

```
cd Projects
rm -f MotmotCamTrig/*.hex MotmotCamTrig/*.o
make -C MotmotCamTrig
```

in Windows A rough sketch is: compile with [WinAVR](#). Follow the [LUFA](#) directions for compilation.

Loading firmware onto AT90USBKEY

dfu-programmer – linux The firmware is loaded over the normal USB cable. You must boot your AT90USBKEY into DFU mode. After doing so enter these commands:

```
sudo dfu-programmer at90usb1287 erase
sudo dfu-programmer at90usb1287 flash MotmotCamTrig/camtrig.hex
sudo dfu-programmer at90usb1287 start
```

Atmel FLIP – Windows Use Atmel’s GUI app to load the firmware (file MotmotCamTrig/camtrig.hex) to the device.

Installation to OS

How to get USB device recognized by Linux To get your linux system to recognize the device copy the udev rules file to the appropriate location:

```
sudo cp 99-motmot-cam-trig.rules /etc/udev/rules.d/
```

This will automatically allow users of the group “video” to access the device without special permissions. These instructions assume the video group exists and you are already a member. To check this, type:

```
groups
```

If *video* appears in the output, you are already a member of the video group.

Finally, you will need for the above changes to take effect. The easiest way is to reboot your computer. If you want to avoid that, try this:

```
# Force udev to notice the new file specifying the group for the device
sudo /etc/init.d/udev reload
sudo /etc/init.d/udev restart

# Start a new shell with membership in the new group - this console only!
su $USER
# (Enter your password)
```

How to get USB device recognized by Windows Follow the instructions [here](#), although please note that I have not tried this. The Vendor ID is 0x1781 and the Product ID is 0x0baf for this USB device.

Wiring the motmot camera trigger

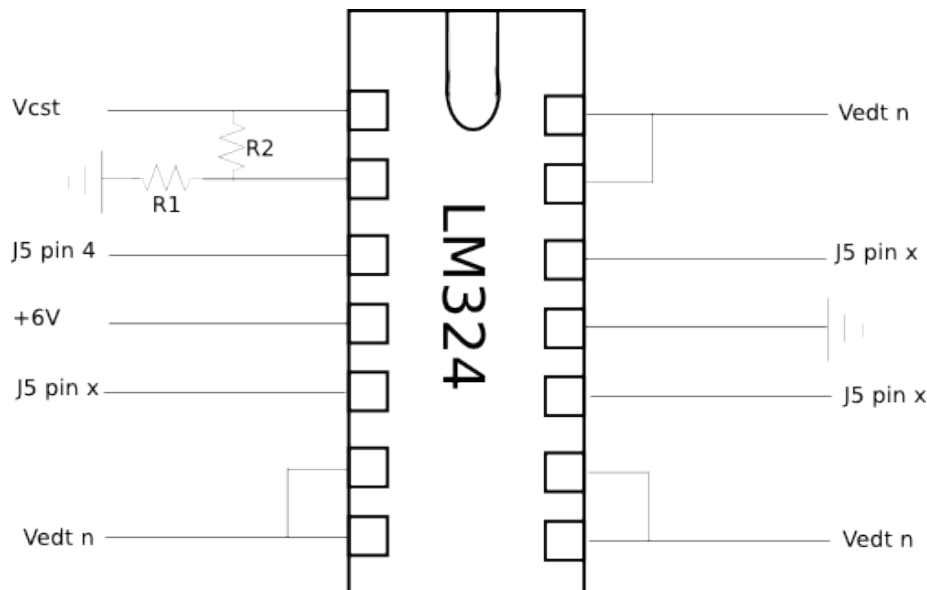
We have been connecting the AT90USBKEY in the following way. We hope to provide details on a canonical enclosure soon.

Connection details

Signal	AT90USBKEY contact	AT90USB1287 contact	Op-Amp
Camera Sync Trigger	J5 pin 4	C6, OCR3A	optional
External Device Trigger 1	J5 pin 9	C1	optional
External Device Trigger 2	J5 pin 8	C2	optional
External Device Trigger 3	J5 pin 7	C3	optional
Analog input 0	J1 pin 10	F0	
Analog input 1	J1 pin 9	F1	
Analog input 2	J1 pin 8	F2	
Analog input 3	J1 pin 7	F3	
Ground	J5 pin 2, J1 pin 2		

Optional 5V outputs

An optional Op-Amp may be used to boost the output signal voltage from 3.3V to 5.0V and provide more current. Furthermore, it should provide a degree of protection to the microcontroller from adverse connections.



In the drawing above, V_{CST} is the Camera Sync Trigger, V_{ETD_n} are the External Device Triggers. J5 refers to the jumper on the AT90USBKEY device. The resistors should be chosen to give the appropriate gain. Values of 100 ohms for R1 and 200 ohms for R2 will give a 3x gain, which will saturate the op-amp given a 6V power supply and a 3.3V input from the AT90USB device. The +6 V power is taken from an external power supply, which can be connected into the trigger box via the provided plug.

Example implementation

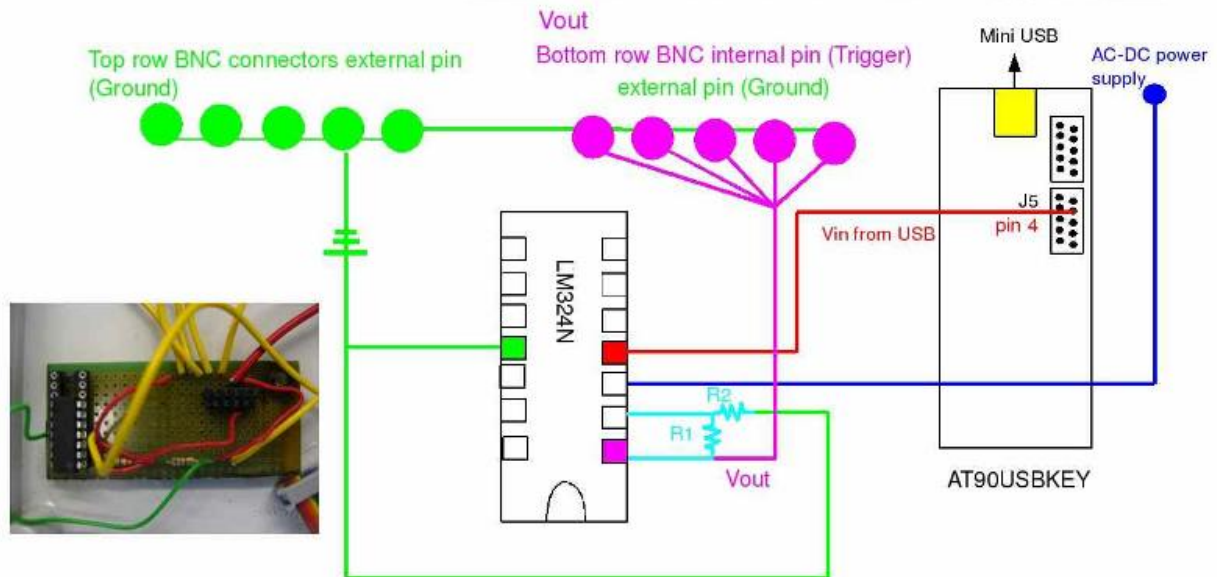
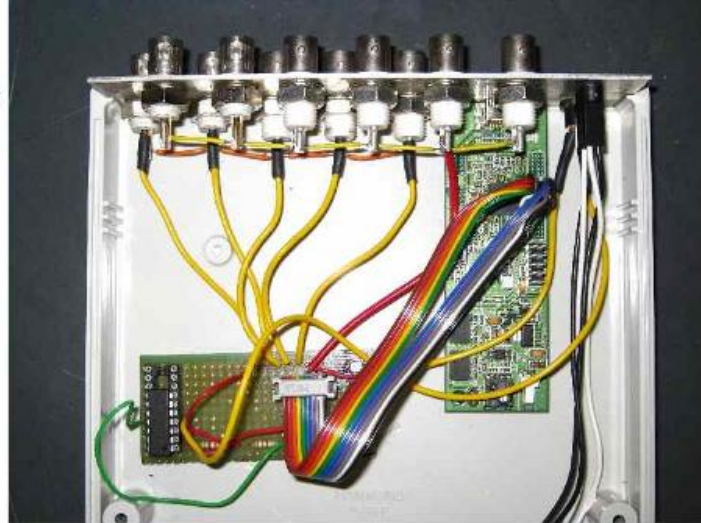
Andrew Straw built the trigger device pictured below, which was photographed and illustrated by Paolo Segre. It is meant to trigger multiple cameras from the Camera Sync Trigger pin, and thus brings this amplified signal out to the bottom row of BNC connectors. (The top row of BNCs is available for expansion and is currently unused.)

Motmot CamTrig example
courtesy Paolo Segre, UC Riverside

$$V_{out} = V_{in} \frac{R1+R2}{R1}$$

$V_{in} = 3.3$ volts

V_{out} is capped by DC power supply voltage



MOTMOT DEVELOPMENT

8.1 Code repository

The source code repository is hosted at <http://github.com/motmot/>.

To check out the source code using git, use:

```
git clone git://github.com/motmot/motmot.git
cd motmot
git submodule update --init
```

See the README.rst file in the cloned directory for more information.

8.2 Buildbot

We maintain a buildbot that continually tests the latest versions of the motmot packages on several systems. You may see the results on the [motmot buildbot waterfall display](#).

8.3 Email list

If you're interested in helping develop motmot, please join our [email list](#).

PACKAGES REFERENCE

Below are links to the package-level documentation for components of motmot. To see how these all fit together, look at the [overview](#).

9.1 libcamiface - camera interface C API

9.1.1 Overview

libcamiface (“camera interface”) is a C API that provides a camera and OS independent image acquisition framework. There is also a Python wrapper ([cam_iface](#)) of the libcamiface libraries.

9.1.2 What cameras are supported?

Camera support is determined by a libcamiface “backend”. Each backend is a piece of code that creates the libcamiface interface for each particular camera driver.















As seen below, the supported camera drivers are currently libdc1394, the Prosilica Gigabit Ethernet SDK, Point Grey Research FlyCapture2, and the QuickTime SequenceGrabber. For a list of cameras supporting the libdc1394 software, see <http://damien.douxchamps.net/ieee1394/cameras/>. For the Prosilica Gigabit cameras, see <http://www.prosilica.com/>. For the Point Grey cameras, see <http://www.ptgrey.com>. The QuickTime SequenceGrabber supports any camera available through QuickTime. This includes the built-in cameras on Mac laptops and desktops.

9.1.3 What frame rates, image sizes, bit depths are possible?




This depends primarily on the camera and camera interface. A primary goal of the Motmot suite is not to limit the capabilities possible. The fastest frame rates that we routinely use are 500 fps with a Basler A602f camera using a small region of interest and 200 fps on a Prosilica GE 680 at full frame resolution. The largest image size we routinely use is 1600x1200 on a Pt. Grey Scorpion camera. The largest bit depths we routinely use are 12 bits/pixel on a Basler A622f camera.

9.1.4 Backend status

A number of backends are supported.

Backend	GNU/Linux i386	GNU/Linux x86_64 (amd64)	win32 (XP)	Mac OS X
libdc1394			 newest library version supports Windows, but untested with libcamiface	 triggering options disabled
Prosilica GigE Vision				
Point Grey Research FlyCapture2	 untested	 untested		NA
ImperX	NA	NA	 rudiments present in git 'cruft' branch	NA
Basler BCAM 1.8	NA	NA	 rudiments present in git 'cruft' branch, frequent BSOD	NA
QuickTime SequenceGrabber	NA	NA	NA (supported by driver?)	 basic functionality works

Key to the above symbols:

-  Works well, no known bugs or missing features
-  Mostly works, some missing features
-  **Not working, although with some effort, could presumably be made to work**
- NA The upstream driver does not support this configuration

9.1.5 Download

Download official releases from [the download page](#).

9.1.6 Install from binary releases

Mac OS X

The Mac installer is called `libcamiface-x.y.z-Darwin.dmg`.

This will install the files:

```
/usr/include/cam_iface.h
/usr/bin/liveview-glut-*
/usr/bin/ ( other demos )
/usr/lib/libcam_iface_*
```

To run a demo program, open `/usr/bin/liveview-glut-mega`.

Windows

The Windows installer is called `libcamiface-x.y.z-win32.exe`.

This will install the files:

```
C:\Program Files\libcamiface x.y.z\bin\simple-prosilica_gige.exe
C:\Program Files\libcamiface x.y.z\bin\liveview-glut-prosilica_gige.exe
C:\Program Files\libcamiface x.y.z\include\cam_iface.h
C:\Program Files\libcamiface x.y.z\lib\cam_iface_prosilica_gige.lib
C:\Program Files\libcamiface x.y.z\bin\cam_iface_prosilica_gige.dll
```

To run a demo program, open `C:\Program Files\libcamiface x.y.z\bin\liveview-glut-prosilica_gige.exe`

9.1.7 Compile from source

On all platforms, you need to install `cmake`. `cmake` is available from <http://www.cmake.org/>

You will also need the libraries for any camera software. `Cmake` should automatically find these if they are installed in the default locations.

linux

```
mkdir build
cd build
cmake ..
make
make install
```

To build with debug symbols, include the argument `-DCMAKE_BUILD_TYPE=Debug` in your call to `cmake`. To install in `/usr`, include `-DCMAKE_INSTALL_PREFIX=/usr`. To make verbose makefiles, include `-DCMAKE_VERBOSE_MAKEFILE=1`.

To cut a source release:

```
VERSION="0.6.2"
git archive --prefix=libcamiface-$VERSION/ release/$VERSION | gzip -9 > ../libcamiface-$VERSION.tar.gz
git archive --prefix=libcamiface-$VERSION/ --format=zip release/$VERSION > ../libcamiface-$VERSION.zip
```

To make a Debian source package:

```
VERSION="0.6.2"
ln ../libcamiface-$VERSION.tar.gz ../libcamiface_$VERSION.orig.tar.gz
rm -rf ../libcamiface_*.orig.tar.gz.tmp-nest
git-buildpackage --git-debian-branch=debian --git-upstream-branch=master --git-no-create-orig --git-t
```

Mac OS X

Download and install Apple's XCode. This requires signing up (free) as an Apple ADC member.

```
mkdir build
cd build
cmake ..
```

```
make
cpack
```

To build with debug symbols, include the argument `-DCMAKE_BUILD_TYPE=Debug` in your call to `cmake`.

In fact, I use the following commands to set various environment variables prior to my call to `cmake`:

```
# You will doubtless need to change these to match your system
export PROSILICA_CMAKE_DEBUG=1
export PROSILICA_TEST_LIB_PATHS=/Prosilica\ GigE\ SDK\lib-pc\x86\4.0
export GLEW_ROOT="/Users/astrow/other-peoples-src/glew/glew-1.5.1"
```

This will build a Mac installer, called `libcamiface-x.y.z-Darwin.dmg`.

To build an Xcode project, run `cmake` with the argument `-DCMAKE_GENERATOR=Xcode`.

Windows

Windows XP 32bit with CMake 2.6

Install Microsoft's Visual Studio 2008. (Tested with Express Edition.) Install CMake.

Open a Visual Studio Command Prompt from Start Menu->All Programs->Microsoft Visual C++ 2008 Express Edition->Visual Studio Tools->Visual Studio 2008 Command Prompt. Change directories into the `libcamiface` source directory.

```
cmakesetup
rem In the cmakesetup GUI, set your source and build directories.
rem Click "configure".
rem In the "Select Generator" menu that pops up, press "NMake Makefiles".
rem After it's done configuring, click "configure" again.
rem Finally, click "OK".

rem Now change into your build directory.
cd build
nmake

rem Now, to build an NSIS .exe Windows installer.
cpack
```

This will build a Windows installer, called `libcamiface-x.y.z-win32.exe`.

Windows 7 64bit with CMake 2.8 to make 32 bit libcamiface

Install Microsoft's Visual Studio 2008. Install CMake.

Open a Visual Studio Command Prompt from Start Menu->All Programs->Microsoft Visual Studio 2008->Visual Studio Tools->Visual Studio 2008 Command Prompt. Change directories into the `libcamiface` source directory.

```
cd build
"C:\Program Files (x86)\CMake 2.8\bin\cmake.exe" .. -G "NMake Makefiles"
nmake

rem Now, to build an NSIS .exe Windows installer.
cpack
```


This will build a Windows installer, called `libcamiface-x.y.z-win32.exe`.

9.1.8 Backend notes

prosilica_gige

Here is an example of setting attributes on the camera using Prosilica's command line tools:

```
export CAM_IP=192.168.1.63
CamAttr -i $CAM_IP -s StreamBytesPerSecond 123963084
CamAttr -i $CAM_IP -s PacketSize 1500
```

Environment variables:

- *PROSILICA_BACKEND_DEBUG* print various debugging information.

libdc1394

Environment variables:

- *DC1394_BACKEND_DEBUG* print libdc1394 error messages. (You may also be interested in libdc1394's own *DC1394_DEBUG* environment variable, which prints debug messages.)
- *DC1394_BACKEND_1394B attempt to force use of firewire*
800. (Otherwise defaults to 400.)
- *DC1394_BACKEND_AUTO_DEBAYER use dc1394 to de-Bayer the images*, resulting in RGB8 images (rather than MONO8 Bayer images).

9.1.9 Git source code repository

The development version of libcamiface may be downloaded via git:

```
git clone git://github.com/motmot/libcamiface.git
```

9.1.10 License

libcamiface is licensed under the BSD license. See the LICENSE.txt file for the full description.

9.2 cam_iface – Python wrapper of libcamiface

This is a ctypes wrapper of *libcamiface - camera interface C API*. It requires libcamiface 0.5.9 or higher.

9.2.1 Installation

The usual:

```
python setup.py install
```

9.2.2 License

pycamiface is licensed under the BSD license. See the LICENSE.txt file for the full description.

9.2.3 Example usage

A very simple example of usage is in the file `demo/very_simple.py`. This file contains the following:

```
# Also the example in ../README.rst -- so keep in sync
import pkg_resources
import motmot.cam_iface.cam_iface_ctypes as cam_iface
import numpy as np

mode_num = 0
device_num = 0
num_buffers = 32

cam = cam_iface.Camera(device_num, num_buffers, mode_num)
cam.start_camera()
frame = np.asarray(cam.grab_next_frame_blocking())
print 'grabbed frame with shape %s'%(frame.shape,)
```

9.3 motmot.fview_ext_trig - camera trigger device

This package provides Python access to the *motmot camera trigger device*. There are three modules within this package:

- `motmot.fview_ext_trig.ttrigger` – This is a Python module with `traited` classes to interact with the *camtrig* device. (lowest level)
- `motmot.fview_ext_trig.live_timestamp_modeler` – This module provides a class, `LiveTimestampModeler`, that links the clocks of the camera and the host computer, keeping a running update of the relationship between the computer's clock and the current framenummer on the camera. The `LiveTimestampModelerWithAnalogInput` class also provides a time-locked data stream on up to four channels. (middle level)
- `motmot.fview_ext_trig.fview_ext_trig` – This model implements a plugin for *fview* to use the `LiveTimestampModelerWithAnalogInput` class. (high level)

9.3.1 Command-line commands

This package installs several command-line commands:

- **trigger_set_frequency** – Set the frequency of the camera sync trigger on the CamTrig device.
- **trigger_check_device** – Check for the presence of the CamTrig device. If the device is not found, it returns with a non-zero exit code.
- **trigger_enter_dfu_mode** – Reboot the CamTrig device into the DFU (Device Firmware Upload) mode. This can be used in conjunction with `dfu_programmer` or Atmel FLIP to flash the device with new firmware.

9.3.2 `motmot.fview_ext_trig.fview_ext_trig`

This Python module implements a class, `FviewExtTrig`, that provides an `fview` plugin to use the *camtrig* device. In the `fview_ext_trig` `setup.py` file, this class is registered as an `FView` plugin (see *writing FView plugins*).

`fview_ext_trig` requirements

`traits`, `remote_traits`, `pylibusb`

To use the `fview` plugin, you will also need `chaco`.

9.4 `fview` - Extensible realtime image viewing, saving, and analysis app

9.4.1 Overview and Usage Guide

fview is an application to view and record data from uncompressed digital video cameras. The name “fview” derives from “fly viewer” – the software was developed within the [Dickinson Lab](#) at [Caltech](#) to record movies of flies.

See the *Gallery* for some screenshots.

9.4.2 Features

- **Plugins for realtime image analysis** – Plugins to perform realtime image analysis are straightforward to write, with templates included to get you started quickly. Plugins have been written, for example, to perform background subtraction very quickly by making use of Intel’s Integrated Performance Primitives (IPP) library. See *writing FView plugins*.
- **camera trigger device with precise timing and analog input** – see *this page*
- **Many supported cameras** – `fview` uses *libcamiface - camera interface C API* to interact with cameras. This means that if you use `fview`, your code is independent from the particular camera hardware you’re using.
- **Written in Python** – `Python` is used as the “glue” that hold the application together – the underlying image processing and saving is performed by high performance C code. Flexible memory allocation is possible for easy integration with other languages and libraries.

9.4.3 Running `fview`

`Fview` has options which can be set via environment variables. These are:

`FVIEW_NO_REDIRECT`

Set to non-zero to direct all output to the console (linux) or to a pop-up window (Windows, Mac OS X). Otherwise, the default behavior of saving to `fview.log`.

`FVIEW_RAISE_ERRORS`

If this is non-zero, it will cause `FView` to raise an exception and thus close noisily, if any of its plugins raise exceptions. Otherwise, the default behavior of warning about the exception and continuing without the plugin will be used.

`FVIEW_NO_OPENGL`

Set to non-zero to disable use of OpenGL for image display. This will be slower and take more CPU time, but will avoid any potential bugs with the OpenGL side of things.

FVIEW_SAVE_PATH

Set to the directory name in which to record movies. (This can also be set with the Menu option “File->set record Directory...”.)

9.4.4 Current limitations

fview currently only supports a single camera. Although the plugin structure and the camera interface are inherently multi-camera ready, fview itself has not been written to support capturing from multiple cameras simultaneously. [Flydra](#), for example, makes use of multiple cameras using motmot.

9.5 flytrax - fview plugin for tracking 2D points in realtime

Flytrax is a plugin for [fview](#) to track a single point in realtime. It saves (X,Y) position, orientation, and small (spatially-cropped) movies that follow the tracked point. It uses background subtraction with optional online updating to deal with slowly changing lighting conditions.

See also *[fview and flytrax screenshot](#)*.

9.6 pyarrayimage – fast display of numpy arrays to OpenGL/pyglet

9.6.1 Description

pyarrayimage allows display of Python objects supporting the [array interface](#) as OpenGL textures without a copy. The OpenGL texture handling is done using [pyglet](#).

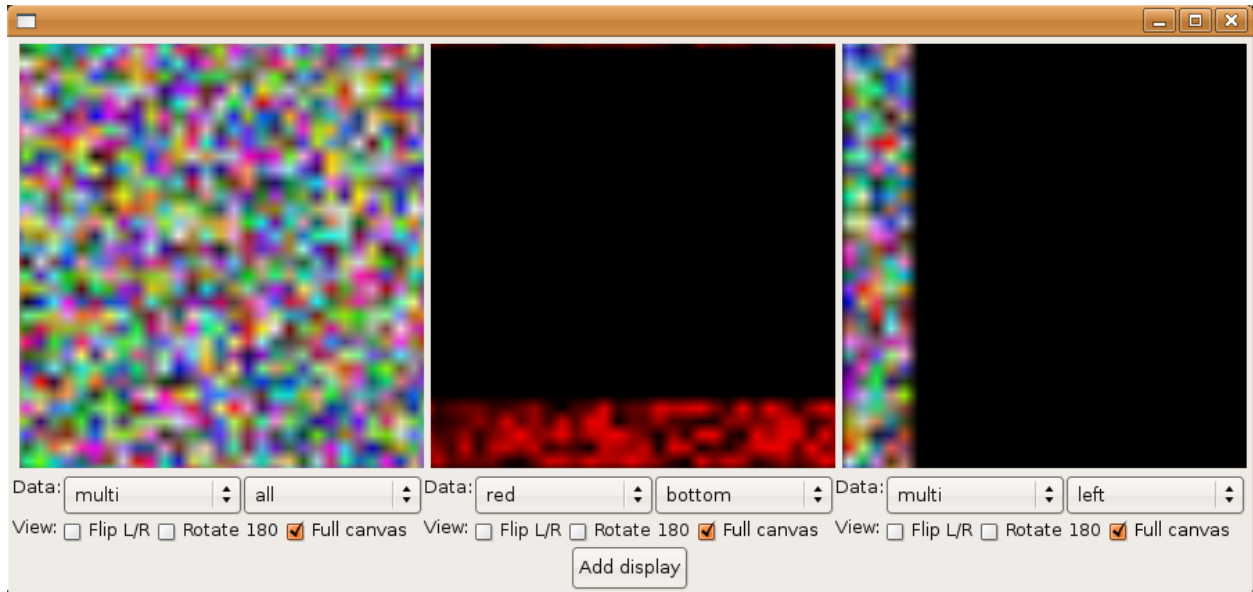
In other words, this allows fast transfer of data from numpy arrays (or any other data source supporting the array interface) to the video card.

9.6.2 Discussion

For the time being, all discussion for this software should happen on the [pyglet-users mailing list](#).

9.6.3 Screenshot

One use case of pyarrayimage is the [wxglvideo](#). Here is a screenshot of the [wxglvideo_demo](#) program:



9.6.4 Download

Download official releases from [the download page](#).

The development version of `pygarrayimage` (raw files) may be downloaded via git:

```
git clone git://github.com/motmot/pygarrayimage.git
```

9.7 motmot.FlyMovieFormat - simple, uncompressed movie storage format

This package contains code to read, write and view FlyMovieFormat files, which end with extension `.fmf`.

The primary design goals of FlyMovieFormat were:

- Single pass, low CPU overhead writing of lossless movies for realtime streaming applications
- Precise timestamping for correlation with other activities
- Simple format that can be read from Python, C, and MATLAB®.

These goals are achieved via using a very simple format. After an initial header containing meta-data such as image size and color coding scheme (e.g. monochromatic 8 bits per pixel, YUV422, etc.), repeated chunks of raw image data and timestamp are saved. Because the raw image data from the native camera driver is saved, no additional processing is performed. Thus, streaming of movies from camera to disk will keep the CPU free for other tasks, but it will require a lot of disk space. Furthermore, the disk bandwidth required is equivalent to the camera bandwidth (unless you save only a region of the images, or if you only save a fraction of the incoming frames).

9.7.1 FlyMovieFormat (.fmf) file specification

This file type defines 8-bit grayscale image sequences where each image is associated with a timestamp. There are currently two versions implemented, versions 1 and 3. (Version 2 was briefly used internally and is now

best forgotten.) For documentation about the purpose of the format and the reference Python implementation, see `motmot.FlyMovieFormat.FlyMovieFormat`.

File structure

- header (either version 1 or version 3)
- arbitrary number of timestamp and frame pairs (“frame chunks”)

.fmf header version 1

Only supports MONO8 format.

Typecode	Name	Description
I	version	Version number (1)
II	framesize	Number of rows and columns in each frame
Q	chunksize	Bytes per “chunk” (timestamp + frame)
Q	n_frames	Number of frames (0=unknown, read file to find out)

.fmf header version 3

Typecode	Name	Description
I	version	Version number (3)
I	lenformat	Length of the subsequent format string
*B	format	string containing format, e.g. ‘MONO8’ or ‘YUV422’
I	bpp	Bits per pixel, e.g. 8
II	framesize	Number of rows and columns in each frame
Q	chunksize	Bytes per “chunk” (timestamp + frame)
Q	n_frames	Number of frames (0=unknown, read file to find out)

.fmf frame chunks

Typecode	Name	Description
d	timestamp	Timestamp (seconds in current epoch)
*B	frame	Image data, rows*columns bytes, row major ordering

Typecodes used above

All numbers are little-endian (Intel standard).

Typecode	description	bytes	C type
B	uint8	1	unsigned char
I	uint32	4	unsigned int
Q	uint64	8	unsigned long long (__int64 on Windows)
d	double64	8	double
*B	data		an unsigned char buffer of arbitrary length

9.7.2 Converting files to .fmf

The command **ffmpeg2fmf** will use avbin (part of [Pyglet](#)) to read a variety of movie formats and save them as .fmf files. Use it like so:

```
ffmpeg2fmf --format=mono8 /path/to/my/movie.mp4
# or
ffmpeg2fmf --format=mono8 /path/to/my/movie.avi
```

This will save a file */path/to/my/movie.fmf*.

9.7.3 motmot.FlyMovieFormat.FlyMovieFormat

This module contains code to read and write FlyMovieFormat files, which end with extension .fmf.

Users may like to use these classes:

- `FlyMovie` : read .fmf files
- `FlyMovieSaver` : write .fmf files

The following Exception types may sometimes be raised:

- `NoMoreFramesException` :
- `InvalidMovieFileException` :

Reading an .fmf from Python

In Python accessing frames from an .fmf files is as easy as:

```
import motmot.FlyMovieFormat.FlyMovieFormat as FMF
import sys

fname = sys.argv[1]
fmf = FMF.FlyMovie(fname)
for frame_number in range(10):
    frame,timestamp = fmf.get_frame(frame_number)
```

Writing an .fmf in Python

Writing frames to an .fmf file is also easy:

```
import motmot.FlyMovieFormat.FlyMovieFormat as FMF
import sys
import numpy as np

fname = sys.argv[1]
fmf_saver = FMF.FlyMovieSaver(fname)

width, height = 640,480
for i in range(10):
    fake_image = np.zeros( (height,width), dtype=np.uint8)
    fake_timestamp = 0.0
    fmf_saver.add_frame(fake_image,fake_timestamp)
```

```
fmf_saver.close()
```

Module documentation

class FlyMovie (*file, check_integrity=False*)

.fmf file reader

- *file* : string or file object to open

Optional keyword arguments:

- *check_integrity* : whether to scan the last frame(s) for completeness

get_all_timestamps ()

return all timestamps in .fmf file

Returns: - *all_timestamps* : array of timestamps

get_bits_per_pixel ()

get the number of bits per pixel

Returns: - *bits_per_pixel* : integer, number of bits per pixel (e.g. 8)

get_height ()

returns height of data, in pixels

Returns: - *height* : integer, height of image, in pixels

get_n_frames ()

get the number of frames

Returns: - *n_frames* : integer, number of frames

get_next_frame (*allow_partial_frames=False*)

return image data for frame number

Returns:

- *frame* : array of image data
- *timestamp* : float, timestamp of image time

get_width ()

returns width of data, in bytes

Returns: - *width* : integer, width of image, in bytes

Note, to get the width of underlying image, do this:

```
image_width = fmf.get_width() // (fmf.get_bits_per_pixel() // 8)
```

seek (*frame_number*)

go to frame number

class FlyMovieSaver (*file, version=1, seek_ok=True, format=None, bits_per_pixel=None*)

.fmf file writer

- *file* : string or file object to open

Keyword arguments:

- *version* : 1 or 3, defaults to 1

- seek_ok* : boolean indicating whether calling seek() is OK on this file
- format* : format string (e.g. 'MONO8', required for version 3 only)
- bits_per_pixel* : number of bits per pixel. inferred from format if not supplied.

add_frame (*origframe*, *timestamp=nan*, *error_if_not_fast=False*)

save a single image frame to the file

- origframe* : array of image data

Optional keyword arguments:

- timestamp* : double precision floating point timestamp (default: nan)
- error_if_not_fast* : boolean: raise error if no origframe.dump_to_file()

add_frames (*frames*, *timestamps=None*)

add multiple image frames

- frames* : arrays of image data

Optional keyword arguments:

- timestamps* : double precision floating point timestamps

close ()

finish writing the file

exception InvalidMovieFileException

Bases: `exceptions.Exception`

The file is not a valid movie file

exception NoMoreFramesException

Bases: `exceptions.Exception`

A frame was requested, but no more frames exist

mmap_flymovie (**args*, ***kwargs*)

map .fmf file to RAM

Note: make mmap does not currently work with RGB images.

Note: to map a 4 GB or larger file to RAM, a 64 bit computer is required.

9.8 pygxinput – use XInput devices in pyglet

9.8.1 Description

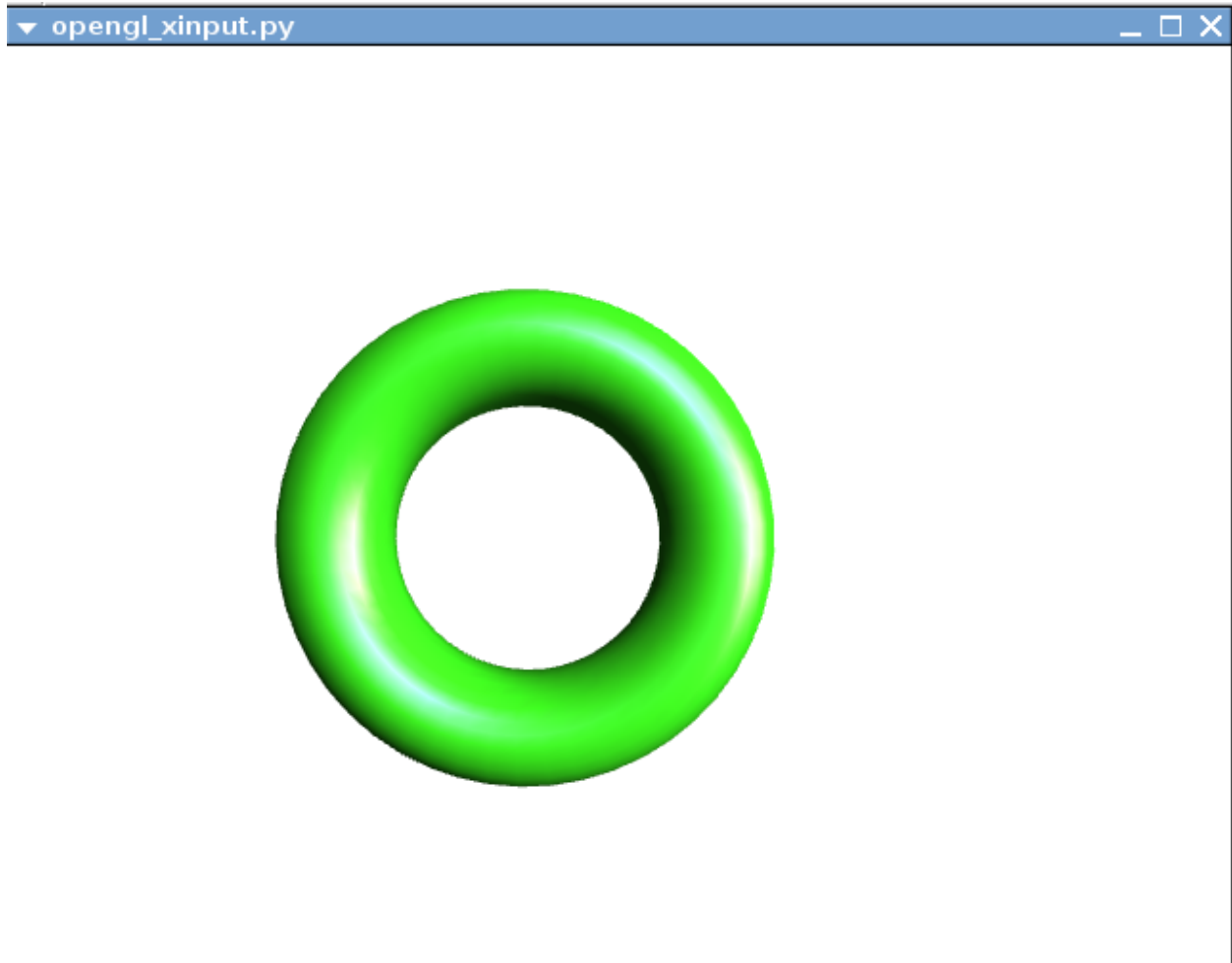
pygxinput allows use of XInput devices in pyglet. Such devices include 3Dconnexion Space Navigator and Wacom tablets.

For information about how to setup the Space Navigator as an XInput device, see <http://www.fedorawiki.de/index.php/SpaceNavigator> (in German).

It is provided under a BSD license by author Andrew Straw.

9.8.2 Screenshot

Screenshot of example program `opengl_xinput.py` demonstrating use of a 3Dconnexion Space Navigator to control the movement of a torus in 3D.



9.8.3 Download

The development version of `pygxinput` (raw files) may be downloaded via git:

```
git clone git://github.com/motmot/pygxinput.git
```

9.8.4 Discussion

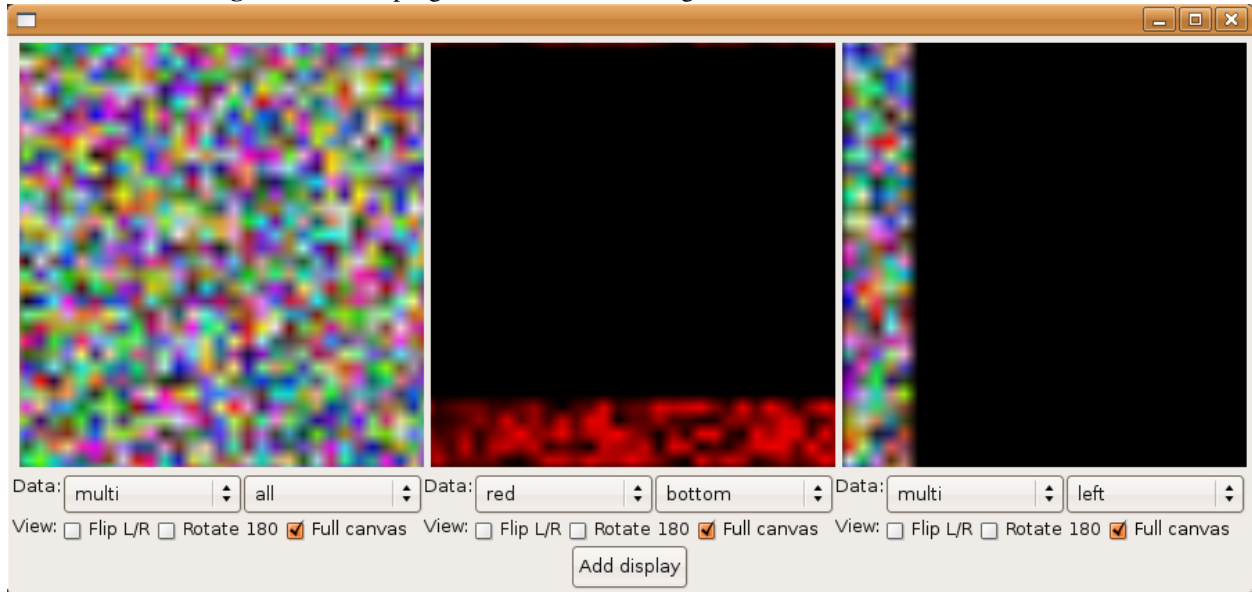
For the time being, all discussion for this software should happen on the [pyglet-users mailing list](#).

9.9 motmot.wxglvideo – CPU friendly display of uncompressed video using wxPython

The wxglvideo package allows rapid display of numpy arrays into wxPython OpenGL contexts. In particular, it defines a class `DynamicImageCanvas`, which is a subclass of `wx.glcanvas.GLCanvas` into which arrays are blitted. By using the `pyarrayimage` module, it is possible to enforce that no copy is made of the data on its way to OpenGL, ensuring minimal resource use.

See also `motmot.wxvideo.wxvideo` for a similar module that does not make use of OpenGL.

Screenshot of the `wxglvideo_demo` program, included with wxglvideo:



9.9.1 motmot.wxglvideo.wxglvideo

```
class DynamicImageCanvas(*args, **kw)
```

Bases: `wx.glcanvas.GLCanvas`

Display image data to OpenGL using as few resources as possible

```
set_flip_LR(value)
```

update the view transformation to include a left-right image flip for all images

Arguments val : boolean

Whether to flip the image

```
set_rotate_180(value)
```

update the view transformation to include a 180 degree rotation for all images

Arguments

val [boolean] Whether to rotate the image

```
update_image(image)
```

update the image to be displayed

9.9.2 motmot.wxglvideo.simple_overlay

class DynamicImageCanvas (*args, **kw)

Bases: wx._windows.Panel

This class mimics the behavior of `motmot.wxvideo.wxvideo.DynamicImageCanvas`, allowing multiple image sources and the simple overlay of points and lines.

set_flip_LR (value)

update the view transformation to include a left-right image flip for all images

Arguments val : boolean

Whether to flip the image

set_rotate_180 (value)

update the view transformation to include a 180 degree rotation for all images

Arguments

val [boolean] Whether to rotate the image

update_image_and_drawings (id_val, image, format='MONO8', points=None, point_colors=None, linesegs=None, lineseg_colors=None, xoffset=0, yoffset=0, sort_add=False)

update the displayed image

Arguments

id_val [string] An identifier for the particular source being updated

image [numpy array] The image data to update

Optional keyword arguments

format [string] The image format (e.g. 'MONO8', 'RGB8', or 'YUV422')

points [list of points] Points to display (e.g. [(x0,y0),(x1,y1)])

linesegs [list of line segments] Line segments to display (e.g. [(x0,y0,x1,y1),(x1,y1,x2,y2)])

class PointDisplayCanvas (*args, **kw)

Bases: `motmot.wxglvideo.wxglvideo.DynamicImageCanvas`

A single image view with overlaid points and line segments

copy_array_including_strides (arr)

copy a numpy array, including the strides

9.10 motmot.wxvideo – display of uncompressed video using wxPython

The wxvideo package allows display of numpy arrays into wxPython OpenGL contexts. In particular, it defines a class `DynamicImageCanvas`, which is a subclass of `wx.glcanvas.GLCanvas` into which arrays are blitted.

See also `motmot.wxglvideo.simple_overlay` for hardware accelerated video display in a manner compatible with this module.

9.10.1 motmot.wxvideo.wxvideo

class DynamicImageCanvas (*args, **kw)

Bases: wx._core.Window

Display uncompressed video images

This class supports the display of multiple, side-by-side images. Each of these images is from a single source (a camera, for example), so multiple views can be displayed with one `DynamicImageCanvas` instance. Each source has an identity string `id_val` which is used when updating that view's image.

Simple overlay drawings are also possible. Points and lines may be drawn on top of the displayed images.

get_canvas_copy ()

get a copy of the current image as an RGB8 numpy array

gui_repaint (drawDC=None)

blit bitmap to drawing context

Optional keyword Arguments

drawDC [wx.PaintDC instance] The draw context into which to blit

set_flip_LR (val)

update the view transformation to include a left-right image flip for all images

Arguments val : boolean

Whether to flip the image

set_rotate_180 (val)

update the view transformation to include a 180 degree rotation for all images

Arguments

val [boolean] Whether to rotate the image

update_image_and_drawings (id_val, image, format=None, points=None, linesegs=None, point_colors=None, point_radii=None, lineseg_colors=None, lineseg_widths=None, xoffset=0, yoffset=0, doresize=None)

update the displayed image

Arguments

id_val [string] An identifier for the particular source being updated

image [numpy array] The image data to update

Optional keyword arguments

format [string] The image format (e.g. 'MONO8', 'RGB8', or 'YUV422')

points [list of points] Points to display (e.g. [(x0,y0),(x1,y1)])

linesegs [list of line segments] Line segments to display (e.g. [(x0,y0,x1,y1),(x1,y1,x2,y2)])

9.11 motmot.FastImage – SIMD image processing

9.11.1 Description

FastImage implements low-level image processing operations designed to operate very quickly by using SIMD instructions. This is achieved by calling the `Framework` library. A bridge to numpy is made through the `array interface`.

9.11.2 Allocation of aligned memory

For the SIMD instructions to perform at maximal speed, images must be aligned on 32-byte boundaries. FastImage relies on the underlying image processing library to allocate the memory, trusting that it knows best:

```
import motmot.FastImage.FastImage as FastImage
import numpy as np

# Allocate the image
fi_im1 = FastImage.FastImage8u( FastImage.Size(4,5) ) # width, height

# Get a numpy view
im1 = np.asarray( fi_im1 )
assert im1.shape == (5,4) # height, width
```

In the above example, `im1.strides` will be `(32,1)`, indicating that each row is aligned on a 32 byte boundary.

9.12 motmot.imops - manipulate image codings

This package contains code to convert between different image codings.

9.12.1 motmot.imops.imops

manipulate image codings

is_coding_color()
return whether a coding represents a color image

to_mono8()
convert image to MONO8 encoding

Arguments

format [string] a string specifying the input format (e.g. 'MONO8', 'YUV422', etc.)

image [array-like] the raw image data in the format specified

Returns

mono8 [array-like] the image data in MONO8 encoding

to_rgb8()
convert image to RGB8 encoding

Arguments

format [string] a string specifying the input format (e.g. 'MONO8', 'YUV422', etc.)

image [array-like] the raw image data in the format specified

Returns

rgb8 [array-like] the image data in RGB8 encoding

CITATIONS

If you use Motmot, please cite this paper:

Straw, A.D. and Dickinson, M.H. (2009) [Motmot, an open-source toolkit for realtime video acquisition and analysis. Source Code for Biology and Medicine](#) doi: 10.1186/1751-0473-4-5 [PDF](#)

If you publish work that used motmot, I (Andrew Straw) ask that you cite this paper. These citations will help my employment and funding situation and thus indirectly fund Motmot development and support.

10.1 Scientific papers which used the motmot suite of tools

John A. Bender and Michael H. Dickinson (2006) [Visual stimulation of saccades in magnetically tethered Drosophila](#). *Journal of Experimental Biology*.

John A. Bender and Michael H. Dickinson (2006) [A comparison of visual and haltere-mediated feedback in the control of body saccades in Drosophila melanogaster](#). *Journal of Experimental Biology*.

Seth A. Budick, Michael B. Reiser, and Michael H. Dickinson (2007) [The role of visual and mechanosensory cues in structuring forward flight in Drosophila melanogaster](#). *Journal of Experimental Biology*.

Maimon, Gaby, Straw, Andrew D. & Dickinson, Michael H. (2008) [A Simple Vision-Based Algorithm for Decision Making in Flying Drosophila](#). *Current Biology* 18(6): 464-470. doi: [10.1016/j.cub.2008.02.054](#)

Straw, A.D., Branson, K., Neumann, T.R., and Dickinson, M.H. (submitted to *Computer Vision and Image Understanding*) Multi-camera Realtime 3D Tracking of Multiple Flying Animals

Han, Shuo, Straw, Andrew D., Dickinson, Michael H. & Murray, Richard M. (2009) A Real-Time Helicopter Testbed for Insect-Inspired Visual Flight Control. *Proceedings of the 2009 IEEE Conference on Robotics and Automation*

K. Branson, A. Robie, J. Bender, P. Perona, and M.H. Dickinson (2009) High-throughput Ethomics in Large Groups of *Drosophila*. *Nature Methods* doi: [10.1038/nmeth.1328](#)

INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*

MODULE INDEX

C

`cam_iface`, 37

F

`flytrax`, 40

`fview`, 39

M

`motmot.FastImage.FastImage`, 49

`motmot.FlyMovieFormat`, 41

`motmot.FlyMovieFormat.FlyMovieFormat`,
43

`motmot.fview_ext_trig`, 38

`motmot.imops`, 50

`motmot.imops.imops`, 50

`motmot.wxglvideo`, 47

`motmot.wxglvideo.simple_overlay`, 48

`motmot.wxglvideo.wxglvideo`, 47

`motmot.wxvideo`, 48

`motmot.wxvideo.wxvideo`, 49

P

`pygarrayimage`, 40

`pygxinput`, 45

INDEX

A

add_frame() (motmot.FlyMovieFormat.FlyMovieFormat.FlyMovieSaver
method), 45

add_frames() (motmot.FlyMovieFormat.FlyMovieFormat.FlyMovieSaver
method), 45

analog input, 25, 38

C

cam_iface, 37
module, 37

cam_iface (module), 37

close() (motmot.FlyMovieFormat.FlyMovieFormat.FlyMovieSaver
method), 45

copy_array_including_strides() (in module mot-
mot.wxglvideo.simple_overlay), 48

D

DynamicImageCanvas (class in mot-
mot.wxglvideo.simple_overlay), 48

DynamicImageCanvas (class in mot-
mot.wxglvideo.wxglvideo), 47

DynamicImageCanvas (class in mot-
mot.wxvideo.wxvideo), 49

E

environment variable

- FVIEW_NO_OPENGL, 39
- FVIEW_NO_REDIRECT, 39
- FVIEW_RAISE_ERRORS, 39
- FVIEW_SAVE_PATH, 39

F

FastImage, 49
module, 49

FlyMovie (class in mot-
mot.FlyMovieFormat.FlyMovieFormat),
44

FlyMovieFormat, 41

FlyMovieSaver (class in mot-
mot.FlyMovieFormat.FlyMovieFormat),
44

flytrax, 40
module, 40

flytrax (module), 40

fview, 39
module, 39

fview (module), 39

fview_ext_trig, 25, 38
module, 38

G

get_all_timestamps() (mot-
mot.FlyMovieFormat.FlyMovieFormat.FlyMovie
method), 44

get_bits_per_pixel() (mot-
mot.FlyMovieFormat.FlyMovieFormat.FlyMovie
method), 44

get_canvas_copy() (mot-
mot.wxvideo.wxvideo.DynamicImageCanvas
method), 49

get_height() (motmot.FlyMovieFormat.FlyMovieFormat.FlyMovie
method), 44

get_n_frames() (motmot.FlyMovieFormat.FlyMovieFormat.FlyMovie
method), 44

get_next_frame() (mot-
mot.FlyMovieFormat.FlyMovieFormat.FlyMovie
method), 44

get_width() (motmot.FlyMovieFormat.FlyMovieFormat.FlyMovie
method), 44

gui_repaint() (motmot.wxvideo.wxvideo.DynamicImageCanvas
method), 49

I

imops, 50

InvalidMovieFileException, 45

is_coding_color() (in module motmot.imops.imops), 50

M

mmap_flymovie() (in module mot-
mot.FlyMovieFormat.FlyMovieFormat),
45

module

- cam_iface, 37
- FastImage, 49
- flytrax, 40
- fview, 39
- fview_ext_trig, 38
- motmot.FlyMovieFormat, 41
- motmot.imops, 50
- motmot.wxglvideo, 47
- motmot.wxvideo, 48
- pygarrayimage, 40
- pygxinput, 45
- motmot.FastImage.FastImage (module), 49
- motmot.FlyMovieFormat, 41
 - module, 41
- motmot.FlyMovieFormat (module), 41
- motmot.FlyMovieFormat.FlyMovieFormat (module), 43
- motmot.fview_ext_trig (module), 38
- motmot.imops
 - module, 50
- motmot.imops (module), 50
- motmot.imops.imops (module), 50
- motmot.wxglvideo
 - module, 47
- motmot.wxglvideo (module), 47
- motmot.wxglvideo.simple_overlay (module), 48
- motmot.wxglvideo.wxglvideo (module), 47
- motmot.wxvideo
 - module, 48
- motmot.wxvideo (module), 48
- motmot.wxvideo.wxvideo (module), 49

N

NoMoreFramesException, 45

P

- PointDisplayCanvas (class in mot-
mot.wxglvideo.simple_overlay), 48
- pygarrayimage, 40
 - module, 40
- pygarrayimage (module), 40
- pygxinput, 45
 - module, 45
- pygxinput (module), 45

S

- seek() (motmot.FlyMovieFormat.FlyMovieFormat.FlyMovie
method), 44
- set_flip_LR() (motmot.wxglvideo.simple_overlay.DynamicImageCanvas
method), 48
- set_flip_LR() (motmot.wxglvideo.wxglvideo.DynamicImageCanvas
method), 47
- set_flip_LR() (motmot.wxvideo.wxvideo.DynamicImageCanvas
method), 49

- set_rotate_180() (motmot.wxglvideo.simple_overlay.DynamicImageCanvas
method), 48
- set_rotate_180() (motmot.wxglvideo.wxglvideo.DynamicImageCanvas
method), 47
- set_rotate_180() (motmot.wxvideo.wxvideo.DynamicImageCanvas
method), 49
- synchronization, 25, 38

T

- to_mono8() (in module motmot.imops.imops), 50
- to_rgb8() (in module motmot.imops.imops), 50

U

- update_image() (motmot.wxglvideo.wxglvideo.DynamicImageCanvas
method), 47
- update_image_and_drawings() (mot-
mot.wxglvideo.simple_overlay.DynamicImageCanvas
method), 48
- update_image_and_drawings() (mot-
mot.wxvideo.wxvideo.DynamicImageCanvas
method), 49

W

- wxglvideo, 47
- wxvideo, 48