flydra Documentation

Release 0.7.0

Andrew Straw

Sep 01, 2023

Contents

Flydra is a realtime, multi-camera flying animal tracking system. See the paper for algorithmic details.

Contents:

CHAPTER 1

Getting started with flydra development

This document assumes you are using Ubuntu linux and that you have a basic command line knowledge such as how to change directories, list files, and so on. Flydra is written primarily in the Python language. The Python tutorial is highly recommended when getting started with Python.

1.1 Using a Python virtual environment

We use virtualenv to create an installation environment for flydra that does not overwrite any system files. Get the virtualenv source code, upack it and create a "virtual Python installation" in your home directory called "PY_flydra":

```
wget https://pypi.python.org/packages/source/v/virtualenv/virtualenv-1.3.tar.gz
tar xvzf virtualenv-1.3.tar.gz
cd virtualenv-1.3
python virtualenv.py ~/PY_flydra
```

1.2 Getting and installing flydra from the source code repository

To download the development version of flydra, you need git. To install it, run:

```
sudo apt-get install git-core gitk
```

You will need to send Andrew Straw your ssh key following these instructions.

Now, to download ("checkout") flydra into your current directory, type:

```
git clone git@code.astraw.com:flydra.git
```

To build and install flydra to your virtual Python installation:

```
sudo dpkg --purge python-flydra # remove system flydra to prevent confusion
source ~/PY_flydra/bin/activate
cd flydra
python setup.py develop
```

You should now have a working flydra installation in your virtual Python environment. To test this, type:

```
cd ~ # switch to a directory without a flydra/ subdirectory
python -c "import flydra_core.version; print flydra_core.version.__version__"
```

The output should be something like "0.4.28-svn", indicating this is the development version after release 0.4.28.

1.3 Testing your installation

Finally, the full test suite may be run with nose:

```
source ~/PY_flydra/bin/activate
easy_install nose
nosetests flydra
```

1.4 Editing the documentation

This documentation is built with Sphinx 0.6.2.

To download and install sphinx into your virtual environment:

```
easy_install sphinx
```

Now, to build the flydra documentation:

```
cd /path/to/flydra/flydra-sphinx-docs/
./get-svn.sh
make html
```

The documentation will be built in /path/to/flydra/flydra-sphinx-docs/.build/html/index.html You may view it with:

```
firefox .build/html/index.html
```

Alternatively, you can automatically build and upload it by pushing your changes to the central git repository and clicking the "Force build" button on the "doc uploader" build slave.

1.5 Continuous integration

We are running a Buildbot that builds and tests flydra. You can see the waterfall display at http://flydra.bb.astraw.com/waterfall .

CHAPTER 2

Calibration

Flydra calibration data (also, in the flydra source code, a *reconstructor*) consists of:

- parameters for a *linear pinhole camera model* (including intrinsic and extrinsic calibration).
- parameters for *non-linear distortion*.
- (optional) a *scale factor* and *units* (e.g. 1000.0 and millimeters). If not specified these default to 1.0 and meters. If these are specified they specify how to convert the native units into meters (the scale factor) and the name of the present units. (Meters are the units used the dyanmic models, and otherwise have no significance.)

See *Calibration files and directories - overview* for a discussion of the calibration file formats.

CHAPTER 3

Generating a calibration using MultiCamSelfCal

The basic idea is that you will save raw 2D data points which are easy to track and associate with a 3D object. Typically this is done by waving an LED around.

For this to work, the 2D/3D data association problem must be trivial. Only a single 3D point should be generating 2D points, and every 2D point should come from the (only) 3D point. (Missing 2D points are OK; occlusions *per se* do not cause problems, whereas insufficient or false data associations do.)

(3D trajectories, which can only come from having a calibration and solving the data association problem, will not be used for this step, even if available. If a calibration has already been used to generate Kalman state estimates, the results of this data association will be ignored.)

Typically, about 500 points distributed throughout the tracking volume will be needed for the MATLAB MultiCamSelfCal toolbox to complete successfully. Usually, however, this will mean that you save many more points and then sub-sample them later. See the `config.cal.USE_NTH_FRAME` parameter below.

3.1 Saving the calibration data in flydra_mainbrain

Start saving data normally within the **`flydra_mainbrain`** application. Remember that only time points with more than 2 cameras returning data will be useful, and that time points with more than 1 detected view per camera are useless.

Walk to the arena and wave the LED around. Make sure it covers the entire tracking volume, and if possible, outside the tracking volume, too.

3.2 Exporting the data for MultiCamSelfCal

Now, you have saved an .h5 file. To export the data from it for calibration, run:

```
flydra_analysis_generate_recalibration --2d-data DATAFILE2D.h5 \
  --disable-kalman-objs DATAFILE2D.h5
```

You should now have a new directory named `DATAFILE2D.h5.recal`. This contains the raw calibration data (synchronized 2D points for each camera) in a format that the MATLAB MultiCamSelfCal can understand, the calibration directory.

3.3 Running MultiCamSelfCal

NOTE: This section is out of date. The new version of MultiCamSelfCal does not require MATLAB and can be run purely with Octave. The new version of MultiCamSelfCal is available from https://github.com/strawlab/MultiCamSelfCal

Edit the file `kookaburra/MultiCamSelfCal/CommonCfgAndIO/configdata.m`.

In the `SETUP_NAME` section, there a few variables you probably want to examine:

- In particular, set `config.paths.data` to the directory where your calibration data is. This is the output of the `flydra_analysis_generate_recalibration` command. Note: this must end in a slash (`/`).
- `config.cal.GLOBAL_ITER_THR` is the criterion threshold reprojection error that all cameras must meet before terminating the global iteration loop. Something like 0.8 will be decent for an initial calibration (tracking an LED), but tracking tiny *Drosophila* should enable you to go to 0.3 or so (in other words, generating calibration data with the *Running MultiCamSelfCal* method). To estimate the non-linear distortion (often not necessary), set this small enough that `gocal` runs non-linear parameter estimation at least once. This non-linear estimation step fits the radial distortion term.
- `config.cal.USE_NTH_FRAME` if your calibration data set is too massive, reduce it with this variable. Typically, a successful calibration will have about 300-500 points being used in the final calibration. The number of points used will be displayed during the calibration step (For example, "437 points/frames have survived validations so far".)
- `config.files.idxcams` should be set to `[1:X]` where `X` is the number of cameras you are using.
- `config.cal.UNDO_RADIAL` should be set to 1 if you are providing a .rad file with non-linear distortion parameters.

The other files to consider are `MultiCamSelfCal/CommonCfgAndIO/expname.m` and `kookaburra/MultiCamSelfCal/MultiCamSelfCal/BlueCLocal/SETUP_NAME.m`. The first file returns a string that specifies the setup name, and consequently the filename (written above as `SETUP_NAME`) for the second file. This second file contains (approximate) camera positions which are used to determine the rotation, translation, and scale factors for the final camera calibration. The current dynamic models operate in meters, while the flydra code automatically multiplies post-calibration 3D coordinates by 1000 (thus, converting millimeters to meters) unless a file named `calibration_units.txt` specifies the units. Thus, unless you create this file, use millimeters for your calibration units.

Run MATLAB (e.g. `matlab -nodesktop -nojvm`). From the MATLAB prompt:

```
cd kookaburra/MultiCamSelfCal/MultiCamSelfCal/
gocal
```

When the initial mean reprojection errors are displayed, numbers of 10 pixels or less bode pretty well for this calibration to converge. It is rare, to get a good calibration when the first iteration has large reprojection errors. Running on a fast computer (e.g. Core 2 Duo 2 GHz), a calibration shouldn't take more than 5 minutes before looking pretty good if things are going well. Note that, for the first calibration, it may not be particularly important to get a great calibration because it will be redone due to the considerations listed in *Running MultiCamSelfCal*.

3.4 Advanced: automatic homography (alignment) using approximate camera positions

Let's say your calibration had three cameras and you know their approximate positions in world coordinates. You can automatically compute the homography (rotate, scale, and translate) between your original calibration and the new calibration such that the calibrated camera positions will be maximally similar to the given approximate positions.

Create a file called, e.g. `align-cams.txt`. Each line contains the 3D coordinates of each camera. The order of the cameras must be the same as in the calibration. Now, simply run:

```
``flydra_analysis_align_calibration --orig-reconstructor cal20110309a2.xml --align-
↪cams align-cams.txt --output-xml``
```

The aligned calibration will be in `ORIGINAL_RECONSTRUCTOR.aligned.xml`.

3.5 Advanced: using 3D trajectories to re-calibrate using MultiCam-SelfCal

Often, it is possible (and desirable) to make a higher precision trajectory than that possible by waving an LED. For example, flying *Drosophila* are smaller and therefore more precisely localized points than an LED. Also, in setups in which cameras film through movable transparent material, flies fly in the final experimental configuration, which may have slightly different optics that should be part of your final calibration.

By default, you enter previously-tracked trajectory ID numbers and the 2D data that comprised these trajectories are output.

This method also saves a directory with the raw data expected by the Multi Camera Self Calibration Toolbox.

```
# NOTE: if your 2D and 3D data are in one file,
# don't use the "--2d-data" argument.
flydra_analysis_generate_recalibration DATAFILE3D.h5 EFILE \
   --2d-data DATAFILE2D.h5
# This will output a new calibration directory in
# DATAFILE3D.h5.recal
```

The `EFILE` above should have the following format (for example):

```
# These are the obj_ids of traces to use.
long_ids = [655, 646, 530, 714, 619, 288, 576, 645]
# These are the obj_ids of traces not to use (exluded
# from the list in long_ids)
bad=[]
```

Finally, run the Multi Cam Self Calibration procedure on the new calibration directory. Lower your threshold to, e.g., `config.cal.GLOBAL_ITER_THR = .4;`. You might want to adjust `config.cal.USE_NTH_FRAME` again to get the right number of data points. This is a precise calibration, it might take as many as 30 iterations and 15 minutes.

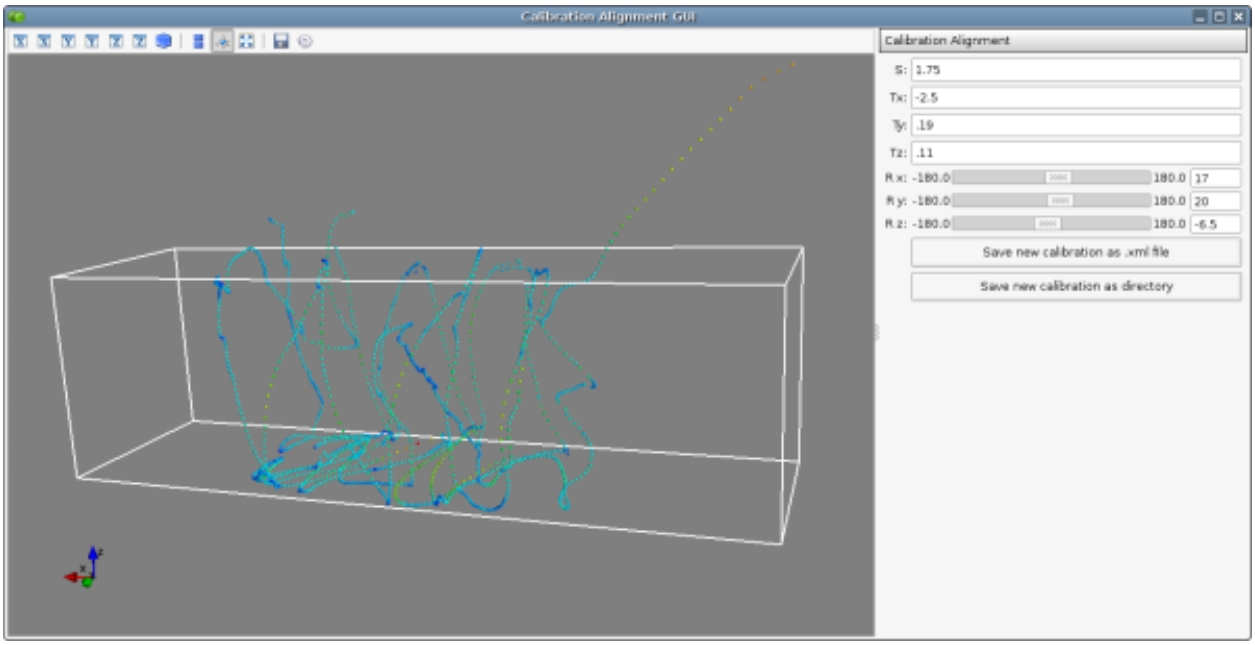
CHAPTER 4

Aligning a calibration

Often, even if a calibration from MultiCamSelfCal creates reprojections with minimal error and the relative camera positions look OK, reconstructed world coordinates do not correspond with desired world coordinates. To align the calibration the **`flydra_analysis_calibration_align_gui`** program may be used:

```
flydra_analysis_calibration_align_gui DATAFILE3D.h5 --stim-xml=STIMULUS.xml
```

This results in a GUI that looks a bit like



Using the controls on the right, align your data such that it corresponds with the 3D model loaded by STIMULUS.xml. When you are satisfied, click either of the save buttons to save your newly-aligned calibration.

4.1 Manually generating 3D points from images to use for alignment

You may want to precisely align some known 3D points. In this case the procedure is:

1. Use **`flydra_analysis_plot_kalman_2d`** to save a *points.h5* file with the 3D positions resulting from the original calibration. In particular, use the hotkeys as defined in `on_key_press()`.
2. Load *points.h5* and a STIMULUS.xml file into **`flydra_analysis_calibration_align_gui`** and adjust the homography parameters until the 3D locations are correct.

CHAPTER 5

Estimating non-linear distortion parameters

The goal of estimating non-linear distortions is to find the image warping such that images of real straight lines are straight in the images. There are two supported ways of estimating non-linear distortion parameters.:

1. Using the pinpoint GUI to manually adjust the warping parameters.
2. Using **`flydra_checkerboard`** to automatically estimate the parameters.

5.1 Use of flydra_checkerboard

`flydra_checkerboard` is a command-line program that generates a .rad file suitable for use by MultiCamSelfCal and the flydra tools (when included in a calibration directory).

The program is run with the name of a config file and possibly some optional command-line arguments.

If everything goes well, it will:

1. Detect the checkerboard corners
2. Cluster these corners into nearly orthogonal multi-segment pieces.
3. Estimate the best non-linear distortion that fits this multi-segments paths as closely as possible to straight lines.

The most important aspect of automatic corner detection is that long, multi-segment paths are detected near the edges of the image.

A minimal, but often sufficient, config file is given here. In this case, this file is named *distorted2.cfg*:

```
fname='distorted2.fmf' # The name of an .fmf movie with frames of a checkerboard
frames= 0,1,2,3 # The frames to extract checkerboard corners from
rad_fname = 'distorted2.rad' # The filename to save the results in.
```

A variety of other options exist:

```
use = 'raw' # The image pre-processing algorithm to use before
            # extracting checkerboard corners. In order of preference, the options␣
↪are:
```

```
            # 'raw'       - the raw image, exactly as-is
            # 'rawbinary' - a thresholded image
            # 'binary'    - a background-subtracted and thresholded image
            # 'no_bg'     - a background-subtracted image
angle_precision_degrees=10.0 # Threshold angular difference between adjacent edges.
aspect_ratio = 1.0           # Aspect ratio of pixel spacing (1.0 is normal,
                                                                0.5 is vertically␣
↪downsampled)

show_lines = False
return_early = False
debug_line_finding = False
epsfcn = 1e09
print_debug_info = False
save_debug_images = False

ftol=0.001
xtol=0
do_plot = False

K13 = 320 # center guess X
K23 = 240 # center guess Y

kc1 = 0.0 # initial guess of radial distortion
kc2 = 0.0 # initial guess of radial distortion
```

After adjusting these parameters, call **flydra_checkerboard**.

Critical to **flydra_checkerboard** is the ability to extract numerous checkerboard corners with few false positives. To ensure that this is happens, here are a few command line options that help debug the process:

```
flydra_checkerboard distorted2.cfg --show-chessboard-finder-preview
```

The first image is a screenshot of the *–show-chessboard-finder-preview* output when using the 'raw' image. The detection of corners is good throughout most of the image, but lacking particularly in the lower left corner. The second image used the 'rawbinary' preprocessing mode. It appears to have detected more points, which is good.

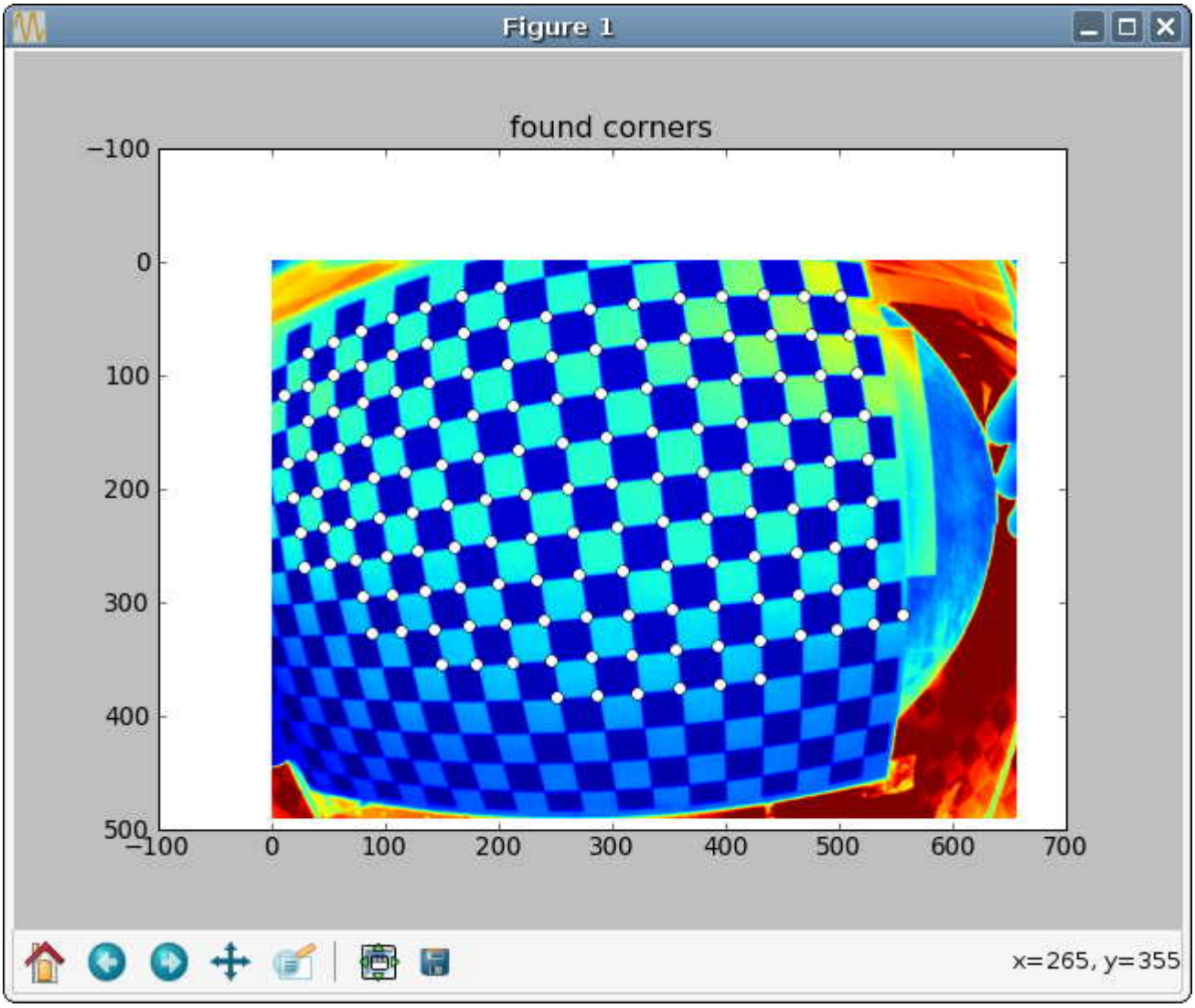
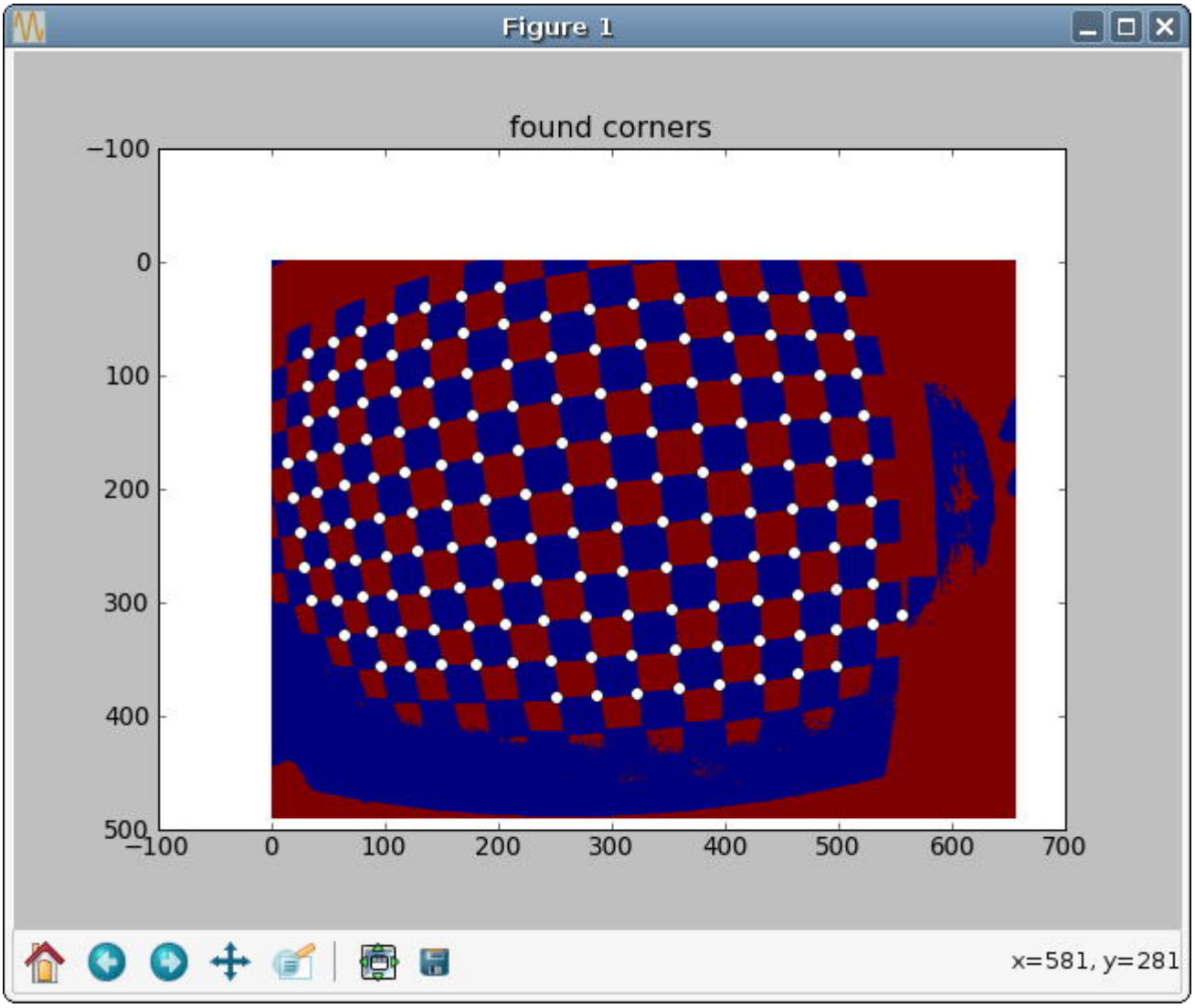


Figure 1
found corners
−100
0
100
200
300
400
500
−100
0
100
200
300
400
500
600
700
x=265, y=355



Finding the horizontal and vertical edges

The next step is for **flydra_checkerboard** to find the grid of the chessboard. Run the folllowing command to see how well it does:

```
flydra_checkerboard distorted2.cfg --find-and-show1
```

Here are two sample images this was performed on. In the first image, we can see that the grid detection was very good, with no obvious mistakes. In the second example, the grid detection had a couple mistakes – one in the lower right corner and one in the upper right corner.

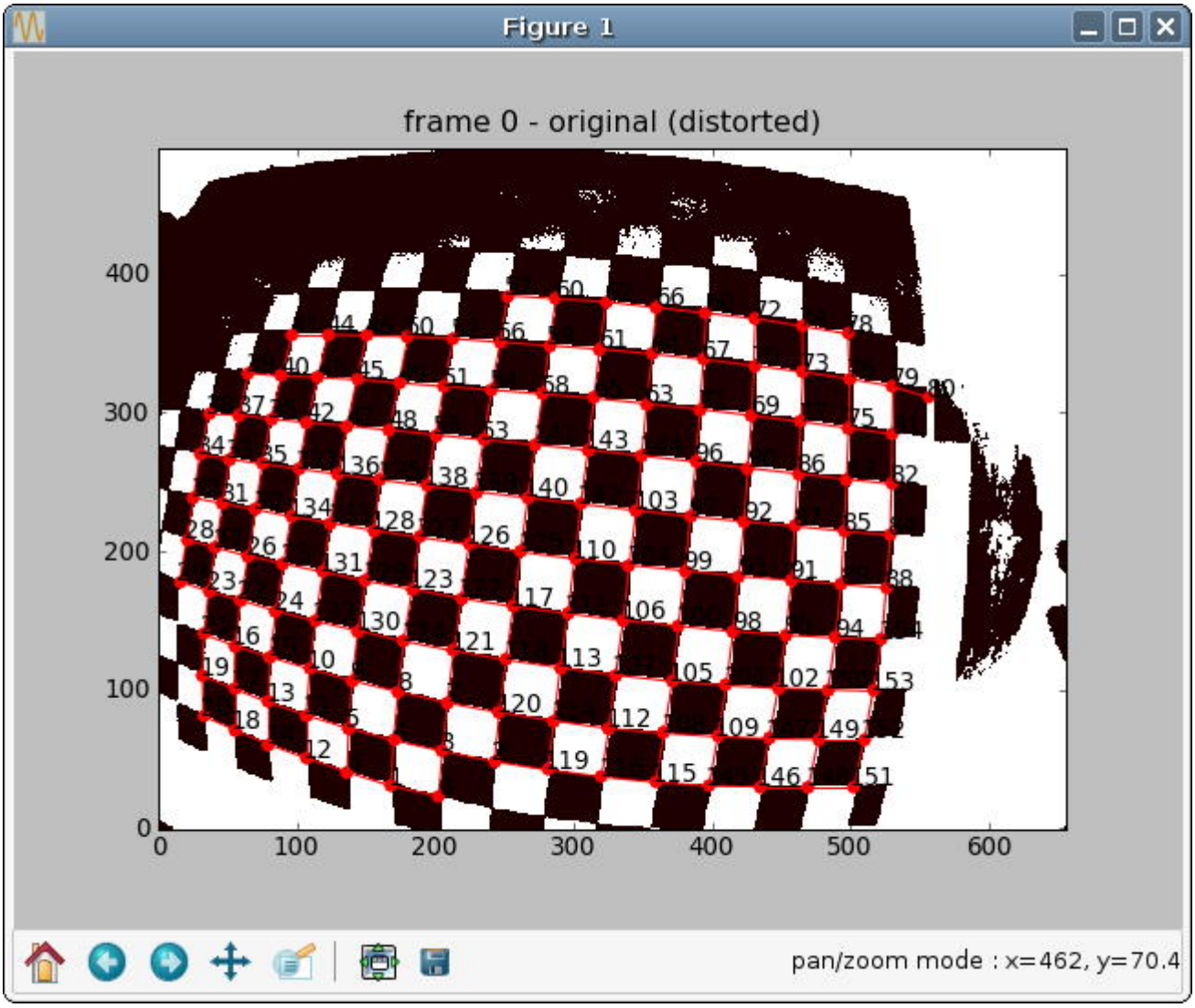
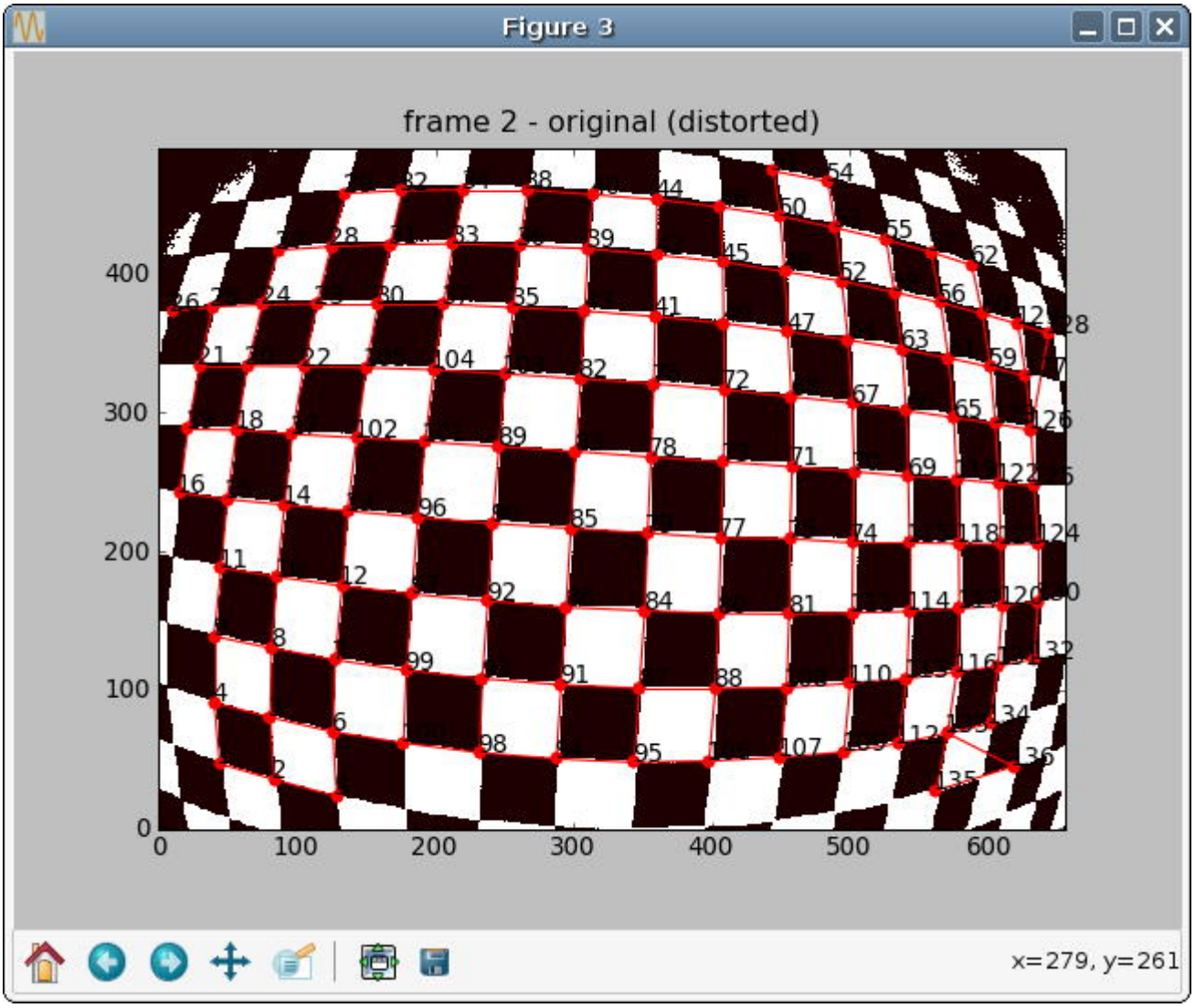


Figure 1
frame 0 - original (distorted)
0
100
200
300
400
500
600
pan/zoom mode : x=462, y=70.4



If everything looks good to this point, you may be interested in a final check with the *–find-and-show2*, which identifies individual paths. It is these paths that will be attempted to straighted in the optimization procedure to follow.

To actually estimate the radial distortion, call the command with no options:

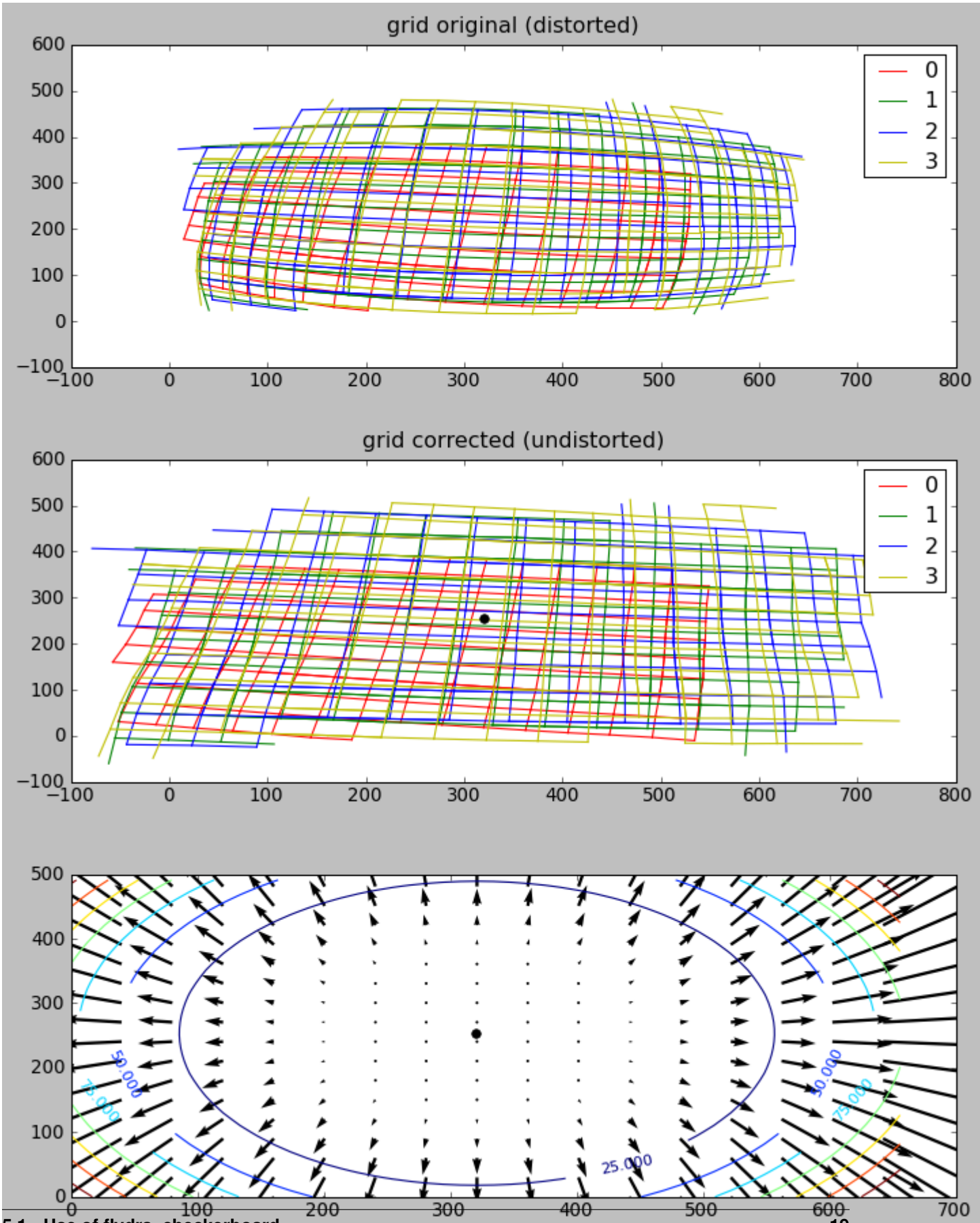
```
flydra_checkerboard distorted2.cfg
```

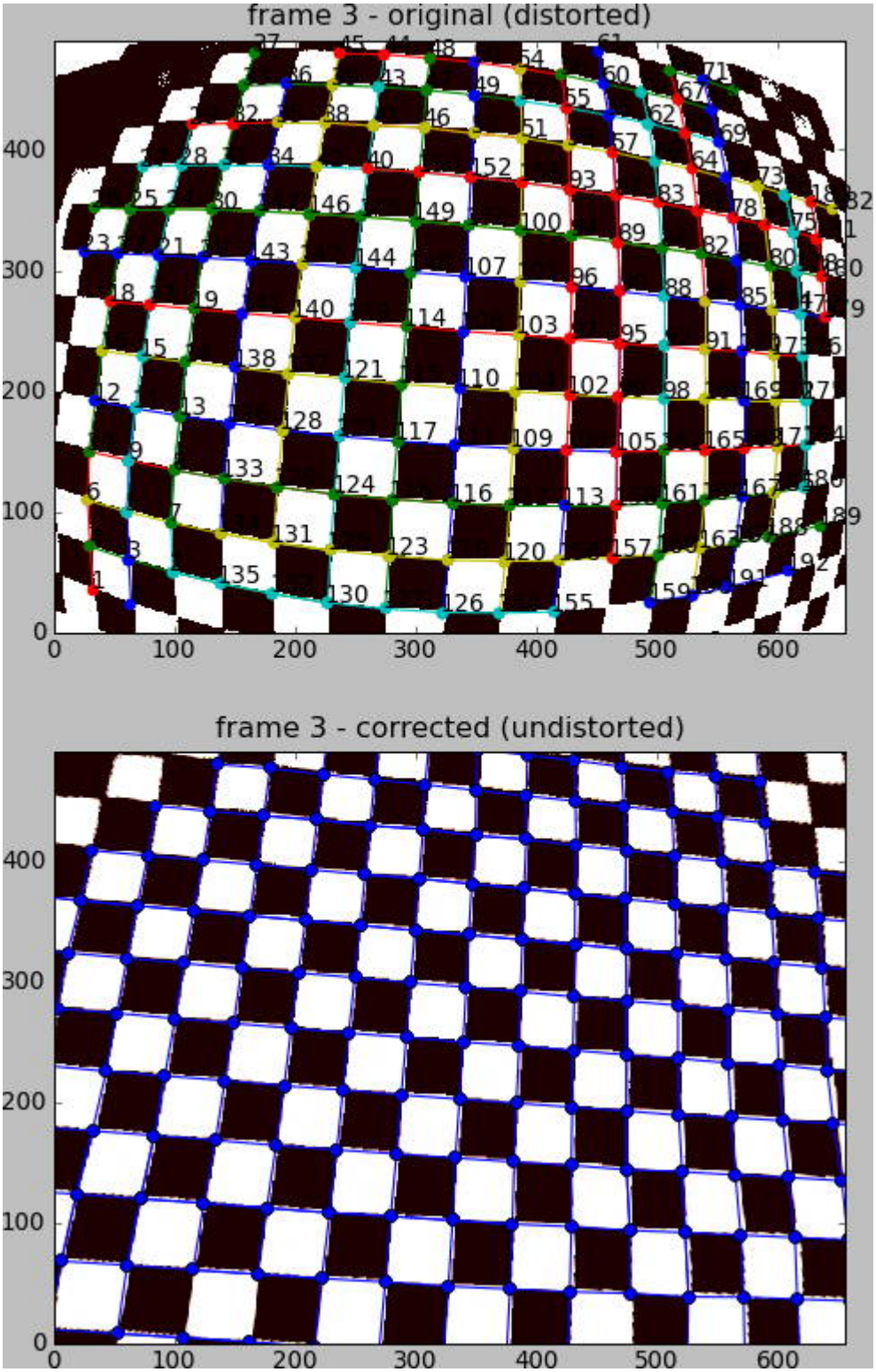
This will begin a gradient descent type optimization and will hopefully return a set of good values. Note that you can seed the starting values for the parameters with the K13, K23, kc1, and kc2 parameters described above. Over time you should see the error decreasing, rapidly at first, and then more slowly.

Once the optimization is done, you may visualize the results. This command reads the non-linear distortion terms from the .rad files:

```
flydra_checkerboard distorted2.cfg  --view-results
```

This command reads all the files, re-finds the corners, and plots several summary plots. **The most importart thing is that straight lines look straight.** In the example images below, the distortion estimation appears to have done a reasonably good job – the undistorted image has rather straight lines, and the "grid corrected" panel appears to show mostly straight (although not perfect) checkerboards.


grid original (distorted)
0
1
2
3
grid corrected (undistorted)
0
1
2
3
50.000
75.000
25.000
50.000
75.000


frame 3 - original (distorted)
0
100
200
300
400
500
600
frame 3 - corrected (undistorted)
0
100
200
300
400
500
600

CHAPTER 6

Realtime data

Flydra is capable of outputing a low latency stream of state estimate from the Kalman filter.

The data are originally encoded into a "data packet" by `flydra_core.kalman.flydra_tracker.Tracker.encode_data_packet()`.

Then, they are transferred to a separate thread in the `flydra_core.MainBrain.CoordinateSender` class. There, multiple packets are combined into a "super packet" with `flydra_core.kalman.data_packets.encode_super_packet()`. By combining into a single buffer to send over the network, fewer system calls are made, resulting in better performance and reducing overall latency.

CHAPTER 7

Data analysis

7.1 Types of data files

7.1.1 Calibration files and directories - overview

(See *Calibration* for an overview on calibration.)

Calibrations may be saved as:

- A calibration directory (see below)
- An .xml file with just the calibration.

Additionally, calibrations may be saved in:

- .h5 files
- .xml files that also include stimulus or trajectory information.

All calibration sources can save the camera matrix for the linear pinhole camera model for each camera, the scale factor and units of the overall calibration, and the non-linear distortion parameters for each camera.

Calibration directories

To provide compatibility with the Multi Camera Self Calibration Toolbox by Svoboda, et al, the calibration directory includes the following files:

- calibration_units.txt - a string describing the units of the calibration
- camera_order.txt - a list of the flydra cam_ids
- IdMat.dat - booleans indicating the valid data
- original_cam_centers.dat - TODO
- points.dat - the 2D image coordinates of the calibration object
- Res.dat - the resolution of the cameras, in pixels

- camN.rad - the non-linear distortion terms for camera N

The Multi Camera Self Calibration Toolbox (written in MATLAB) adds several more files. These files contain the extrinsic and intrinsic parameters of a pinhole model as well as non-linear terms for a radial distortion. flydra_mainbrain loads these files and sends some of this information to the camera nodes. Specifically the files are:

- TODO - (Need to write about the files that the MultiCamSelfCal toolbox adds here.)

7.1.2 Image files

- .fmf (Fly Movie Format) files contain raw, uncompressed image data and timestamps. Additionally, floating point formats may be stored to facilitate saving ongoing pixel-by-pixel mean and variance calculations.
- .ufmf (micro Fly Movie Format) files contain small regions of the entire image. When most of the image is unchanging, this format allows reconstruction of near-lossless movies at a fraction of the disk and CPU usage of other formats.

7.1.3 Tracking data files

Tracking data (position over time) is stored using the HDF5 file format using the pytables library. At a high level, there are two types of such files:

- raw 2D data files. These contain `/data2d_distorted` and `/cam_info` tables. Throughout the documentation, such files will be represented as `DATAFILE2D.h5`.
- "Kalmanized" 3D data files. These contain `/kalman_estimates` and `/ML_estimates` tables in addition to a `/calibration` group. Throughout the documentation, such files will be represented as `DATAFILE3D.h5`.

Note that a single .h5 file may have both sets of features, and thus may be both a raw 2D data file in addition to a kalmanized 3D data file. For any data processing step, it is usually only one or the other aspect of this file that is important, and thus the roles above could be played by a single file.

7.1.4 Stimulus, arena, and compound trajectory descriptions

XML files may be used to specify many aspects of an experiment and analysis through an extensible format. Like *Tracking data files*, these XML files may have multiple roles within a single file type. The roles include

- Arena description. The tags `cubic_arena` and `cylindrical_arena`, for example, are used to define a rectangular and cylindrical arena, respectively.
- Stimulus description. The tag `cylindrical_post`, for example, is used to define a stationary cylinder.

Because the format is extensible, adding further support can be done in a backwards-compatible way. These XML files are handled primarily through *`flydra_analysis.a2.xml_stimulus`*.

7.2 Predefined analysis programs

(TODO: port the list of programs from the webpage.)

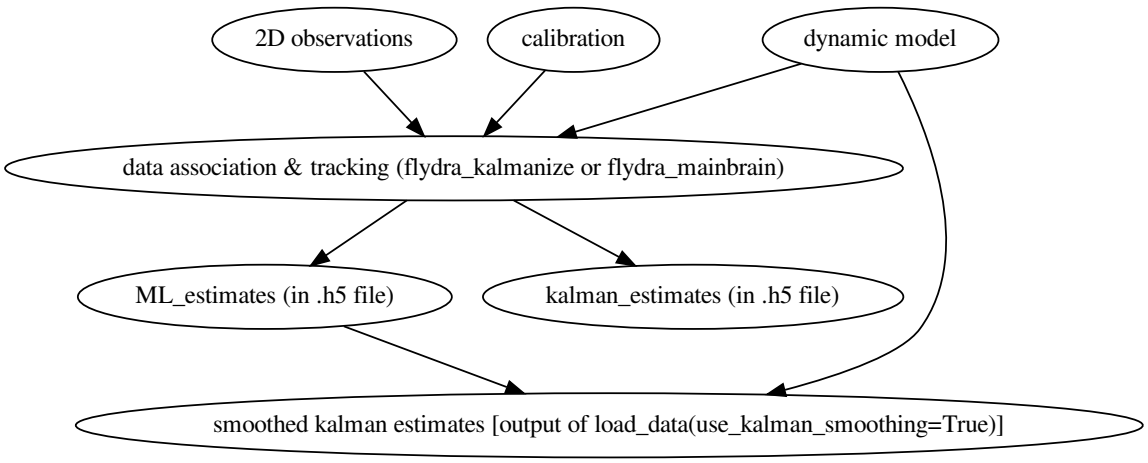
7.3 Automating data analysis

The module `flydra_analysis.a2.flydra_scons` provides definitions that may be useful in building SConstruct files for scons. Using scons allows relatively simple batch processing to be specified, including the ability to concurrently execute several jobs at once.

7.4 Source code for your own data analysis

The module *`flydra_analysis.a2.core_analysis`* has fast, optimized trajectory opening routines.

7.5 Data flow



7.6 Extracting longitudinal body orientation

See also *Estimating orientations with flydra*.

7.6.1 Theoretical overview

Our high-throughput automated pitch angle estimation algorithm consists of two main steps: first, the body angle is estimated in (2D) image coordinates for each camera view, and second, the data from multiple cameras are fused to establish a 3D estimate of longitudinal body orientation. We take as input the body position, the raw camera images, and an estimate of background appearance (without the fly). These are calculated in a previous step according to the EKF based algorithm described in the flydra manuscript.

For the first step (2D body angle estimation), we do a background subtraction and thresholding operation to extract a binary image containing the silhouette of the fly. A potential difficulty is distinguishing the portion of the silhouette

caused by the wings from the portion caused by the head, thorax, and abdomen. We found empirically that performing a connected components analysis on the binary image thresholded using an appropriately chosen threshold value discriminates the wings from the body with high success. Once the body pixels are estimated in this way, a covariance matrix of these pixels is formed and its eigenvalues and eigenvectors are used to determine the 2D orientation of luminance within this binary image of the fly body. **To add:** a description of the image blending technique used with high-framerate images for ignoring flapping wings.

From the N estimates of body angle from N camera views, an estimate of the 3D body axis direction is made. See *Estimating orientations with flydra* for a description of this step.

7.6.2 Practical steps

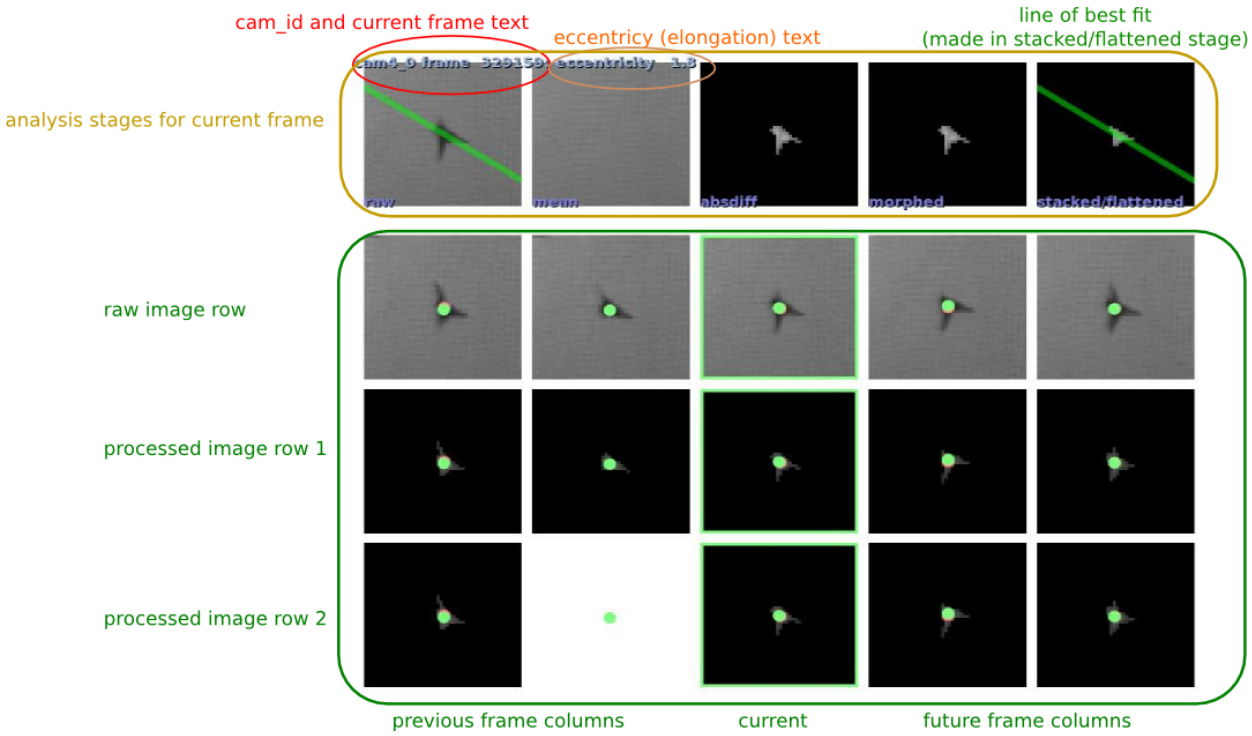
Estimating longitudinal body orientation happens in several steps:

- Acquire data with good 2D tracking, a good calibration, and .ufmf movies in good lighting.
- Perform tracking and data assocation on the 2D data to get 3D data using **`flydra_kalmanize`**.
- Run **`flydra_analysis_image_based_orientation`** to estimate 2D longitudinal body axis.
- Check the 2D body axis estimates using **`flydra_analysis_montage_ufmfs`** to generate images or movies of the tracking.
- Finally, take the 2D orientation data and make 3D estimates. Nowdays the best way to do this is with **`flydra_analysis_orientation_ekf_fitter`**, as described *here*. (The old way was another run through the tracker and data association using the **`flydra_kalmanize`** program again.) This 2D to 3D stage is covered in the *estimating orientations with flydra* section.

An example of a call to **`flydra_analysis_image_based_orientation`** is: (This was automatically called via an SConstruct script using `flydra_analysis.a2.flydra_scons`.)

```
flydra_analysis_image_based_orientation --h5=DATA20080915_164551.h5 --
↪kalman=DATA20080915_164551.kalmanized.h5 \
  --ufmfs=small_20080915_164551_cam1_0.ufmf:small_20080915_164551_cam2_0.ufmf:small_
↪20080915_164551_cam3_0.ufmf:small_20080915_164551_cam4_0.ufmf \
  --output-h5=DATA20080915_164551.image-based-re2d.h5
```

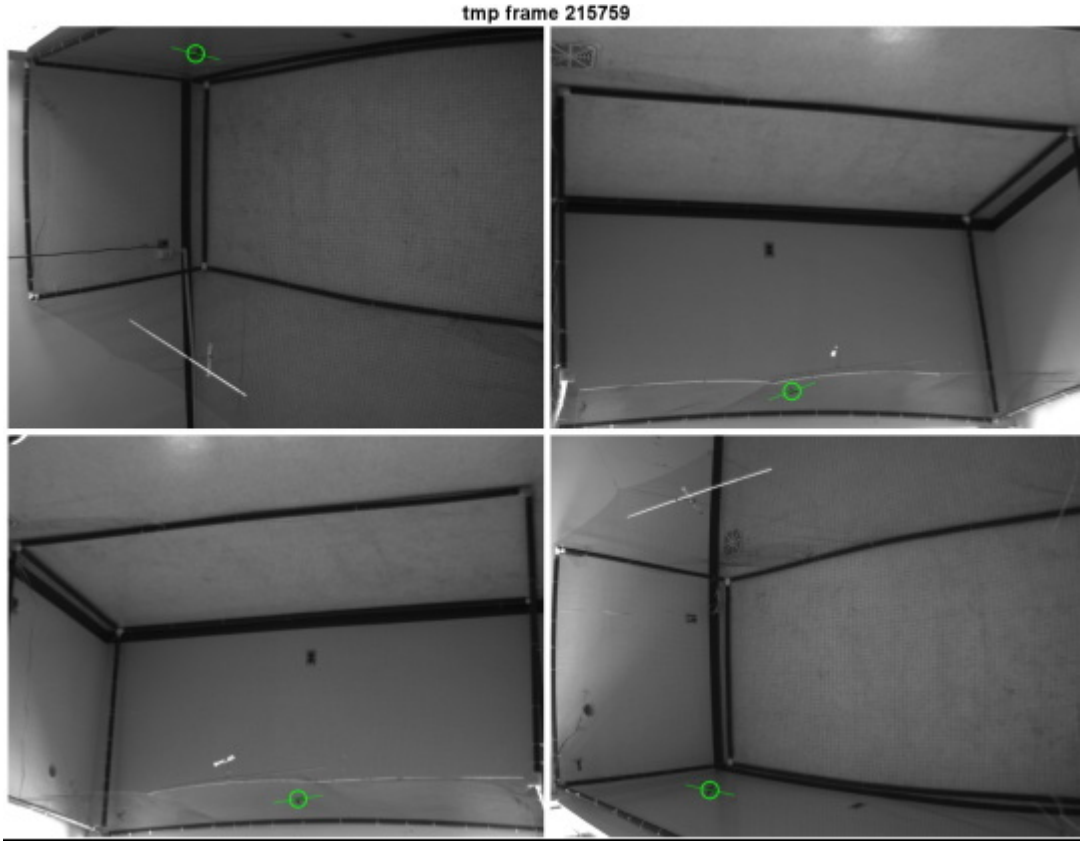
You can use the `--save-images` option to the **`flydra_analysis_image_based_orientation`** command. You will then generate a series of images that look like this:



When calling **`flydra_analysis_montage_ufmfs`**, you'll need to use at least the following elements in a configuration file:

```
[what to show]
show_2d_orientation = True
```

An example output from from doing something like this is shown here:



The **critical issue** is that the body orientations are well tracked in 2D. There's nothing that can be done in later processing stages if the 2D body angle extraction is not good.

CHAPTER 8

Gallery

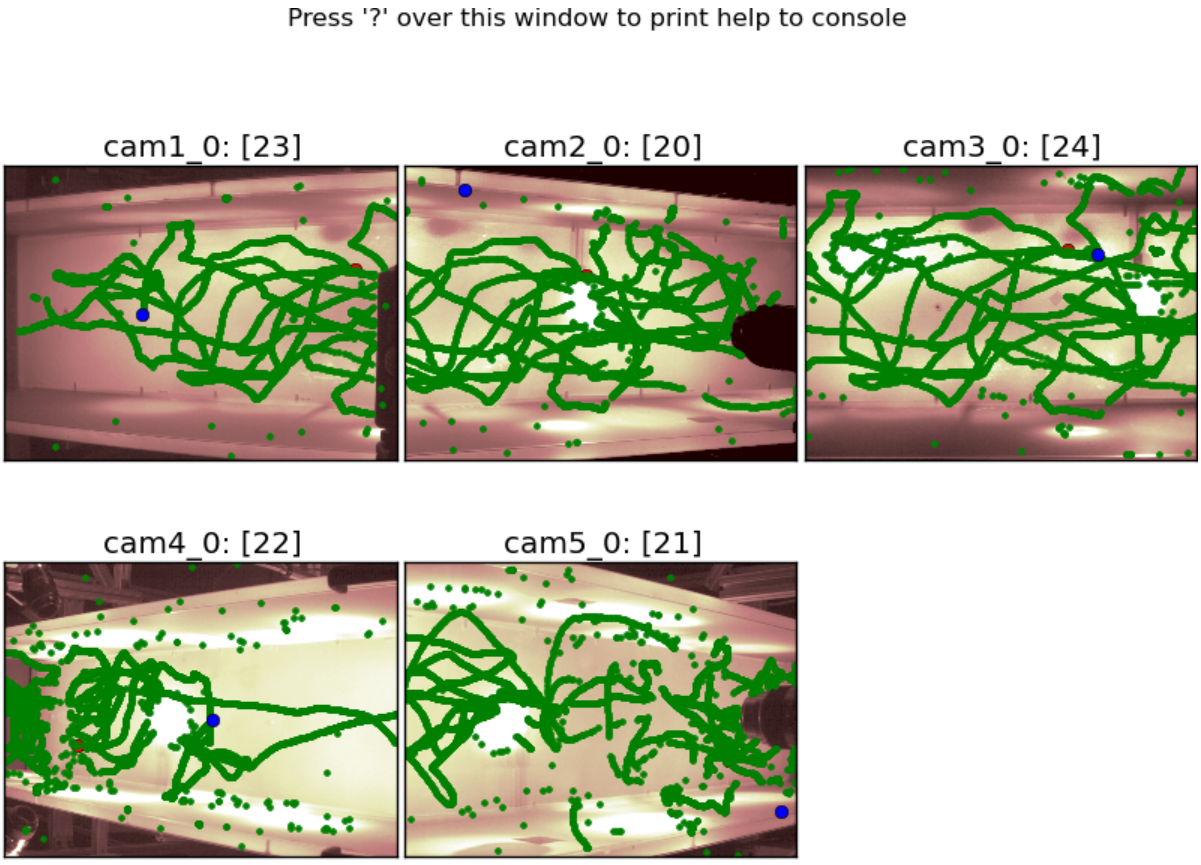
This page shows images that were automatically generated by the command line tools installed with flydra. The command line used to generate each figure is shown. These figures also serve as unit tests for flydra – the stored versions are compared with newly generated versions whenever nosetests is run.

8.1 Image gallery

8.1.1 Camera view of 2D data

The following command generated this image:

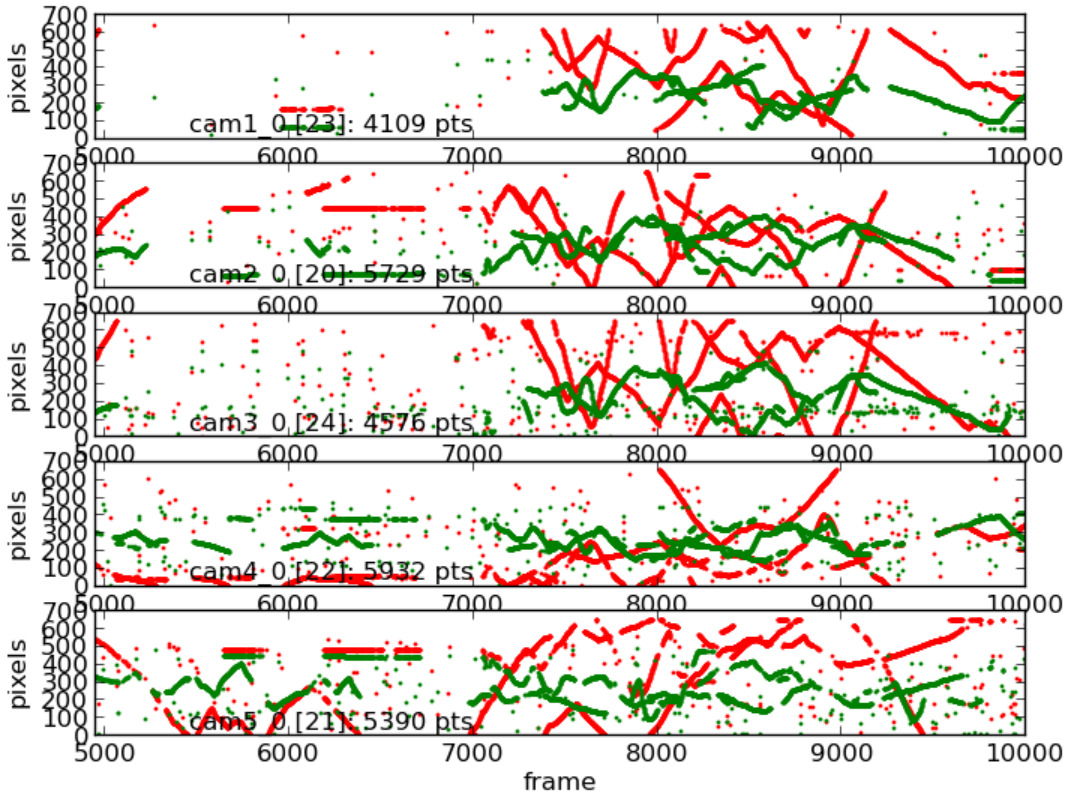
```
flydra_analysis_plot_kalman_2d DATAFILE2D.h5 --save-fig=image.png
```



8.1.2 Timeseries of 2D data

The following command generated this image:

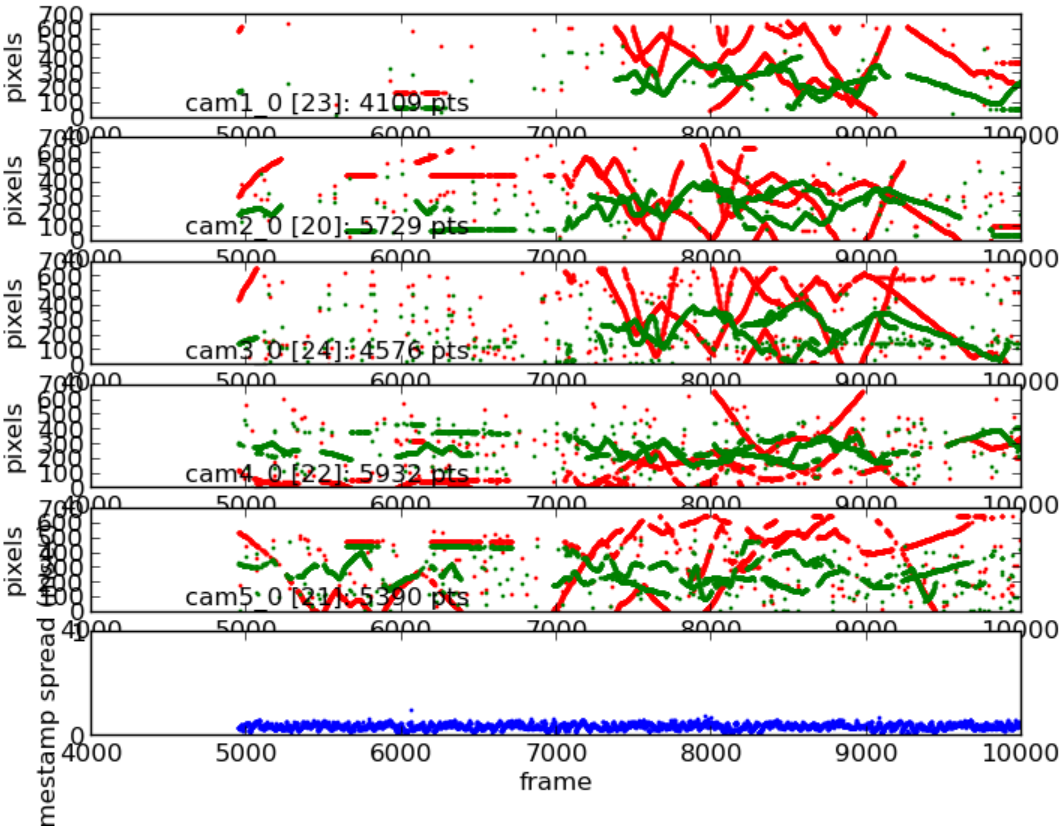
```
flydra_analysis_plot_timeseries_2d_3d DATAFILE2D.h5 --save-fig=image.png \
     --hide-source-name
```



8.1.3 Timeseries of 2D data with frame synchronization data

The following command generated this image:

```
flydra_analysis_plot_timeseries_2d_3d DATAFILE2D.h5 \
     --spreadh5=DATAFILE2D.h5.spreadh5 --save-fig=image.png \
     --hide-source-name
```

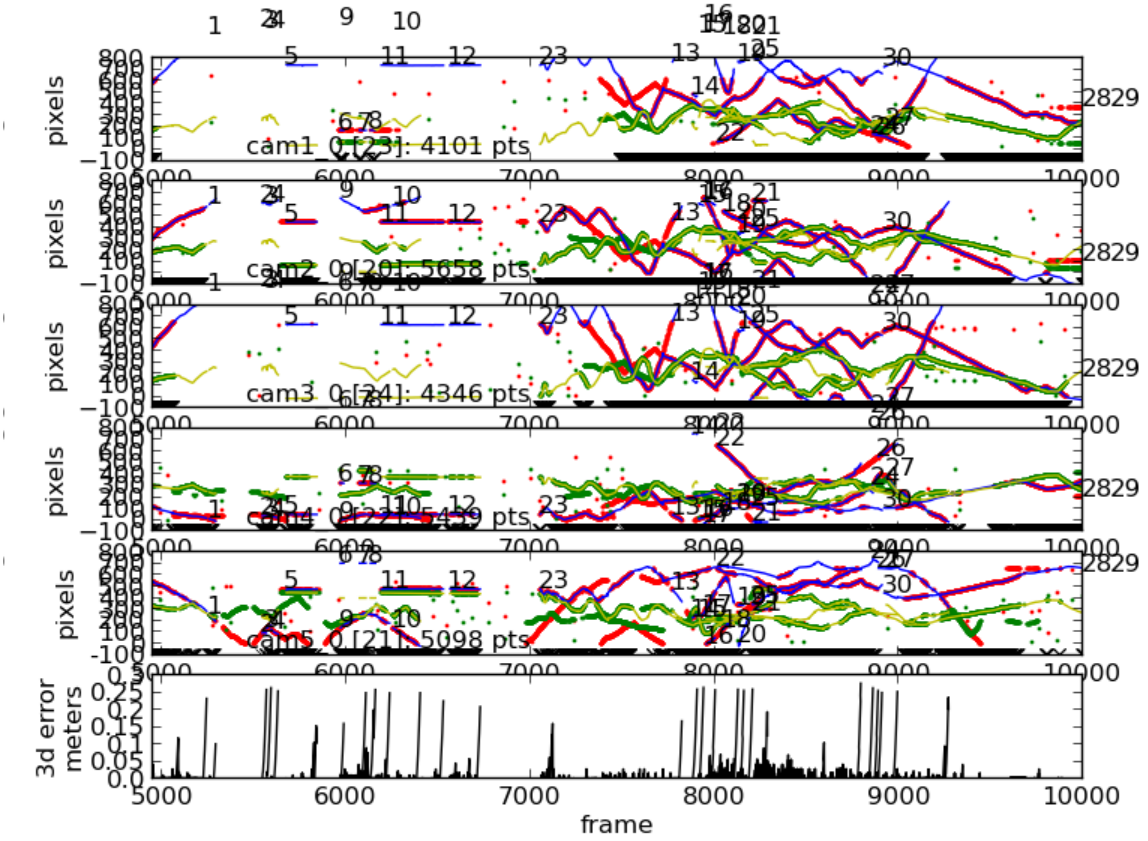


8.1.4 Timeseries of 2D and 3D data

The following command generated this image:

```
flydra_analysis_plot_timeseries_2d_3d DATAFILE2D.h5 \
     --kalman-file=DATAFILE3D.h5 --disable-kalman-smoothing \
     --save-fig=image.png --likely-only --hide-source-name
```

The `--likely-only` argument limits the 2D data plotted.



8.2 Command gallery

8.2.1 Re-run the data association algorithm

```
flydra_kalmanize DATAFILE2D.h5 --reconstructor=CALIBRATION.xml --max-err=10.0 \
     --min-observations-to-save=10 --dest-file=DATAFILE2D.kalmanized.h5
```

This re-runs the data association algorithm. It is useful to do this because the original realtime run may have skipped some processing to meet realtime constraints or because a better calibration is known. The new data are saved to an .h5 file named `DATAFILE2D.kalmanized.h5`.

8.2.2 Export data to MATLAB .mat file

```
flydra_analysis_data2smoothed DATAFILE3D.h5 --time-data=DATAFILE2D.h5 \
     --dest-file=DATAFILE3D_smoothed.mat
```

This produces a .mat file named `DATAFILE3D_smoothed.mat`. This file contains smoothed tracking data in addition to (unsmoothed) maximum likelihood position estimates.

8.2.3 Extract frame synchronization data

```
flydra_analysis_check_sync DATAFILE2D.h5 --dest-file=frame_sync_info.spreadh5
```

This produces a file named `frame_sync_info.spreadh5` containing the spread of the timestamps in DATAFILE2D which may be plotted with flydra_analysis_plot_timeseries_2d_3d. Additionally, this command exits with a non-zero exit code if there are synchronization errors.

CHAPTER 9

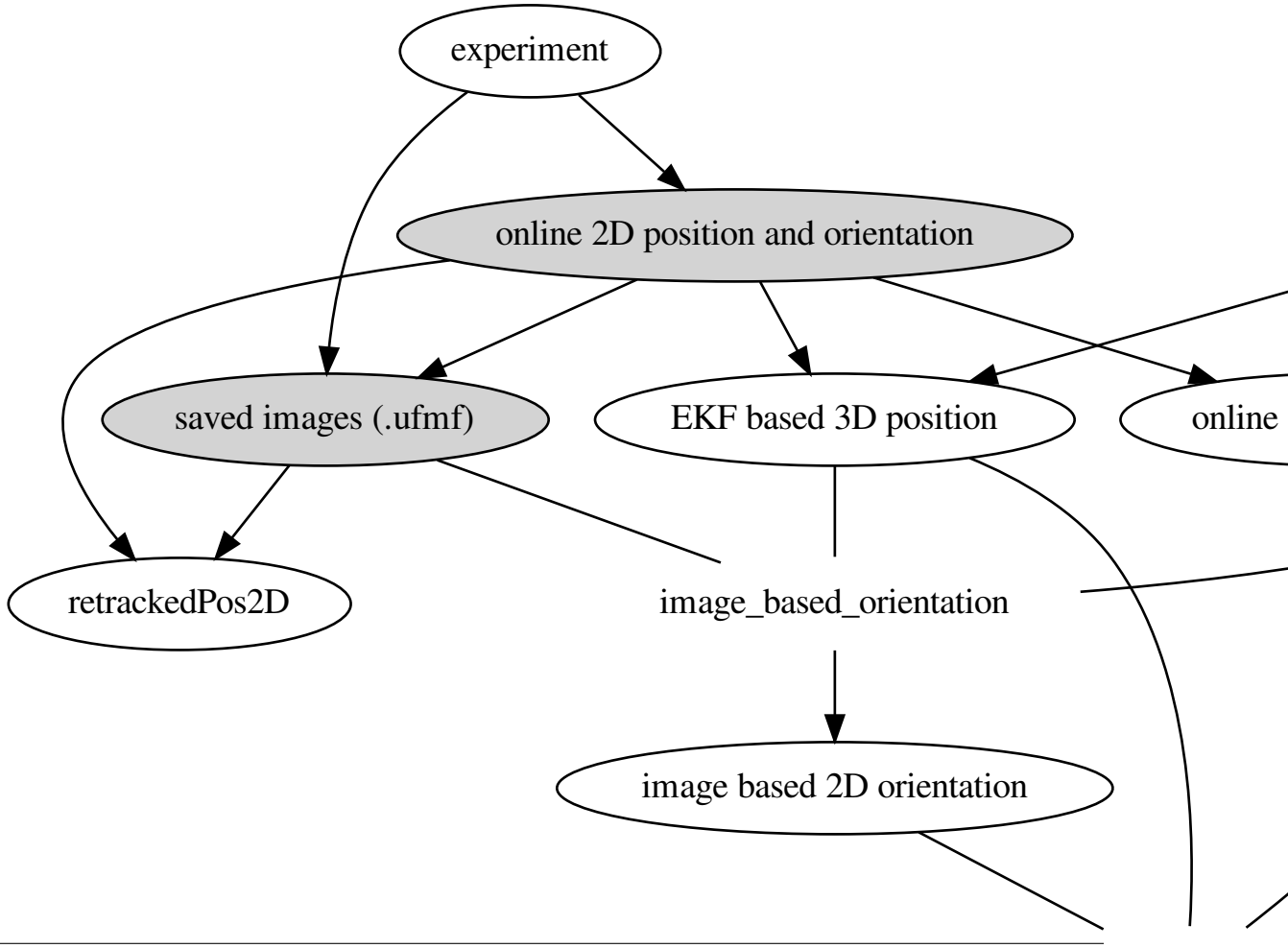
The Flydra Trigger Device

The Flydra Trigger Device serves to provide a synchronized trigger signal to all cameras providing input to the Main Brain. By use of a custom trigger device, precise timing of frame-by-frame inputs can be verified.

All aspects of the trigger device are now hosted here: https://github.com/strawlab/triggerbox

CHAPTER 10

Estimating orientations with flydra



Contents:

10.1 Fusing 2D orientations to 3D

Flydra uses an extended Kalman filter (EKF) and a simple data association algorithm to fuse 2D orientation data into an a 3D orientation estimate. The program **flydra_analysis_orientation_ekf_fitter** is used to perform this step, and takes, amongst other data, the 2D orientation data stored in the `slope` column of the `data2d_distorted` table and converts it into the `hz_line*` columns of the `ML_estimates` table. (The directional component of these Pluecker coordinates should be ignored, as it is meaningless.)

See *smoothing orientations* for a description of the step that chooses orientations (and thus removes the 180 degree ambiguity in the body orientation estimates).

The following sections detail the algorithm used for the finding of the hz_line data.

10.1.1 Process model

We are using a quaternion-based Extended Kalman Filter to track body orientation and angular rate in 3D.

From (Marins, Yun, Bachmann, McGhee, and Zyda, 2001) we have state $\mathbf{x} = x_1, x_2, x_3, x_4, x_5, x_6, x_7$ where x_1, x_2, x_3 are angular rates p, q, r and x_4, x_5, x_6, x_y are quaternion components a, b, c, d (with the scalar component being d).

The temporal derivative of $\mathbf{x}$ is $\dot{\mathbf{x}} = f(\mathbf{x})$ and is defined as:

$$\left(-\frac{x_1}{\tau_{rx}} \; -\frac{x_2}{\tau_{ry}} \; -\frac{x_3}{\tau_{rz}} \; \frac{x_1x_7+x_3x_5-x_2x_6}{2\sqrt{{x_4}^2+{x_5}^2+{x_6}^2+{x_7}^2}} \; \frac{x_1x_6+x_2x_7-x_3x_4}{2\sqrt{{x_4}^2+{x_5}^2+{x_6}^2+{x_7}^2}} \; \frac{x_2x_4+x_3x_7-x_1x_5}{2\sqrt{{x_4}^2+{x_5}^2+{x_6}^2+{x_7}^2}} \; \frac{x_3x_6-x_1x_4-x_2x_5}{2\sqrt{{x_4}^2+{x_5}^2+{x_6}^2+{x_7}^2}} \right)$$

The process update equation (for $\mathbf{x}_t|\mathbf{x}_{t-1}$) is:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + f(\mathbf{x}_t)dt + \mathbf{w}_t$$

Where $\mathbf{w}_t$ is the noise term with covariance Q and dt is the time step.

10.1.2 Observation model

The goal is to model how the target orientation given by quaternion $q = ai + bj + ck + d$ results in a line on the image, and finally, the angle of that line on the image. We also need to know the target 3D location, the vector A, and the camera matrix P. Thus, the goal is to define the function $G(q, A, P) = \theta$.

Quaternion q may be used to rotate the vector u using the matrix R:

$$R = \begin{pmatrix} a^2+d^2-b^2-c^2 & -2cd+2ab & 2ac+2bd \\ 2ab+2cd & b^2+d^2-a^2-c^2 & -2ad+2bc \\ -2bd+2ac & 2ad+2bc & c^2+d^2-a^2-b^2 \end{pmatrix}$$

Thus, for $u = (1, 0, 0)$ the default, non-rotated orientation, we find $U = Ru$, the orientation estimate.

$$U = Ru = \begin{pmatrix} a^2+d^2-b^2-c^2 \\ 2ab+2cd \\ -2bd+2ac \end{pmatrix}$$

Now, considering a point passing through A with orientation given by U, we define a second point $B = A + U$.

Given the camera matrix P:

$$P = \begin{pmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \end{pmatrix}$$

The image of point A is PA. Thus the vec on the image is $PB - PA$.

$$G(q, A, P) = \theta = \operatorname{atan}\left(\frac{P_{13}+AxP_{10}+AyP_{11}+AzP_{12}-2P_{12}bd+2P_{11}ab+2P_{11}cd+2P_{12}ac+P_{10}a^2+P_{10}d^2-P_{10}b^2-P_{10}c^2}{P_{23}+AxP_{20}+AyP_{21}+AzP_{22}-2P_{22}bd+2P_{21}ab+2P_{21}cd+2P_{22}ac+P_{20}a^2+P_{20}d^2-P_{20}b^2-P_{20}c^2} - \frac{P_{13}+AxP_{10}+AyP_{11}+AzP}{P_{23}+AxP_{20}+AyP_{21}+AzP} \right.$$
$$\left. \frac{P_{03}+AxP_{00}+AyP_{01}+AzP_{02}-2P_{02}bd+2P_{01}ab+2P_{01}cd+2P_{02}ac+P_{00}a^2+P_{00}d^2-P_{00}b^2-P_{00}c^2}{P_{23}+AxP_{20}+AyP_{21}+AzP_{22}-2P_{22}bd+2P_{21}ab+2P_{21}cd+2P_{22}ac+P_{20}a^2+P_{20}d^2-P_{20}b^2-P_{20}c^2} - \frac{P_{03}+AxP_{00}+AyP_{01}+AzP}{P_{23}+AxP_{20}+AyP_{21}+AzP} \right)$$

Now, we need to shift coordinate system such that angles will be small and thus reasonably approximated by normal distributions. We thus take an expected orientation quaternion q^* and find the expected image angle for that θ^*:

$$\Phi(q^*, A, P) = \theta^*$$

We define our new observation model in this coordinate system:

$$H(q, q^*, A, P) = G(q, A, P) - \Phi(q^*, A, P) = \theta - \theta^*$$

Of course, if the original observation was y, the new observation z must also be placed in this coordinate system.

$$z_t = y_t - \theta^*$$

The EKF prior estimate of orientation is used as the expected orientation q^*, although is possible to use other values for expected orientation.

10.1.3 Data association

The data association follows a very simple rule. An observation z_t is used if and only if this value is close to the expected value. Due to the definition of z_t above, this is equivalent to saying only small absolute values of z_t are associated with the target. This gating is established by the `--gate-angle-threshold-degrees` parameter to **`flydra_analysis_orientation_ekf_fitter`**. `--gate-angle-threshold-degrees` is defined between 0 and 180. The basic idea is that the program has a prior estimate of orientation and angular velocity from past frames, and any new 2D orientation is accepted or not (gated) based on whether the acceptance makes sense – whether it's close to the predicted value. So a value of zero means reject everything and 180 means accept everything. 10 means that you believe your prior estimates and only accept close observations, where as 170 means you think the prior is less reliable than the observation. (IIRC, the presence or absence of the green line in the videos indicates whether the 2D orientation was gated in or out, respectively.)

`--area-threshold-for-orientation` lets you discard a point if the area of the 2D detection is too low. Many spurious detections often have really low area, so this is a good way to get rid of them. However, the default of this value is zero, so I think when I wrote the program I found it to be unnecessary.

10.2 Smoothing 3D orientations

Given the raw Pluecker coordinates estimating body direction found with **`flydra_analysis_orientation_ekf_fitter`** (and described *here*), we remove the 180 degree ambiguity ("choose orientations") and perform final smoothing in the code in the module *`flydra_analysis.a2.core_analysis`*. Command-line programs that export this data include **`flydra_analysis_data2smoothed`** and **`flydra_analysis_montage_ufmfs`**. The command-line arguments to these programs support changing all the various parameters in this smoothing and orientation selection. Specifically, these parameters are:

```
--min-ori-quality-required=MIN_ORI_QUALITY_REQUIRED
                      minimum orientation quality required to emit 3D
                      orientation info
--ori-quality-smooth-len=ORI_QUALITY_SMOOTH_LEN
                      smoothing length of trajectory
```

See also *Data analysis* (specifically the "Extracting longitudinal body orientation" section).

CHAPTER 11

Modules reference

11.1 flydra_analysis.a2 - analysis (second generation)

11.1.1 `flydra_analysis.a2.benu`

11.1.2 flydra_analysis.a2.core_analysis

get_global_CachingAnalyzer

CachingAnalyzer

choose_orientations

11.1.3 flydra_analysis.a2.plot_kalman_2d

11.1.4 flydra_analysis.a2.utils

11.1.5 flydra_analysis.a2.xml_stimulus

11.2 flydra_analysis.analysis - data analysis modules

11.2.1 flydra_analysis.analysis.PQmath

11.3 Miscellaneous flydra_core modules

11.3.1 flydra_core.geom

ThreeTuple

PlueckerLine

11.3.2 flydra_core.kalman.dynamic_models

get_model_names

get_kalman_model

create_dynamic_model_dict

MamaramaMMEKFAllParams

11.3.3 flydra_core.kalman.flydra_tracker

Tracker

11.3.4 flydra_core.MainBrain

11.3.5 flydra_core.reconstruct

CHAPTER 12

old Trac wiki page about Flydra

This page was copied from the old Trac wiki page about Flydra and may have suffered some conversion issues.

Flydra is heavily based on the motmot packages.

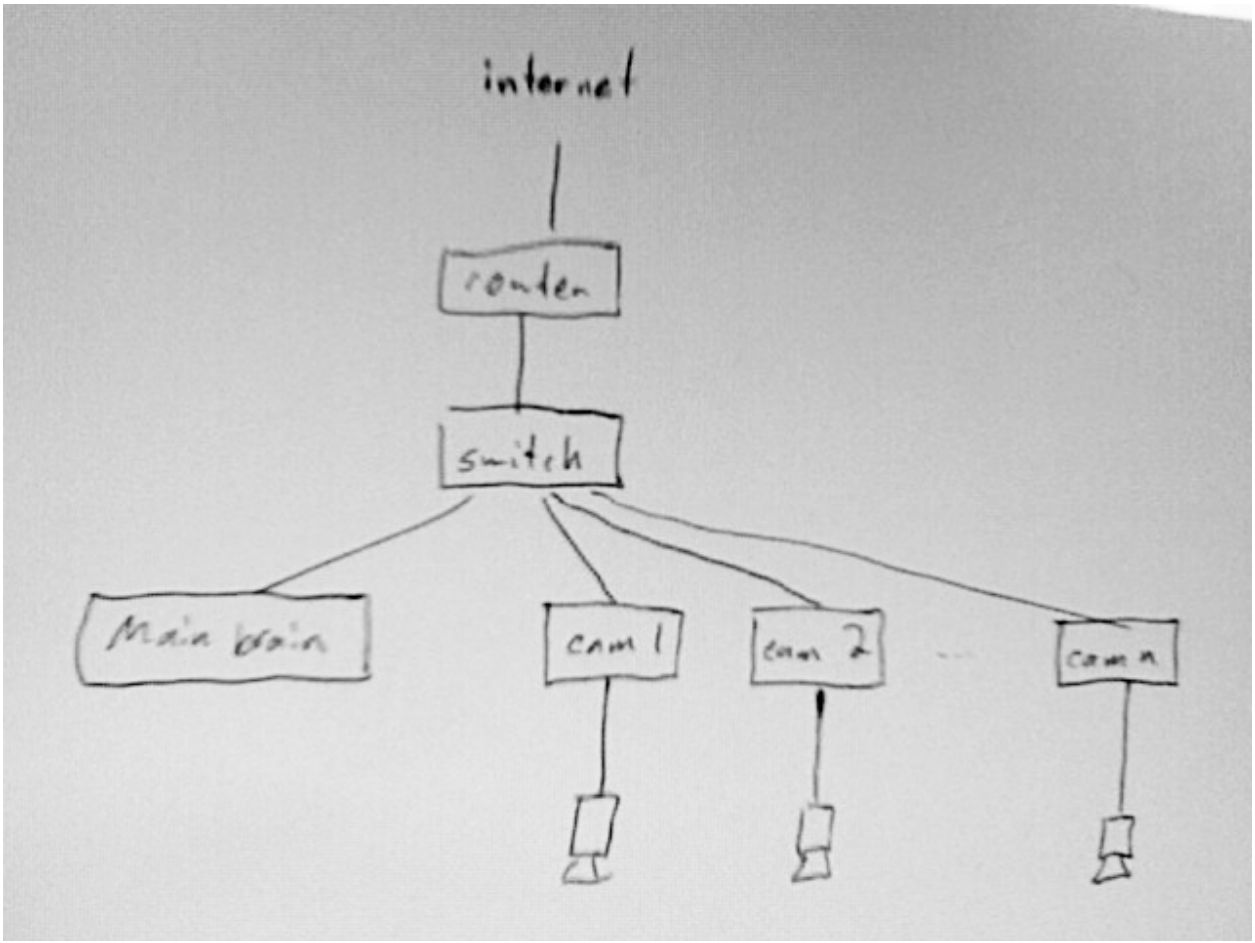
See Andrew's manuscript about flydra. Please send him any comments/questions/feedback about the manuscript as he is planning on submitting it.

12.1 Subpages about flydra

12.1.1 old Trac wiki page about Flydra Network Setup

This page was copied from the old Trac wiki page about Flydra and may have suffered some conversion issues.

Network topology



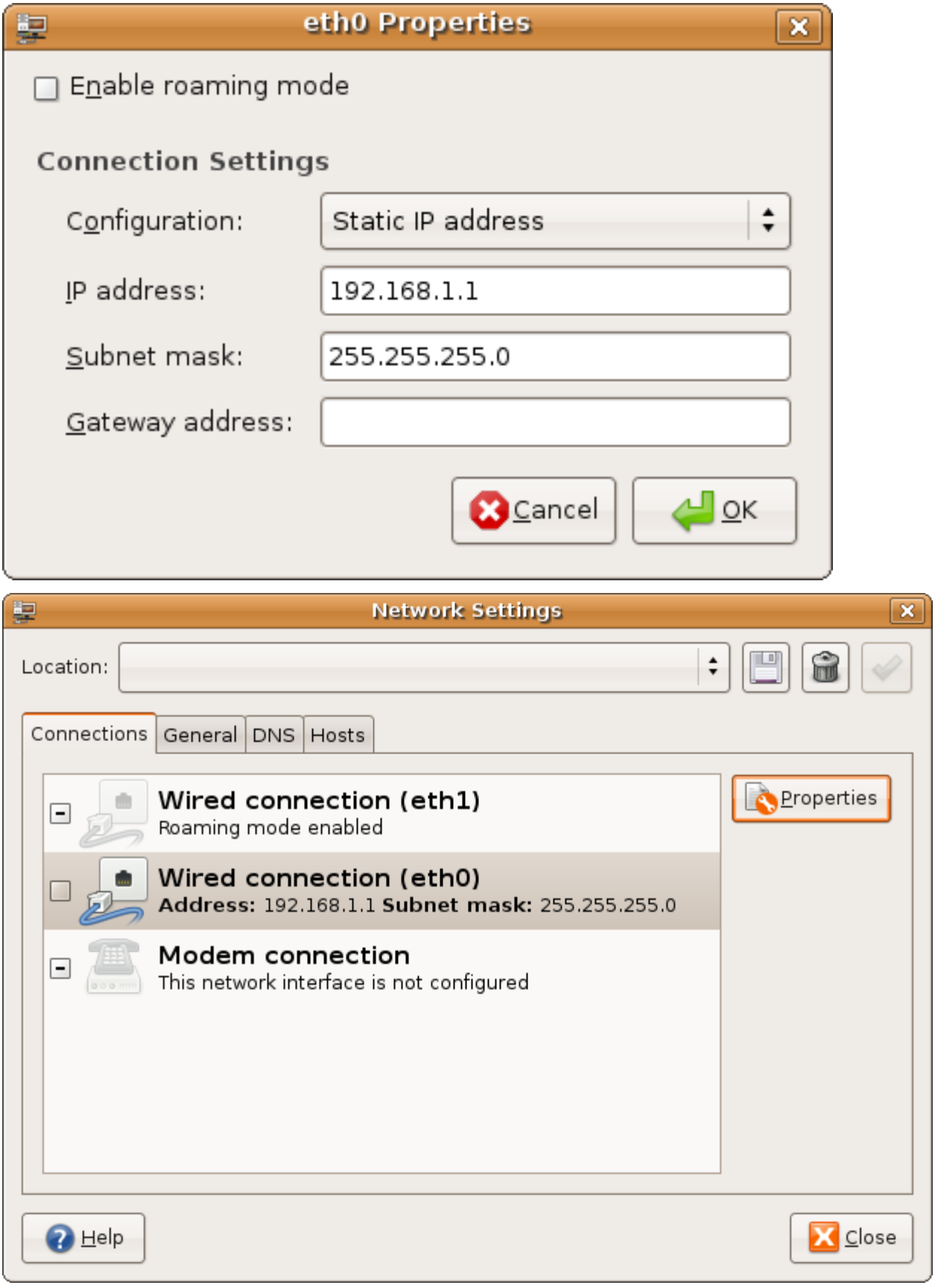
Setup of openssh to allow public key based authentication

It is very handy to setup public-key based authentication – many fewer passwords to type! I can't find a simple-enough tutorial right now.

(Note that host-based authentication is a real pain to setup.)

Specific to Prosilica cameras

The camera drivers (v 1.14 anyway) seem to have trouble unless the camera is on eth0. Also, they suggest using Intel GigE network adapters for the cameras. If the required port is not coming up as eth0, you can force it in Ubuntu by adjusting the file /etc/udev/rules.d/70-persistent-net.rules


eth0 Properties
Enable roaming mode
Connection Settings
Configuration:
Static IP address
IP address:
192.168.1.1
Subnet mask:
255.255.255.0
Gateway address:
Cancel
OK
Network Settings
Location:
Connections
General
DNS
Hosts
Wired connection (eth1)
Roaming mode enabled
Wired connection (eth0)
Address: 192.168.1.1 Subnet mask: 255.255.255.0
Modem connection
This network interface is not configured
Properties
Help
Close

12.1.2 old Trac wiki page about Running Flydra

This page was copied from the old Trac wiki page about Flydra and may have suffered some conversion issues.

Get the MainBrain GUI running

Starting the Precise Time Protocol Daemon (PTPd)

On the mainbrain (assuming it is running NTP):

```
# check NTP is running
ntpq -p
# There should be some output here indicating at least one NTP server with some␣
↪offset and jitter.
# (If there is an error or no servers are listed, see below.) (Also of interest is␣
↪the output
#  of "ntpdc -c kern").

# Run PTPd and disable frequency scaling:
# (The options below mean: identifier NTP, stratum 2, do not adjust system clock.)
sudo -s -H
echo "performance" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
echo "performance" > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
killall ptpd
nice -n -19 ptpd -s2 -i NTP -t
```

Doug/humdra - use the following instead of the last 2 lines (because your LAN is on eth1):

```
killall ptpd
nice -n -19 ptpd -b eth1 -s2 -i NTP -t
```

If NTP is not running, or the "ntpq -p" command returns errors, restart it with:

```
sudo /etc/init.d/ntp restart
```

On the camera computers:

```
# run PTPd and disable frequency scaling
sudo -s -H

# This next 2 lines are only necessary if your computer supports
# frequency scaling (if you have more than 2 cores, repeat accordingly).
echo "performance" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
echo "performance" > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor

killall ptpd
nice -n -19 ptpd
```

Doug/humdra - use the following instead of the last 2 lines (because your LAN is on eth1):

```
killall ptpd
nice -n -19 ptpd -b eth1
```

On any computer, you can ensure that PTPd is running by checking the list of running processes:

```
ps aux | grep ptpd
# There should be a line indicating a running ptpd process.
# (Be careful not to be confused by your "grep ptpd" process.)
```

Running the MainBrain GUI

On the main brain computer, run:

```
# You probably do NOT want to run this as root.
ydra_mainbrain
```

Running the camera nodes

Once the Main Brain GUI is up and running, run the following on the camera node computers:

```
# You probably do NOT want to run this as root.
./run_cam.sh
# (This should call "flydra_camera_node" with the appropriate options).
```

Note that some of old Basler cameras have broken firmware and require a trigger mode "2" to trigger properly.

(You may also be interested in some command line options. See `flydra_camera_node --help` for more information.

OLD - Running the camera nodes

"'Ignore this section – it is old."' Once the GUI is up and running, run the following on the camera node computers:

Prosilica cameras:

```
# If your camera is at IP address of 192.168.1.51, you can
# set the packet size on the command line with:
CamAttr -i 192.168.1.51 -s PacketSize 1500
CamAttr -i 192.168.1.51 -s StreamBytesPerSecond 123963084

# Now start the camera grabbing program:
shmwrap_prosilica_gige
```

Basler (or other 1394) cameras:

```
# To reset the 1394 bus (necessary if you get the "Could not allocate bandwidth"␣
↪error):
dc1394_reset_bus

# Now start the camera grabbing program:
shmwrap_dc1394
```

Now that the shared-memory camera grabber is running, run this:

```
flydra_camera_node --wrapper sharedmem --backend sharedmem --num-points=2
```

Load the Camera Configurations

This is currently broken. Don't do this right now – change the settings by hand. In `File->Open Camera Configuration`, open an appropriate configuration.

Load a calibration

Press the `Load Calibration...` button. Select an appropriate calibration.

Make sure cameras are synchronized

(As a prerequisite, the cameras must be in external trigger mode. This is usually Trig mode: "1" in the per-camera configuration in the main brain.)

Press the "Synchronize cameras" button. (Or, if you don't have a working Flydra Trigger Device, unplug your function generator for > 1 second.)

Basic operation

The green dots are tracked in 2D on the local cameras.

The red dots are the 3D reconstruction back-projected into the camera view.

The ongoing background estimate can be cleared (set to zero) on individual cameras by pressing the appropriate GUI button, or on all cameras by pressing the `<C>` key. The estimate can be set to the current image by doing `take` (or pressing the `<T>` key for all cameras). (Note: I always press `<C>` then `<T>` because I think there may be a small bug when just `<T>` is pressed.)

No longer useful for MainBrain

Getting the mainbrain computer able to talk to the trigger device

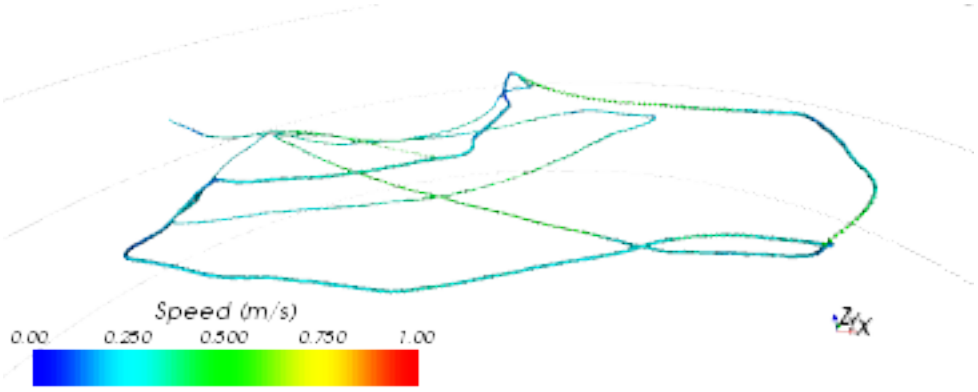
Prevent the linux kernel from de-powering the USB trigger device (or any USB device):

```
# login as root:
sudo -s -H
# (enter password)

echo -n -1 > /sys/module/usbcore/parameters/autosuspend
```

12.2 Scripts of great interest

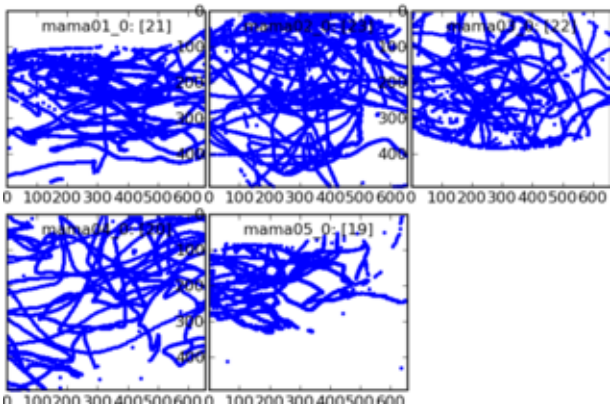
- source(command: kdviewer, file: flydra/a2/kdviewer.py) 3D viewer of Kalmanized trajectories saved in .h5 data file. (This is newer version of source(file: flydra/analysis/flydra_analysis_plot_kalman_data.py) that uses TVTK.)



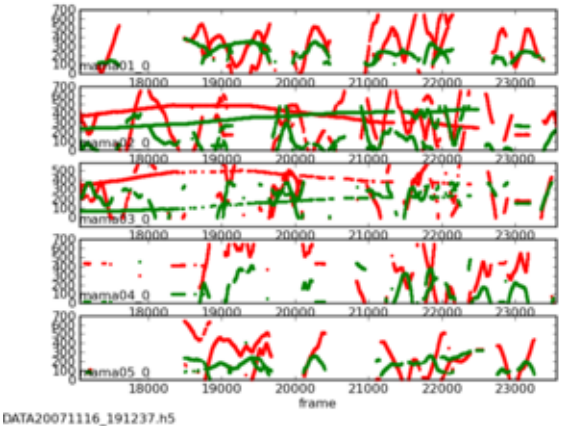
- source(command: flydra_kalmanize, file: flydra/kalman/kalmanize.py) re-analyze 2D data saved in .h5 data file using same Kalman filtering code as realtime analysis. Note: this will only run the causal Kalman filter, which allows re-segmenting the data into trajectories. However, another step, of running the non-causal smoothing algorithm, is typically desired. (See below for reasons why you might want to do this.)
- source(command: flydra_analysis_generate_recalibration, file: flydra/analysis/flydra_analysis_generate_recalibration.py) save 2D data from a previously saved .h5 file to perform a (re-)calibration. This new calibration typically has very low reprojection errors. Performing this step can use either 3D trajectories in the .h5 file in order to solve the correspondence problem (which 2D points from which cameras correspond to the same 3D point). Alternatively, if only (at most) a single point is tracked per camera per time point, they are assumed to be from the same 3D point. In this case, specify a start and stop frame.
- `ffpmeg` for example `ffmpeg -b 2008000 -f mpeg2video -r 25 -i image%05d.png movie.mpeg`
- source(command: flydra_analysis_convert_to_mat, file: flydra/analysis/flydra_analysis_convert_to_mat.py) convert .h5 3D data to .mat for MATLAB; includes raw observations and realtime kalman filtered observations.
- source(command: data2smoothed, file: flydra/a2/data2smoothed.py) convert .h5 3D data to .mat file for MATLAB; returns only results of 2nd pass kalman smoothing and timestamps
- source(command: flydra_textlog2csv, file: flydra/a2/flydra_textlog2csv.py) save text log from .h5 file to CSV format (can be opened in Excel, for example)

12.3 Scripts of lesser interest

- source(command: flydra_analysis_filter_kalman_data.py, file: flydra/analysis/flydra_analysis_filter_kalman_data.py) export a subset of the Kalmanized trajectories in an .h5 file to a new (smaller) .h5 file.
- source(command: flydra_analysis_plot_kalman_2d.py, file: flydra/analysis/flydra_analysis_plot_kalman_2d.py) plot 2D data (x against y), including Kalmanized trajectories. (This is more-or-less the camera view.)



- source(command: plot_timeseries_2d_3d.py, file: flydra/a2/plot_timeseries_2d_3d.py plot_timeseries_2d_3d.py) plot 2D and 3D data against frame number. (This is a time series.)



- source(command: plot_timeseries.py, file: flydra/a2/plot_timeseries.py) plot 3D data against frame number. (This is a time series.)
- source(command: flydra_images_export, file: flydra/a2/flydra_images_export.py) export images from .h5 file.

12.4 Reasons to run flydra_kalmanize on your data, even though it's already been Kalmanized

- Re-running uses all the 2D data, including that which was originally dropped during realtime transmission to the mainbrain.
- (If a time-budget for 3D calculations ever gets implemented, this running this will also bypass any time-budget based cuts.)
- You have a new (better) calibration.
- You have a new (better) set of Kalman parameters and model.

12.5 Image masking

Export camera views from the .h5 file:

```
flydra_images_export DATA20080402_115551.h5
```

This should have saved files named cam_id.bmp, (e.g. "cam1_0.bmp"). Open these in GIMP (the GNU image manipulation program).

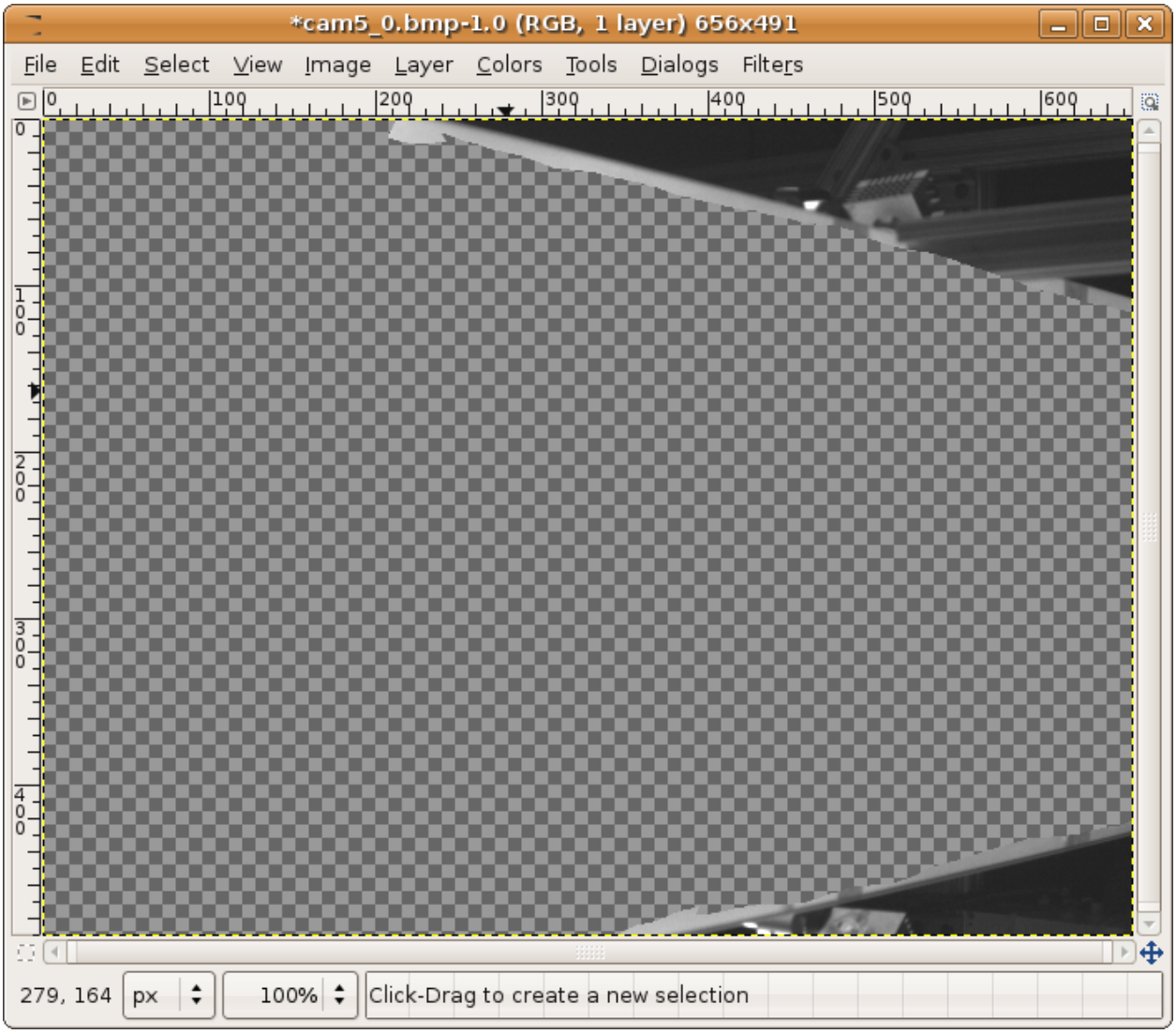
Add an alpha channel: open the Layers dialog (if it's not already open). Right click on the layer (called "Background"), then select "Add alpha channel".

Convert to RGB: Use the "Image" menu in the menubar: "Image->Mode->RGB". (It would be nice if flydra supported grayscale images in addition to RGB, but that's not yet implemented.)

Use the free select tool (like a lasso), to select the region you want to ignore using the tool. Hints: You can select multiple areas by holding down the 'shift' key. You can drag the lasso outside of the image and it will just draw along the edge of the image. Next invert the selection "Select->Invert" and cut the region you want to track over (Ctrl-X).

The alpha channel may not have any values except 0 and 255, therefore if you are not completely sure, use LAYERS/TRANSPARENCY/THRESHOLD ALPHA in Gimp to create a clean alpha channel.

Here's an example of what you're looking for. The important point is that you'll be tracking over the region where this image is transparent.



Finally, "File->Save As…" and save as a .png file. The default options are fine.

If you saved your file as "cam5_0_mask.png", you would use this with a command like:

```
flydra_camera_node --mask-images=cam5_0_mask.png
```

in a launch file, this would for example look like this:

```
<node name="flydra_camera_node" pkg="ros_flydra" type="camnode" args="--num-
↪buffers=100  --background-frame-alpha=0.01 --background-frame-interval=80 --num-
↪points=6 --sleep-first=5 --mask-images=$(find flycave)/calibration/flycube1/oct_
↪2013/Basler_21275576_mask.png:$(find flycave)/calibration/flycube1/oct_2013/Basler_
↪21275577_mask.png:$(find flycave)/calibration/flycube1/oct_2013/Basler_21283674_
↪mask.png:$(find flycave)/calibration/flycube1/oct_2013/Basler_21283677_mask.png:
↪$(find flycave)/calibration/flycube1/oct_2013/Basler_21359437_mask.png" />
```

Watch out! The masks in the launch file (e.g. 'flydra.node' are in the OPPOSITE ORDER than the camera names in the .yaml file (e.g. 'flydra.yaml'). You can check this in the console output of the camnode:

```
--------------------------------------------------------------------------------
↪--------------
```

```
Camera guid ='Basler_21275577'
has mask image: '/opt/ros/ros-flycave.electric.boost1.46/flycave/calibration/flycube1/
↪oct_2013/Basler_21275577_mask.png'
```

Note that, as flydra_camera_node supports multiple cameras, you may specify multiple mask images – one for each camera. In these case, they are separated by the (OS-specific) path separator. This is ':' on linux.

12.6 Latency/performance of flydra

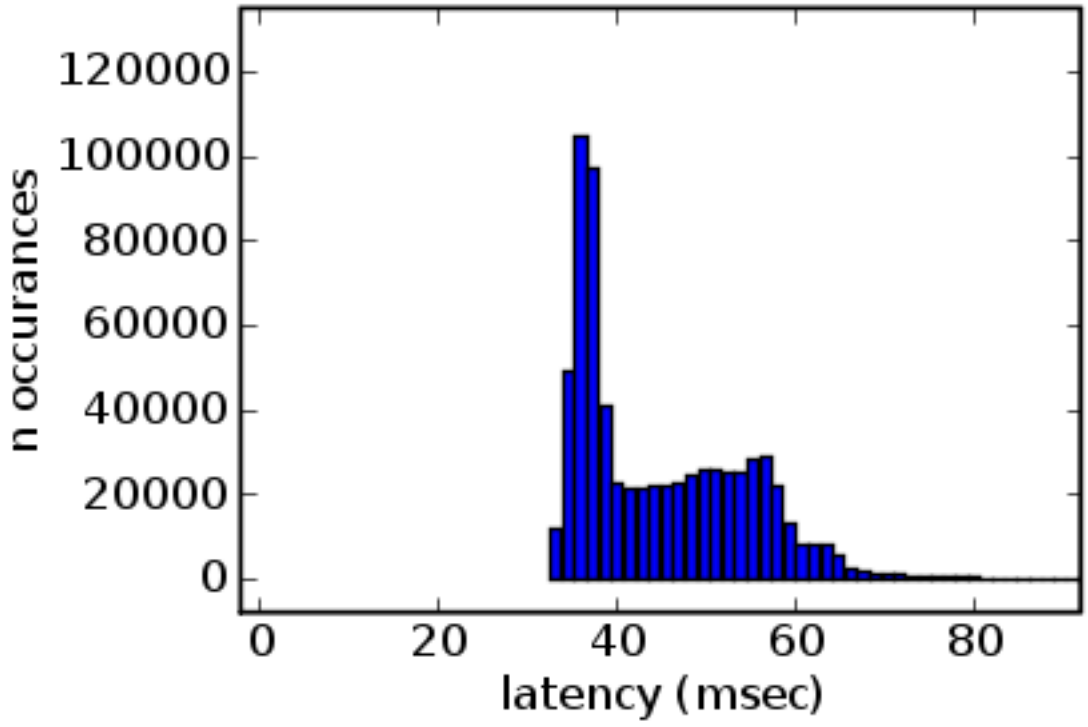
Back in March 2007 or so, Andrew Straw measured the total latency of flydra. At the time of writing (November 6, 2007) I remember the shorter latencies to be approx. 55 msec. Most latencies were in this range, but not being a hard-realtime system, some tended to be longer. Most latencies, however were under 80 msec.

Today (November 6, 2007) I have done some more experimentation and added some profiling options. Here's the breakdown of times:

- 20 msec from the Basler A602f cameras. (This is a fundamental limit of 1394a, as it takes this long for the image to get to the host computer. 1394b or GigE cameras would help.)
- 10 msec for local, 2D image processing and coordinate extraction on a 3.0 GHz Pentium 4 (w/HT) computer when `flydra_camera_node` is using `--num-points=3`.
- 1 msec transmission delay to the Main Brain.
- 15-30 msec for 3D tracking when `flydra_camera_node` is using `--num-points=3`. (This drops to less than 10 msec with `--num-points=1`.) This test was performed with an Intel Core 2 Duo 6300 @ 1.86GHz as the Main Brain.

The values above add up roughly to the values I remember from earlier in 2007, so I guess this is more-or-less what was happening back then.

As of Nov. 16, 2007, the `check_atmel` script was used to generate this figure, which is total latency to reconstructed 3D on the mainbrain with `flydra_camera_node` using `--num-points=2`. This includes the 20 msec Basler A602f latency, so presumably for GigE, you'd subtract approx. 15 msec. Faster 2D camera computers would probably knock off another 5 msec.



As of Feb. 20, 2008, the *LED_test_latency* script was used to test total latency from LED on to UDP packets with position information. This total latency was between 34-70 msec, with the 3D reconstruction latency being from about 25-45 msec. Note that these are from the initial onset of illumination, which may have different amounts of latencies than ongoing tracking.

12.7 Profiling

To profile the realtime Kalman filter code, first run the `flydra_mainbrain` app with the `--save-profiling-data` option. You will have to load the 3D calibration, synchronize the cameras and begin tracking. The data sent to the Kalman tracker is accummulated to RAM and saved to file when you quit the program. There is no UI indication this is happening except when you finally quit it will mention the filename. With the saved data file, run source(command: kalman_profile.py, file: flydra/kalman/kalman_profile.py). This will generate profiling output that can be opened in kcachegrind.

12.8 Flydra simulator

12.8.1 Option 1: Play FMFs through flydra_camera_node

This approach requires pre-recorded .fmf files.

Run the mainbrain:

```
REQUIRE_FLYDRA_TRIGGER=0 flydra_mainbrain
```

Now run an image server:

```
flydra_camera_node --emulation-image-sources full_20080626_210454_mama01_0.fmf --wx
```

12.8.2 Option 2: Play .h5 file

This approach is not yet implemented.

CHAPTER 13

Frequently Asked Questions

13.1 The `timestamp` field is all wrong!

The `timestamp` field for 3D reconstructions is not the time when the data was taken, but when it was done processing. To get timestamps spaced at the inter-frame interval, use `frame * (1.0/fps)`.

CHAPTER 14

Installation notes

14.1 RedHat 64bit

(as of September 2010)

14.1.1 For making pytables install correctly

Install cython:

```
$ pip install cython
```

Download and install a binary distribution of HDF5.

```
$ wget http://www.hdfgroup.org/ftp/HDF5/current/src/hdf5-1.8.5-patch1.tar.gz
$ ./configure --prefix=<dir>
$ make
$ make install
```

Also:

```
$ export HDF5_DIR=<dir>
```

so that pytables can find it.

Also install LZO and remember to do:

```
$ export LZO_DIR=<dir>
```

14.1.2 cgtypes

Download cgkit-1.2.0 from sourceforge.

Use pyrex to recompile cgtypes:

```
$ cd cgkit-1.2.0
$ pyrex cgtypes.pyx
$ python setup.py install
```

14.1.3 Finally

At this point:

```
$ cd flydra
$ python setup.py develop
```

14.2 Mac OS X

`motmot.FastImage` cannot be installed, so you have to do a "light" installation.

Apart from that, everything else can be installed.

14.3 Ubuntu Lucid

(as of October 2010)

Before installing pytables, you have to install the `lzo` library, otherwise you won't be able to read some compressed files created by flydra.

Here's a suitable sequence of commands:

```
$ sudo apt-get install liblzo2-dev
$ sudo apt-get install libbzip2-dev
$ pip install numexpr=1.3
$ pip install tables
```

CHAPTER 15

Contibutions to this documentation

All contributions are welcome. If you'd like to improve something, look into the sources if they contain the information you need (if not, please fix them), otherwise the documentation generation needs to be improved (look in the `flydra-sphinx-docs/` directory). For more information, see *Editing the documentation*.

CHAPTER 16

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

F