Oat is a set of programs for processing images, extracting object position information, and streaming data to disk and/or the network in real-time. Oat subcommands are independent programs that each perform a single operation but that can communicate through shared memory. This allows a user to chain operations together in arrangements suitable for particular context or tracking requirement. This architecture enables scripted construction of custom data processing chains. Oat is primarily used for real-time animal position tracking in the context of experimental neuroscience, but can be used in any circumstance that requires real-time object tracking.

Contributors

- jonnew http://www.mit.edu/~jpnewman/

Table of Contents

Manual

Introduction

Oat's design is influenced by the UNIX philosophy, suckless tools, and MEABench. Oat consists of a set of small, composable programs (called **components**). Components are equipped with standard interfaces that permit communication through shared memory to capture, process, and record video streams. Currently, Oat components act on two basic data types: `frames` and `positions`.

- `frame` - Video frame.
- `position` - 2D position.

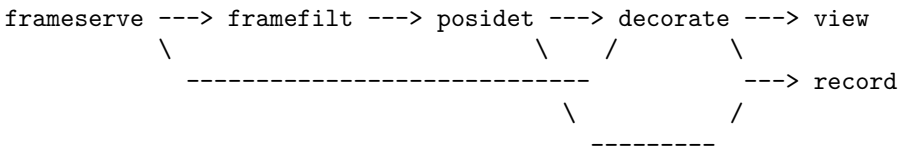
Oat components can be chained together to realize custom dataflow networks that operate on instances of the aforementioned datatypes, called **tokens**. Token processing pipelines can be split and merged while maintaining thread-safety and sample synchronization. The messaging library underlying the communication between Oat components has been optimized to reduce token copying. For instance, `frame` passing is performed using a zero-copy protocol. This means that passing `frames` between components in a user-configured processing network incurs almost no memory or CPU cost compared to the monolithc equivalent. Further, great care was taken during implementations of Oat components to minimize time spent in critical sections. This means that individual components execute largely in parallel, even when components are highly interdependent, facilitating efficient use of multi-core CPUs and GPU-based processing acceleration.

To get a feel for how Oat is used, here is a script to detect the position of a single object in pre-recorded video file:

```
# Serve frames from a video file to the 'raw' stream
oat frameserve file raw -f ./video.mpg &

# Perform background subtraction on the 'raw' stream
# Serve the result to the 'filt' stream
# If an appropriately configured GPU is available, this process will
# use it
oat framefilt mog raw filt &

# Perform color-based object position detection on the 'filt' stream
# Serve the object positionto the 'pos' stream. Allow parameter tuning
# through a simple GUI.
oat posidet hsv filt pos --tune &

# Decorate the 'raw' stream with the detected position form the `pos` stream
# Serve the decorated images to the 'dec' stream
oat decorate -p pos raw dec &
```

```
# View the 'dec' stream
oat view dec &

# Record the 'dec' and 'pos' streams to file in the current directory
oat record -i dec -p pos -f ./
```

This script has the following graphical representation:



Generally, an Oat component is called in the following pattern:

`oat <subcommand> [TYPE] [IO] [CONFIGURATION]`

The `<subcommand>` indicates the component that will be executed. Components are classified according to their type signature. For instance, `framefilt` (frame filter) accepts a frame and produces a frame. `posifilt` (position filter) accepts a position and produces a position. `frameserve` (frame server) produces a frame, and so on. The `TYPE` parameter specifies a concrete type of transform (e.g. for the `framefilt` subcommand, this could be `bsub` for background subtraction). The `IO` specification indicates where the component will receive data from and to where the processed data should be published. The `CONFIGURATION` specification is used to provide parameters to shape the component's operation. Aside from command line options and switches, which are listed using the `--help` option for each subcommand, the user can often provide an external file containing a configuration table to pass parameters to a component. Some configuration parameters can only be specified using a configuration file. Configuration files are written in plain text using TOML. A multi-component processing script can share a configuration file because each component accesses parameter information using a file/key pair, like so

```
[key]
parameter_0 = 1                 # Integer
parameter_1 = true              # Boolean
parameter_2 = 3.14              # Double
parameter_3 = [1.0, 2.0, 3.0]   # Array of doubles
```

or more concretely,

```
# Example configuration file for frameserve --> framefilt
[frameserve-config]
frame_rate = 30                 # FPS
roi = [10, 10, 100, 100]        # Region of interest

[framefilt-config]
mask = "~/Desktop/mask.png"     # Path to mask file
```

The type and sanity of parameter values are checked by Oat before they are used. Below, the type signature, usage information, available configuration parameters, examples, and configuration options are provided for each Oat component.

Frame Server

`oat-frameserve` - Serves video streams to shared memory from physical devices (e.g. webcam or GIGE camera) or from file.

Signature

```
oat-frameserve --> frame
```

Usage

```
Usage: frameserve [INFO]
   or: frameserve TYPE SINK [CONFIGURATION]
Serve frames to SINK.

INFO:
  --help                  Produce help message.
  -v [ --version ]        Print version information.

TYPE
  wcam: Onboard or USB webcam.
  usb: Point Grey USB camera.
  gige: Point Grey GigE camera.
  file: Video from file (*.mpg, *.avi, etc.).
  test: Write-free static image server for performance testing.

SINK:
  User-supplied name of the memory segment to publish frames to (e.g. raw).
```

Configuration File Options

TYPE = `gige`

- **`index`**=`+int` User specified camera index. Useful in multi-camera imaging configurations.
- **`fps`**=`+float` Acquisition frame rate (Hz). Ignored if `trigger_on=true` and `enforce_fps=false`. If unspecified, then the maximum frame rate will be used.
- **`exposure`**=`float` Automatically adjust both shutter and gain to achieve given exposure (EV).
- **`shutter`**=`+float` Shutter time in milliseconds. Specifying `exposure` overrides this option.
- **`gain`**=`float` Sensor gain value. Specifying `exposure` overrides this option (dB).
- **`white_bal`**=`{red=+int, blue=+int}` White-balance specified as red/blue intensity values (0-1000).

- `roi={x_offset=+int, y_offset=+int, width=+int, height+int}` Region of interest to extract from the camera or video stream (pixels).
- `trigger_on=bool` True to use camera trigger, false to use software polling.
- `trigger_rising=bool` True to trigger on rising edge, false to trigger on falling edge.
- `trigger_mode=+int` Point-grey trigger mode. Common values are:
  - 0 - Standard external trigger. Trigger edge causes sensor exposure, then sensor readout to internal memory.
  - 1 - Blub shutter mode. Same as 0, except that sensor exposure duration is determined by trigger active duration.
  - 7 - Continuous internal trigger. No external trigger required, but not synchronized to an external clock.
  - 14 - Overlapped exposure/readout external trigger. Sensor exposure occurs during sensory readout to internal memory. This is the fastest external trigger mode.
- `trigger_pin=+int` Hardware pin number on Point-grey camera that trigger is sent to.
- `strobe_pin=+int` Hardware pin number on Point-grey camera that a gate signal for the camera shutter is copied to.
- `enforce_fps=bool`If true, ensures that frames are produced at the `fps` setting by retransmitting frames if the requested period is exceeded. This is sometimes needed in the case of an external trigger because PG cameras sometimes just ignore them. I have opened a support ticket on this, but PG has no solution yet.

TYPE = `file`

- `fps=float` Target frame rate in frames per second. If left undefined, frames will be read as quickly as possible.
- `roi={x_offset=+int, y_offset=+int, width=+int, height+int}` Region of interest to extract from the camera or video stream (pixels).

TYPE = `wcam`

- `index=+int` User specified camera index. Useful in multi-camera imaging configurations.

Examples

```
# Serve to the 'wraw' stream from a webcam
oat frameserve wcam wraw

# Stream to the 'graw' stream from a point-grey GIGE camera
# using the gige_config tag from the config.toml file
oat frameserve gige graw -c config.toml gige_config

# Serve to the 'fraw' stream from a previously recorded file
```

```
# using the file_config tag from the config.toml file
oat frameserve file fraw -f ./video.mpg -c config.toml file_config
```

Frame Filter

`oat-framefilt` - Receive frames from a frame source, filter, and publish to a second memory segment. Generally used to pre-process frames prior to object position detection. For instance, `framefilt` could be used to perform background subtraction or application of a mask to isolate a region of interest.

Signature

```
frame --> oat-framefilt --> frame
```

Usage

```
Usage: framefilt [INFO]
   or: framefilt TYPE SOURCE SINK [CONFIGURATION]
Filter frames from SOURCE and publish filtered frames to SINK.

INFO:
  --help                Produce help message.
  -v [ --version ]      Print version information.

TYPE
  bsub: Background subtraction
  mask: Binary mask
  mog: Mixture of Gaussians background segmentation.
  undistort: Compensate for lens distortion using distortion model.

SOURCE:
  User-supplied name of the memory segment to receive frames from (e.g. raw).

SINK:
  User-supplied name of the memory segment to publish frames to (e.g. filt).
```

Configuration File Options

TYPE = `bsub`

- **`background`**=`string` Path to a background image to be subtracted from the SOURCE frames. This image must have the same dimensions as frames from SOURCE.
- **`adaption-coeff`**=`+float` Value, 0 to 1.0, specifying how quickly the new frames are used to update the backgound image. Default is 0, specifying no adaptation and a static background image that is never updated. **TYPE = `mask`**

- `mask`=`string` Path to a binary image used to mask frames from SOURCE. SOURCE frame pixels with indices corresponding to non-zero value pixels in the mask image will be unaffected. Others will be set to zero. This image must have the same dimensions as frames from SOURCE.

TYPE = `mog`

- `learning-coeff`=`+float` Value, 0 to 1.0, specifying how quickly the statistical model of the background image should be updated. Default is 0, specifying no adaptation.
- `gpu-index`=`+int` Index of the GPU to use for performing background subtraction if Oat was compiled with CUDA support.

TYPE = `undistort`

- `camera-model`=`+int` Value, 0 or 1, specifying the camera model to use. 0 specifies a Pinhole camera model, 1 specifies fisheye. Generated by oat-calibrate.
- `camera-matrix`= `[+float,+float,+float,+float,+float,+float,float,+float,+float]`, Camera matrix for your imaging setup. Generated by oat-calibrate.
- `distortion-coeffs`= `[+float,+float,+float,+float,+float, ...]`, Five to eight element vector specifying lens distortion coefficients. Generated by oat-calibrate.
- `rotation`=`+double` Counter clockwise Degrees that undistorted image should be rotated. If not specified, defaults to 0.0.

Examples

```
# Receive frames from 'raw' stream
# Perform background subtraction using the first frame as the background
# Publish result to 'sub' stream
oat framefilt bsub raw sub

# Receive frames from 'raw' stream
# Apply a mask specified in a configuration file
# Publish result to 'roi' stream
oat framefilt mask raw roi -c config.toml mask-config
```

Frame Viewer

`oat-view` - Receive frames from named shared memory and display them on a monitor. Additionally, allow the user to take snapshots of the currently displayed frame by pressing s while the display window is in focus.

Signature

```
frame --> oat-view
```

Usage

```
Usage: view [INFO]
   or: view TYPE SOURCE [CONFIGURATION]
Graphical visualization of SOURCE stream.

INFO:
  --help                 Produce help message.
  -v [ --version ]       Print version information.

TYPE
  frame: Display frames in a GUI

SOURCE:
  User-supplied name of the memory segment to receive frames from (e.g. raw).
```

Example

```
# View frame stream named raw
oat view frame raw

# View frame stream named raw and specify that snapshots should be saved
# to the Desktop with base name 'snapshot'
oat view frame raw -f ~/Desktop -n snapshot
```

Position Detector

`oat-posidet` - Receive frames from named shared memory and perform object position detection within a frame stream using one of several methods. Publish detected positions to a second segment of shared memory.

Signature

```
frame --> oat-posidet --> position
```

Usage

```
Usage: posidet [INFO]
   or: posidet TYPE SOURCE SINK [CONFIGURATION]
Perform object detection on frames from SOURCE and publish object positions to SINK.

INFO:
  --help                Produce help message.
  -v [ --version ]      Print version information.

TYPE
  diff: Difference detector (grey-scale, motion)
  hsv : HSV detector (color)

SOURCE:
  User-supplied name of the memory segment to receive frames from (e.g. raw).

SINK:
  User-supplied name of the memory segment to publish positions to (e.g. pos).
```

Configuration File Options

TYPE = `hsv`

- **`tune`**=`bool` Provide GUI sliders for tuning hsv parameters
- **`erode`**=`+int` Candidate object erosion kernel size (pixels)
- **`dilate`**=`+int` Candidate object dilation kernel size (pixels)
- **`min_area`**=`+double` Minimum object area (pixels2)
- **`max_area`**=`+double` Maximum object area (pixels2)
- **`h_thresholds`**=`{min=+int, max=+int}` Hue pass band
- **`s_thresholds`**=`{min=+int, max=+int}` Saturation pass band
- **`v_thresholds`**=`{min=+int, max=+int}` Value pass band

TYPE = `diff`

- **`tune`**=`bool` Provide GUI sliders for tuning diff parameters
- **`blur`**=`+int` Blurring kernel size (normalized box filter; pixels)

- **diff_threshold**=+int Intensity difference threshold

Example

```
# Use color-based object detection on the 'raw' frame stream
# publish the result to the 'cpos' position stream
# Use detector settings supplied by the hsv_config key in config.toml
oat posidet hsv raw cpos -c config.toml hsv_config

# Use motion-based object detection on the 'raw' frame stream
# publish the result to the 'mpos' position stream
oat posidet diff raw mpos
```

Position Generator

`oat-posigen` - Generate positions for testing downstream components. Publish generated positions to shared memory.

Signature

```
oat-posigen --> position
```

Usage

```
Usage: posigen [INFO]
   or: posigen TYPE SINK [CONFIGURATION]
Publish generated positions to SINK.

TYPE
  rand2D: Randomly accelerating 2D Position

SINK:
  User supplied position sink name (e.g. pos).

OPTIONAL ARGUMENTS:

INFO:
  --help                    Produce help message.
  -v [ --version ]          Print version information.

CONFIGURATION:
  -r [ --rate-hz ] arg      Samples per second. Overriden by information in
                            configuration file if provided. Defaults to as fast
                            as possible.
  -n [ --num-samples ] arg  Number of position samples to generate and serve.
                            Overriden by information in configuration file if
                            provided. Deafaults to approximately infinite.
  -c [ --config ] arg       Configuration file/key pair.
```

Configuration File Options

TYPE = `rand2D`

- **`rate-hz`**=`+double` Position update rate in Hz.
- **`num-samples`**=`+int` Number of position samples to produce.
- **`room`**=`[+double, +double, +double, +double]` The 'room' in which generated positions reside specified as [x origin, y origin, width, height]. Arbitrary units. The room has periodic boundaries so when a position leaves one side it will enter the opposing one.

Example

```
# Publish randomly moving positions to the 'pos' position stream
oat posigen rand2D pos
```

Position Filter

`oat-posifilt` - Receive positions from named shared memory, filter, and publish to a second memory segment. Can be used to, for example, remove discontinuities due to noise or discontinuities in position detection with a Kalman filter or annotate categorical position information based on user supplied region contours.

Signature

```
position --> oat-posifilt --> position
```

Usage

```
Usage: posifilt [INFO]
   or: posifilt TYPE SOURCE SINK [CONFIGURATION]
Filter positions from SOURCE and publish filtered positions to SINK.

INFO:
  --help                Produce help message.
  -v [ --version ]      Print version information.

TYPE
  kalman: Kalman filter
  homography: homography transform
  region: position region annotation

SOURCE:
  User-supplied name of the memory segment to receive positions from (e.g. pos).

SINK:
  User-supplied name of the memory segment to publish positions to (e.g. filt).
```

Configuration File Options

TYPE = `kalman`

- **`dt`**=`+float` Sample period (seconds).
- **`timeout`**=`+float` Time to perform position estimation detection with lack of updated position measure (seconds).
- **`sigma_accel`**=`+float` Standard deviation of normally distributed, random accelerations used by the internal model of object motion (position units/s2; e.g. pixels/s2).
- **`sigma_noise`**=`+float` Standard deviation of randomly distributed position measurement noise (position units; e.g. pixels).
- **`tune`**=`bool` Use the GUI to tweak parameters.

TYPE = `homography`

- **`homography`** = `[+float,+float,+float, +float,+float,+float,+float,+float,+float]`, Homography matrix for 2D position (1x9; world units/pixel). Generate using oat-calibrate.

TYPE = `region`

- **`<regions>`**=`[[+float, +float],[+float, +float],...,[+float, +float]]` User-named region contours (pixels). Regions counters are specified as `n`-point matrices, `[[x0, y0],[x1, y1],...,[xn, yn]]`, which define the vertices of a polygon. The name of the contour is used as the region label. For example, here is an octagonal region called `CN` and a tetragonal region called `R0`:

```
# These regions could be named anything...
CN = [[336.00, 272.50],
      [290.00, 310.00],
      [289.00, 369.50],
      [332.67, 417.33],
      [389.33, 413.33],
      [430.00, 375.33],
      [433.33, 319.33],
      [395.00, 272.00]]

R0 = [[654.00, 380.00],
      [717.33, 386.67],
      [714.00, 316.67],
      [655.33, 319.33]]
```

Example

```
# Perform Kalman filtering on object position from the 'pos' position stream
# publish the result to the 'kpos' position stream
# Use detector settings supplied by the kalman_config key in config.toml
oat posifilt kalman pos kfilt -c config.toml kalman_config
```

Position Combiner

`oat-posicom` - Combine positions according to a specified operation.

Signature

```
position 0 --> |
position 1 --> |
  :            | oat-posicom --> position
position N --> |
```

Usage

```
Usage: posicom [INFO]
   or: posicom TYPE SOURCES SINK [CONFIGURATION]
Combine positional information from two or more SOURCES and Publish combined position to SI

INFO:
  --help                Produce help message.
  -v [ --version ]      Print version information.

TYPE
  mean: Geometric mean of positions

SOURCES:
  User-supplied position source names (e.g. pos1 pos2).

SINK:
  User-supplied position sink name (e.g. pos).
```

Configuration File Options

TYPE = `mean`

- **`heading_anchor`**=`+int` Index of the SOURCE position to use as an anchor when calculating object heading. In this case the heading equals the mean directional vector between this anchor position and all other SOURCE positions. If unspecified, the heading is not calculated.

Example

```
# Generate the geometric mean of 'pos1' and 'pos2' streams
# Publish the result to the 'com' stream
oat posicom mean pos1 pos2 com
```

Frame Decorator

`oat-decorate` - Annotate frames with sample times, dates, and/or positional information.

Signature

```
     frame --> |
position 0 --> |
position 1 --> | oat-decorate --> frame
  :            |
position N --> |
```

Usage

```
Usage: decorate [INFO]
   or: decorate SOURCE SINK [CONFIGURATION]
Decorate the frames from SOURCE, e.g. with object position markers and sample number. Publis

SOURCE:
  User-supplied name of the memory segment from which frames are received (e.g. raw).

SINK:
  User-supplied name of the memory segment to publish frames to (e.g. out).

INFO:
  --help                          Produce help message.
  -v [ --version ]                Print version information.

CONFIGURATION:
  -c [ --config ] arg             Configuration file/key pair.
                                  e.g. 'config.toml mykey'

  -p [ --position-sources ] arg   The name of position SOURCE(s) used to draw
                                  object position markers.

  -t [ --timestamp ]              Write the current date and time on each
                                  frame.

  -s [ --sample ]                 Write the frame sample number on each frame.

  -S [ --sample-code ]            Write the binary encoded sample on the corner
                                  of each frame.

  -R [ --region ]                 Write region information on each frame if
```

```
                                    there is a position stream that contains it.

  -h [ --history ]                  Display position history.
```

Example

```
# Add textual sample number to each frame from the 'raw' stream
oat decorate raw -s

# Add position markers to each frame from the 'raw' stream to indicate
# objection positions for the 'pos1' and 'pos2' streams
oat decorate raw -p pos1 pos2
```

Recorder

`oat-record` - Save frame and position streams to file.

- `frame` streams are compressed and saved as individual video files (H.264 compression format AVI file).
- `position` streams are combined into a single JSON file. Position files have the following structure:

```
{oat-version: X.X},
{header: {timestamp: YYYY-MM-DD-hh-mm-ss},
         {sample_rate_hz: X.X},
         {sources: [ID_1, ID_2, ..., ID_N]} }
{positions: [ [ID_1: position, ID_2: position, ..., ID_N: position ],
              [ID_1: position, ID_2: position, ..., ID_N: position ],

              [ID_1: position, ID_2: position, ..., ID_N: position ] }
}
```

where each `position` object is defined as:

```
{
  samp: Int,                  | Sample number
  unit: Int,                  | Enum spcifying length units (0=pixels, 1=meters)
  pos_ok: Bool,               | Boolean indicating if position is valid
  pos_xy: [Double, Double],   | Position x,y values
  vel_ok: Bool,               | Boolean indicating if velocity is valid
  vel_xy: [Double, Double],   | Velocity x,y values
  head_ok: Bool,              | Boolean indicating if heading  is valid
  head_xy: [Double, Double],  | Heading x,y values
  reg_ok: Bool,               | Boolean indicating if region tag  is valid
  reg: String                 | Region tag
}
```

Data fields are only populated if the values are valid. For instance, in the case that only object position is valid, and the object velocity, heading, and region information are not calculated, an example position data point would look like this:

```
{ samp: 501,
  unit: 0,
  pos_ok: True,
  pos_xy: [300.0, 100.0],
  vel_ok: False,
  head_ok: False,
  reg_ok: False }
```

All streams are saved with a single recorder have the same base file name and save location (see usage). Of course, multiple recorders can be used in parallel

to (1) parallelize the computational load of video compression, which tends to be quite intense and (2) save to multiple locations simultaneously.

Signature

```
position 0 --> |
position 1 --> |
  :            |
position N --> |
               | oat-record
   frame 0 --> |
   frame 1 --> |
     :         |
   frame N --> |
```

Usage

```
Usage: record [INFO]
   or: record [CONFIGURATION]
Record frame and/or position streams.

INFO:
  --help                          Produce help message.
  -v [ --version ]                Print version information.

CONFIGURATION:
  -s [ --frame-sources ] arg      The names of the FRAME SOURCES that supply
                                  images to save to video.
  -p [ --position-sources ] arg   The names of the POSITION SOURCES that supply
                                  object positions to be recorded.
  -n [ --filename ] arg           The base file name. If not specified, defaults
                                  to the SOURCE name.
  -f [ --folder ] arg             The path to the folder to which the video
                                  stream and position data will be saved. If not
                                  specified, defaults to the current directory.
  -d [ --date ]                   If specified, YYYY-MM-DD-hh-mm-ss_ will be
                                  prepended to the filename.
  -o [ --allow-overwrite ]        If set and save path matches and existing
                                  file, the file will be overwritten instead of
                                  a incremental numerical index being appended
                                  to the file name.
  -c [ --concise-file ]           If set, indeterminate position data fields
                                  will not be written e.g. pos_xy will not be
                                  written even when pos_ok = false. This means
                                  that position objects will be of variable size
```

	depending on the validity on whether a position was detected or not, potentially complicating file parsing.
--interactive	Start recorder with interactive controls enabled.
--rpc-endpoint arg	Yield interactive control of the recorder to a remote ZMQ REQ socket using an interal REP socket with ZMQ style endpoint specifier: '<transport>://<host>:<port>'. For instance, 'tcp://*:5555' or 'ipc://*:5556' specify TCP and interprocess communication on ports 5555 or 5556, respectively.

Example

```
# Save positional stream 'pos' to current directory
oat record -p pos

# Save positional stream 'pos1' and 'pos2' to current directory
oat record -p pos1 pos2

# Save positional stream 'pos1' and 'pos2' to Desktop directory and
# prepend the timestamp to the file name
oat record -p pos1 pos2 -d -f ~/Desktop

# Save frame stream 'raw' to current directory
oat record -s raw

# Save frame stream 'raw' and positional stream 'pos' to Desktop
# directory and prepend the timestamp and the word 'test' to each filename
oat record -s raw -p pos -d -f ~/Desktop -n test
```

Position Socket

`oat-posisock` - Stream detected object positions to the network in either client or server configurations.

Signature

```
position --> oat-posisock
```

Usage

```
Usage: posisock [INFO]
   or: posisock TYPE SOURCE [CONFIGURATION]
Send positions from SOURCE to a remote endpoint.

INFO:
  --help                  Produce help message.
  -v [ --version ]        Print version information.

TYPE:
  std: Asynchronous position dump to stdout.
  pub: Asynchronous position publisher over ZMQ socket.
       Publishes positions without request to potentially many
       subscribers.
  rep: Synchronous position replier over ZMQ socket.
       Sends positions in response to requests from a single
       endpoint.Several transport/protocol options. The most
       useful are tcp and interprocess (ipc).
  udp: Asynchronous, client-side, unicast user datagram protocol
       over a traditional BSD-style socket.

SOURCE:
  User-supplied name of the memory segment to receive positions from (e.g. pos).
```

Example

```
# Reply to requests for positions from the 'pos' stream to port 5555 using TCP
oat posisock rep pos tcp://*:5555

# Asychronously publish positions from the 'pos' stream to port 5556 using TCP
oat posisock pub pos tcp://*:5556

# Dump positions from the 'pos' stream to stdout
oat posisock std pos
```

Buffer

`oat-buffer` - A first in, first out (FIFO) token buffer that can be use to decouple asynchronous portions of a data processing network. An example of this is the case when a precise external clock is used to govern image acquisition via a physical trigger line. In this case, 'hickups' in the data processing network following the camera should not cause the camera to skip frames. Of course, there is no free lunch: if the processing pipline cannot keep up with the external clock on average, then the buffer will eventually fill and overflow.

Signatures

```
position --> oat-buffer --> position

frame --> oat-buffer --> frame
```

Usage

```
Usage: buffer [INFO]
   or: buffer TYPE SOURCE SINK [CONFIGURATION]
Place tokens from SOURCE into a FIFO. Publish tokens in FIFO to SINK.

INFO:
  --help                  Produce help message.
  -v [ --version ]        Print version information.

TYPE
  frame: Frame buffer
  pos2D: 2D Position buffer

SOURCE:
  User-supplied name of the memory segment to receive tokens from (e.g. input).

SINK:
  User-supplied name of the memory segment to publish tokens to (e.g. output).
```

Example

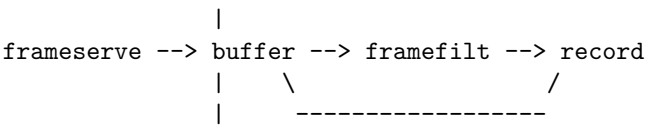
```
# Acquire frames on a gige camera driven by an exnternal trigger
oat frameserve gige raw -c config.toml gige-trig

# Buffer the frames separate asychronous sections of the processing network
oat buffer frame raw buff

# Filter the buffered frames and save
```

```
oat framefilt mog buff filt
oat record -f ~/Desktop/ -p buff filt
```

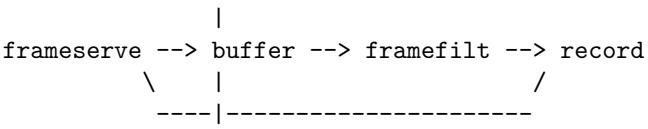
In the above example, one must be careful to fully separate the network across the buffer boundary in order for it to provide any functionality. For instance, if we changed the record command to the following

```
oat record -f ~/Desktop/ -p raw filt
```

Then the buffer would do nothing since the raw token stream must be synchronous with the recorder, which bypasses the buffer. In this case, the buffer is just wasting CPU cycles. Here is a graphical representation of the first configuration where the `oat-buffer` is used properly. The synchronization boundary is shown using vertical lines.



In the second configuration, the connection from frameserve to record breaks the synchronization boundary.



Calibrate

`oat-calibrate` - Interactive program used to generate calibration parameters for an imaging system that can be used to parameterize `oat-framefilt` and `oat-posifilt`. Detailed usage instructions are displayed upon program startup.

Signature

```
frame --> oat-calibrate
```

Usage

```
Usage: calibrate [INFO]
   or: calibrate TYPE SOURCE SINK [CONFIGURATION]
Camera calibration and homography generation routines.

INFO:
  --help                Produce help message.
  -v [ --version ]      Print version information.

TYPE
  camera: Generate calibration parameters (camera matrix and distortion coefficients).
  homography: Generate homography transform between pixels and world units.

SOURCE:
  User-supplied name of the memory segment to receive frames from (e.g. raw).
```

Kill

`oat-kill` - Issue SIGINT to all running Oat processes started by the calling user. A side effect of Oat's architecture is that components can become orphaned in certain circumstances: abnormal termination of attached sources or sinks, running pure sources in the background and forgetting about them, etc. This utility will gracefully interrupt all currently running oat components.

Usage

```
Usage: kill
```

Example

```
# Interupt all currently running oat components
oat kill
```

Clean

`oat-clean` - Programmer's utility for cleaning shared memory segments after following abnormal component termination. Not required unless a program terminates without cleaning up shared memory. If you are using this for things other than development, then please submit a bug report.

Usage

```
Usage: clean [INFO]
   or: clean NAMES [CONFIGURATION]
Deallocate the named shared memory segments specified by NAMES.

INFO:
  --help                Produce help message.
  -v [ --version ]      Print version information.
  -q [ --quiet ]        Quiet mode. Prevent output text.
  -l [ --legacy ]       Legacy mode. Append  "_sh_mem" to input NAMES before
                        removing.
```

Example

```
# Remove raw and filt blocks from shared memory after abnormal terminatiot of
# some components that created them
oat clean raw filt
```

Installation

First, ensure that you have installed all dependencies required for the components and build configuration you are interested in in using. For more information on dependencies, see the dependencies section below. To compile and install Oat, starting in the top project directory, create a build directory, navigate to it, and run cmake on the top-level CMakeLists.txt like so:

```
mkdir release
cd release
cmake -DCMAKE_BUILD_TYPE=Release [CMAKE OPTIONS] ..
make
make install
```

If you just want to build a single component component, individual components can be built using `make [component-name]`, e.g. `make oat-view`. Available cmake options and their default values are:

```
-DUSE_FLYCAP=Off // Compile with support for Point Grey Cameras
-DBUILD_DOCS=Off     // Generate Doxygen documentation
```

If you had to install Boost from source, you must let cmake know where it is installed via the following switch. Obviously, provide the correct path to the installation on your system.

```
-DBOOST_ROOT=/opt/boost_1_59_0
```

To complete installation, add the following to your `.bashrc` or equivalent. This makes Oat commands available within your user profile (once you start a new terminal):

```
# Make Oat commands available to user
eval "$(<path/to/Oat>/oat/bin/oat init -)"
```

If you get runtime link errors when you try to run an Oat program such as >error while loading shared libraries: libboost_program_options.so.1.60.0 then you need to ad the following entry to your `.bashrc`

```
export LD_LIBRARY_PATH=</path/to/boost>/stage/lib:$LD_LIBRARY_PATH
```

Dependencies

License compatibility

Oat is licensed under the GPLv3.0. Its dependences' are licenses are shown below:

- Flycapture SDK: NON-FREE specialized license (This is an optional package. If you compile without Flycapture support, you can get around

this. Also, see the `GigE interface cleanup` entry in the TODO section for a potentially free alternative.)
- OpenCV: BSD
- ZeroMQ: LGPLv3.0
- Boost: Boost software license
- cpptoml: Some kind of Public Domain Dedication
- RapidJSON: BSD
- Catch: Boost software license

These licenses do not violate the terms of Oat's license. If you feel otherwise please submit an bug report.

Flycapture SDK

The FlyCapture SDK is used to communicate with Point Grey digital cameras. It is not required to compile any Oat components. However, the Flycapture SDK is required if a Point Grey camera is to be to be used with the `oat-frameserve` component to acquire images. If you simply want to process pre-recorded files or use a web cam, e.g. via

```
oat-frameserve file raw -f video.mpg
oat-frameserve wcam raw
```

then this library is *not* required.

To install the Point Grey SDK:

- Go to point-grey website
- Download the FlyCapture2 SDK (version >= 2.7.3). Annoyingly, this requires you to create an account with Point Grey.
- Extract the archive and use the `install_flycapture.sh` script to install the SDK on your computer and run

```
tar xf flycapture.tar.gz
cd flycapture
sudo ./install_flycapture
```

Boost

The Boost libraries are required to compile all Oat components. You will need to install versions >= 1.56. To install Boost, use APT or equivalent,

```
sudo apt-get install libboost-all-dev
```

If you are using an Ubuntu distribution older than Wily Werewolf, Boost will be too old and you will need to install from source via

```
# Install latest boost
wget http://sourceforge.net/projects/boost/files/latest/download?source=files -O tarboost
tar -xf tarboost
```

```
cd ./boost*
./bootstrap.sh
./b2 --with-program_options --with-system --with-thread --with-filesystem
cd ..
sudo mv boost* /opt
```

Finally, if you are getting runtime linking errors, you will need to place the following in `.bashrc`

```
export LD_LIBRARY_PATH=<path to boost root directory>/stage/lib:$LD_LIBRARY_PATH
```

OpenCV

opencv is required to compile the following oat components:

- `oat-frameserve`
- `oat-framefilt`
- `oat-view`
- `oat-record`
- `oat-posidet`
- `oat-posifilt`
- `oat-decorate`
- `oat-positest`

Note: OpenCV must be installed with ffmpeg support in order for offline analysis of pre-recorded videos to occur at arbitrary frame rates. If it is not, gstreamer will be used to serve from video files at the rate the files were recorded. No cmake flags are required to configure the build to use ffmpeg. OpenCV will be built with ffmpeg support if something like

```
-- FFMPEG:          YES
-- codec:           YES (ver 54.35.0)
-- format:          YES (ver 54.20.4)
-- util:            YES (ver 52.3.0)
-- swscale:         YES (ver 2.1.1)
```

appears in the cmake output text. The dependencies required to compile OpenCV with ffmpeg support, can be obtained as follows:

```
TODO
```

Note: To increase Oat's video visualization performance using `oat view`, you can build OpenCV with OpenGL and/or OpenCL support. Both will open up significant processing bandwidth to other Oat components and make for faster processing pipelines. To compile OpenCV with OpenGL and OpenCL support, first install dependencies:

```
sudo apt-get install libgtkglext1 libgtkglext1-dev
```

Then, add the `-DWITH_OPENGL=ON` and the `-DWITH_OPENCL=ON` flags to the cmake command below. OpenCV will be build with OpenGL and OpenCL support if `OpenGL support: YES` and `Use OpenCL: YES` appear in the cmake output text. If OpenCV is compiled with OpenCL and OpenGL support, the performance benefits will be automatic, no compiler options need to be set for Oat.

Note: If you have NVIDIA GPU that supports CUDA, you can build OpenCV with CUDA support to enable GPU accelerated video processing. To do this, will first need to install the CUDA toolkit.

- Be sure to **carefully** read the installation instructions since it is a multistep process. Here are some additional hints that worked for me:
- I have found that installing the toolkit via 'runfile' to be the most painless. To do this you will need to switch your system to text mode using Ctrl + Alt + F1, and killing the X-server via `sudo service lightdm stop` (or equivalent), and running the runfile with root privileges.
- I have had the most success on systems that do not use GNOME or other fancy desktop environments. The install on lubunut, which uses LXDE as its desktop environment, was especially smooth.
- Do **not** install the nvidia drivers along with the CUDA toolkit installation. I found that (using ubuntu 14.04) this causes all sorts of issues with X, cinnamon, etc, to the point where I could not even boot my computer into anything but text mode. Instead, install the NVIDIA drivers using either the package manager (`nvidia-current`) or even more preferably, using the [`device-drivers`]'(http://askubuntu.com/a/476659) program or equivalent.
- If you hare getting a `cv::exception` complaining that about `code=30(cudaErrorUnknown) "cudaGetDeviceCount(&device_count)"` or similar, run the affected command as root one time.

If OpenCV is compiled with CUDA suport, the CUDA-enabled portions of the Oat codebase will be enabled automatically. No compile flags are required.

Note: GUI functionality is enhanced in OpenCV is compiled with Qt support. You can build OpenCV with Qt by first installing the Qt SDK and these dependencies:

```
# Additional dependencies for integraged QT with OpenGL
sudo apt-get install libqt5opengl5 libqt5opengl5-dev
```

The you can compile OpenCV using QT support by adding `-DWITH_QT=ON` flag to the cmake command below. QT functionality will then be used by Oat automatically.

Finally, to compile and install OpenCV:

```
# Install OpenCV's dependencies
sudo apt-get install build-essential # Compiler
sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libsw
sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev libpng-dev libti
```

```
sudo apt-get install # ffmpeg support [TODO]
sudo apt-get install # OpenGL support [TODO]
sudo ldconfig -v

# Get OpenCV
wget https://github.com/Itseez/opencv/archive/3.1.0.zip -O opencv.zip
unzip opencv.zip -d opencv

# Build OpenCV
cd opencv/opencv-3.0.0-rc1
mkdir release
cd release

# Run cmake to generate Makefile
# Add -DWITH_CUDA=ON for CUDA support and -DWITH_OPENGL for OpenGL support
cmake -DCMAKE_BUILD_TYPE=RELEASE -DCMAKE_INSTALL_PREFIX=/usr/local ..

# Build the project and install
make
sudo make install
```

ZeroMQ

ZeroMQ is required by the following Oat components:

- `oat-record`
- `oat-posisock`

Download, compile, and install ZeroMQ as follows:

```
wget http://download.zeromq.org/zeromq-4.1.4.tar.gz -O tarzmq
tar -xf tarzmq
cd ./zeromq*
./configure --without-libsodium
make
sudo make install
sudo ldconfig
```

Additionally, you will need to download the ZeroMQ C++ binding (this is just a single header file) and place it somewhere that your compiler will find it.

```
wget https://raw.githubusercontent.com/zeromq/cppzmq/master/zmq.hpp
sudo mv zmq.hpp /usr/local/include/
```

RapidJSON, cpptoml, and Catch

These libraries are installed automatically by cmake during the build process.

RapidJSON is required by the following Oat components:

- `oat-record`
- `oat-posisock`

cpptoml is required by the following Oat components:

- `oat-frameserve`
- `oat-framefilt`
- `oat-posidet`
- `oat-posifilt`
- `oat-posicom`
- `oat-positest`

Catch is required to make and run tests using `make test`

Performance

Oat is designed for use in real-time video processing scenarios. This boils down the following definition

> The average execution time for an Oat dataflow network must not exceed the camera(s) image transfer period

If this condition is not met, then frames will eventually be dropped. There is no way around this. The guts of Oat consist of a simple, but very efficient message passing library that links together processing routines taken from a variety of sources (some written by me, some by third party projects such as OpenCV). The speed of each processing step is determined both by its computational complexity and deftness of implementation, both of which can vary quite a lot for different components. To see some rudimentary performance numbers for Oat components in isolation, have a look at these numbers. There is definitely room for optimization for some components. And, several components that are ripe for GPU implementation do not have one yet. This comes down to free time. If anyone wants to try there hand at making some of the bottleneck components faster, please get in touch.

Outside of code optimization, there are a few things a user should be aware of to make efficient use of Oat, which are listed below.

Frames are slow

The first thing to know is that working with `frames` is orders of magnitude slower than working with `positions`. Therefore, minimizing the number of processing steps operating on `frames` is a good way to reduce computational requirements. Processing on `positions` is in the noise in comparison.

Parallelism

Increasing the number of components in your chain does not necessarily cause an appreciable an increase in processing time because Oat components run in parallel. Instead, up to the limit of the number of hyperthreads/GPU resources your computer supports, the slowest component in a dataflow network will largely determine the speed of the processing rather than the number of components within the processing network.

Resolution

Do you really need that 10 MP camera? Recall that increases in sensor resolution cause a power 2 increase in then number of pixels you need to smash into RAM, process, write to disk, and, probably, post process. Its really best to use the lowest resolution camera that suites your needs, both for the sake of real-time processing in Oat and your future sanity when trying to deal with those 30 GB video files.

Hard-disk

If you are saving video, then the write speed of your hard disk can become the limiting factor in a processing network. To elaborate, I'm just quoting my response to this issue:

> **Q:** I also ran into an issue with RAM and encoding. I have 8 GB, and they fill up within about 20 seconds, then goes into swap.
>
> **A:** I suspect the following is the issue:
>
> (22 FPS * 5 MP * 24 bits/pixel) / (8 bits/ byte) = 330 MB/sec
>
> This (minus compression, which I'm admittedly ignoring, but is probably made up for by the time it takes to do the compression. . .) is the requisite write speed (in actuality, not theoretically) of your hard disk in order not to get memory overflow.
>
> 8 GB / 0.330 GB =~ 24 seconds.
>
> The RAM is filling because your hard disk writes are not occurring fast enough. Oat is pushing frames to be written into a FIFO in main memory that the recorder threads are desperately trying to write to disk. Getting more RAM will just make the process persist for a bit longer before failing . I would get an SSD for streaming video to and then transfer those videos to a slower long term storage after recording.

Setting up a Point-grey PGE camera in Linux

`oat-frameserve` supports using Point Grey GIGE cameras to collect frames. I found the setup process to be straightforward and robust, but only after cobbling together the following notes.

Camera IP Address Configuration

First, assign your camera a static IP address. The easiest way to do this is to use a Windows machine to run the Point Grey 'GigE Configurator'. If someone knows a way to do this without Windows, please tell me. An example IP Configuration might be:

- Camera IP: 192.168.0.1
- Subnet mask: 255.255.255.0
- Default gateway: 192.168.0.64

Point Grey GigE Host Adapter Card Configuration

Using network manager or something similar, you must configure the IPv4 configuration of the GigE host adapter card you are using to interface the camera with your computer.

- First, set the ipv4 method to **manual**.
- Next, you must configure the interface to (1) have the same network prefix and (2) be on the same subnet as the camera you setup in the previous section.
  - Assuming you used the camera IP configuration specified above, your host adapter card should be assigned the following private IPv4 configuration:
    * POE gigabit card IP: 192.168.0.100
    * Subnet mask: 255.255.255.0
    * DNS server IP: 192.168.0.1
- Next, you must enable jumbo frames on the network interface. Assuming that the camera is using `eth2`, then entering

  ```
  sudo ifconfig eth2 mtu 9000
  ```

into the terminal will enable 9000 MB frames for the `eth2` adapter. - Finally, to prevent image tearing, you should increase the amount of memory Linux uses for network receive buffers using the `sysctl` interface by typing

```
sudo sysctl -w net.core.rmem_max=1048576 net.core.rmem_default=1048576
```

into the terminal. *In order for these changes to persist after system reboots, the following lines must be added to the bottom of the `/etc/sysctl.conf` file*:

```
net.core.rmem_max=1048576
net.core.rmem_default=1048576
```

These settings can then be reloaded after reboot using

```
sudo sysctl -p
```

Multiple Cameras

- If you have two or more cameras/host adapter cards, they can be configured as above but *must exist on a separate subnets*. For instance, we could repeat the above configuration steps for a second camera/host adapter card using the following settings:
  - Camera Configuration:
    - Camera IP: 192.168.**1**.1
    - Subnet mask: 255.255.255.0
    - Default gateway: 192.168.**1**.64
  - Host adapter configuration:
    - POE gigabit card IP: 192.168.**1**.100
    - Subnet mask: 255.255.255.0
    - DNS server IP: 192.168.**1**.1

Example Camera Configuration

Below is an example network adapter and camera configuration for a two-camera imaging system provided by Point Grey. It consists of two Blackfly GigE cameras (Point Grey part number: BFLY-PGE-09S2C) and a single dual-port POE GigE adapter card (Point Grey part number: GIGE-PCIE2-2P02).

Camera 0

- Adapter physical connection (looking at back of computer)

```
RJ45 ------------
       |      |
L [  [ ]    [x]  ] R
```

- Adapter Settings
  - Model: Intel 82574L Gigabit Network Connection
  - MAC: 00:B0:9D:DB:D9:63
  - MTU: 9000
  - DHCP: Disabled
  - IP: 192.168.0.100
  - Subnet mask: 255.255.255.0
- Camera Settings

  - Model: Blackfly BFLY-PGE-09S2C
  - Serial No.: 14395177
  - IP: 192.168.0.1 (Static)
  - Subnet mask: 255.255.255.0
  - Default GW: 0.0.0.0
  - Persistent IP: Yes

Camera 1

- Adapter physical connection (looking at back of computer)

```
RJ45 ------------
       |       |
L [  [x]     [ ]  ] R
```

- Adapter Settings
  - Model: Intel 82574L Gigabit Network Connection
  - MAC: 00:B0:9D:DB:A7:29
  - MTU: 9000
  - DHCP: Disabled
  - IP: 192.168.1.100
  - Subnet mask: 255.255.255.0
- Camera Settings
  - Model: Blackfly BFLY-PGE-09S2C
  - Serial No.:
  - IP: 192.168.1.1 (Static)
  - Subnet mask: 255.255.255.0
  - Default GW: 0.0.0.0
  - Persistent IP: Yes

TODO

- [ ] Unit and stress testing
  - Unit tests for `libshmemdf`
    - ~~Nominal data types, `T`~~
    - Specializations for `Frames`
  - Stress tests for data processing chains
    - I need to come up with a series of scripts that configure and run components in odd and intensive, but legal, ways to ensure sample sychronization is maintained, graceful exits, etc
- [ ] GigE interface cleanup
  - ~~The PGGigeCam class is a big mess. It has has tons of code redundancy.~~
    - EDIT: A lot of this is due to the PG API. I've cleaned up a bit, but more would be a waste of time.
  - ~~`oat` frameserve gige can wait indefinitely if the cameras use an external trigger and that trigger source stops before the process is interrupted. Need a timed wait there.~~
    - EDIT: Fixed in a0c97e56bbe66227b03dd9253fdff33f0550465b
  - Should I be using the generic GenICam API instead of PG's non-standard API? e.g. Aravis.
  - ~~Additionally, it needs to be optimized for performance. Are their unnessesary copies of images being made during conversion from PG Image to cv::Mat? Can I employ some move casts to help?~~
    - EDIT: shmemdf takes care of this.
  - There are a couple examples of GigE interfaces in OpenCV targeting other 3rd party APIs: `modules/videoio/src/cap_giganetix.cpp` and `opencv/modules/videoio/src/cap_pvapi.cpp`. I don't know that these are great pieces of code, but I should at least use them for inspiration.
  - `oat-frameserve gige` lacks the ability to set FPS in free running (non-triggered mode)
  - ~~Configuration of the camera can get into impossible states if oat is used in combo with other programs that mess with the camera's registers. Configuration via Oat should start with a clean slate by setting the camera to a default register state.~~
  - ~~See flycap -> advanced camera settings -> restor default memory channel for how.~~
  - Binning should be specified in terms of mode number (see pg. 66 of blackfly tech ref) instead of bin size since not all bin sizes are allowed.
- [ ] Position type generalization
  - It might be a good idea to generalize the concept of a position to a multi-positional element
  - For things like the `oat-decorate`, `oat-posicom`, and potentially `oat-detect`, this could increase performance and decrease user script

complexity if multiple targets common detection features needed to be tracked at once.
  - Down side is that it potentially increases code complexity and would require a significant refactor.
  - Additionally, position detection might no longer be stateless. E.g. think of the case when two detected objects cross paths. In order to ID the objects correctly in subsequent detections, the path of the objects would need to be taken into account (and there is not guarantee this result will be correct...). A potential work around is to have IDed 'position groups' with annoymous position members. This would get us back to stateless detection. However, it would make the concept of position combining hard to define (although that is even true now is just a design choice, really).
- [ ] ~~Saving tuning parameters~~
  - Components that have a `--tune` option should also allow for the user to press a key and those tuning paramters to be injected into the current `config.toml` file so that they don't have to write them down and manually edit the file later
  - EDIT: Counter argument: thinking about correct implementation brings to mind GUI file dialogs or more command line switches to configure save path. This makes me want to barf a little so maybe lets put this on hold until its a big issue.
- [x] Something is wrong with sample synchronization
  - When working with Jennie's data, we found that position samples were being recorded multiple times - they had the same sample number and position info. Seems to be very intermittent, but points to a serious issue with sample synchronization. It seems likely this occurring in the recorder component due to its mulitthreaded implementation.
  - EDIT: With `libshmemdf`, sample numbers now travel with samples. I need to use this to implement `asserts` in each component that (1) check that samples increase monotonically (buffer overflows means that samples can be skipped, so uniary incrementation is not a given) and (2) that multisource components are dealing with sychonized sample numbers when pull-based sychornization strategy is enforced (no external clock driving acqusition, so no chance for buffer overrun).
- [ ] Command line switches should take precedence over TOML file options. This is standard practice, but is not how Oat works currently.
  - `oat-frameserve`
  - `oat-framefilt`
  - `oat-posifilt`
  - `oat-posigen`
  - `oat-posicom`
  - `oat-posidet`
  - NOTE: A way to do this is to create (or maybe find) and factory that generates a dictionary of possible parameters in key/value pairs for a given component. This dictionary is then modified first by the

file-based configuration method and then by command line switch input. Component behavior is determined by dictioary values after these two steps.

- [ ] ~~For (all?) most components, `configure` is pure abstract in the component's base class. This doesn't make too much sense because options are often common to many components. For instance, in `oat-posigen`, the sample period, and number of samples parameters are certainly relevant to any implementation of the position generator idea. Therefore, `configure` should be abstract with a base implementation containing guaranteed-to-be-common parameters. For components that have no common parameters, it can be left pure-abstract for the time being.~~
  - `oat-frameserve`
  - `oat-framefilt`
  - `oat-posifilt`
  - ~~`oat-posigen`~~
  - `oat-posicom`
  - `oat-posidet`
  - EDIT: There are not actually many components that have common configuration parameters among concrete types. `oat-posidet` and possibly `oat-frameserve` seem like the only candidates.
  - EDIT: The program option refactor branch may take care of this automatically
- [ ] [CBOR](http://tools.ietf.org/html/rfc7049) binary messaging and data files
- CBOR is an extremely simple binary encoding scheme for JSON
- It would be great to allow the option to save CBOR files (`oat-record`) or send CBOR messages (`oat-posisock`) by creating a CBOR `Writer` acceptable to by `Position` datatype's serialization function.
- And, while I'm at it, Position's should be forced to support serialization, so this should be a pure abstract member of the base class.
- Another option that is very similar is messagepack. Don't know which is better.
- [ ] `oat-framefilt undistort`
  - Very slow. Needs an OpenGL or CUDA implementation
  - User supplied frame rotation occurs in a separate step from undistortion. Very inefficient. Should be able to combine rotation with camera matrix to make this a lot faster.
- [ ] Should components always involve a user IO thread?
  - For instance, some generalization of `oat-record ... --interactive`
  - For instance, it would be nice if PURE SINKs (e.g. `oat frameserve`) could have their sample clock reset via user input, without having to restart the program.
  - For instance, it would be nice to be able to re-acquire the background image in `oat-framefilt bsub` without have to restart the program.
  - Where should this come from? Command line input?

- [ ] Add past position line toggle in `oat-decorate`