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

Oat components are a set of programs that communicate through shared memory to capture, process, perform object detection within, and record video streams. Oat components act on two basic data types: `frames` and `positions`.

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

Oat components are be chained together to realize custom data processing pipelines, with individual components executing largely in parallel. Processing pipelines can be split and merged while maintaining thread-safety and sample synchronization. For example, a script to detect the position of a single object in pre-recorded video file might look like this:

```
# 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
oat framefilt bsub raw filt &

# Perform color-based object position detection on the 'filt' stream
# Serve the object positionto the 'pos' stream
oat posidet hsv filt pos &

# 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:

```
frameserve --> framefilt --> posidet --> decorate ---> view
           \                           /         \
             -------------------------            ---> record
```

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-par]
frame_rate = 30                 # FPS
roi = [10, 10, 100, 100]        # Region of interest

[framefilt-par]
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-frameview --> frame
```

Usage

```
Usage: frameserve [INFO]
   or: frameserve TYPE SINK [CONFIGURATION]

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

TYPE
  wcam: Onboard or USB webcam.
  gige: Point Grey GigE camera.
  file: Stream video from file.

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

CONFIGURATION:
  -c [ --config-file ] arg  Configuration file.
  -k [ --config-key ] arg   Configuration key.
  -f [ --video-file ] arg   Path to video file if 'file' is selected as the
                            server TYPE.
```

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`. 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.
- **triger_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.

TYPE = file

- **frame_rate**=float Frame rate in frames per second
- **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 -k 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 -k file_config
```

Frame Filter

`oat-framefilt` - Receive frames from named shared memory, 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 published filtered frames to SINK.

TYPE
  bsub: Background subtraction
  mask: Binary mask
   mog: Mixture of Gaussians background segmentation (Zivkovic, 2004)

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).

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

CONFIGURATION:
  -c [ --config-file ] arg  Configuration file.
  -k [ --config-key ] arg   Configuration key.
  -m [ --invert-mask ]      If using TYPE=mask, invert the mask before applying
```

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.

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.

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 -k 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 SOURCE [CONFIGURATION]
Display frame SOURCE on a monitor.

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

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

CONFIGURATION:
  -n [ --filename ] arg  The base snapshot file name.
                         The timestamp of the snapshot will be prepended to
                         this name.If not provided, the SOURCE name will be
                         used.

  -f [ --folder ] arg    The folder in which snapshots will be saved.
```

Example

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

# View frame stream named raw and specify that snapshots should be saved
# to the Desktop with base name 'snapshot'
oat view 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 position detection on frames from SOURCE.
Publish detected object positions to SINK.

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
  detected positions to (e.g. pos).

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

CONFIGURATION:
  -c [ --config-file ] arg  Configuration file.
  -k [ --config-key ] arg   Configuration key.
```

Configuration File Options

TYPE = `hsv`

- **`tune`**=`bool` Provide 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 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 -k 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 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 published filtered positions to SINK.

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

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

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

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

CONFIGURATION:
  -c [ --config-file ] arg  Configuration file.
  -k [ --config-key ] arg   Configuration key.
```

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 = `homo`

- `homography` = `[[+float,+float,+float],[+float,+float,+float],[+float,+float,+float]]`, Homography matrix for 2D position (3x3; world units/pixel).

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 -k 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.
Publish combined position to SINK.

TYPE
  mean: Geometric mean of SOURCE positions

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

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

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

CONFIGURATION:
  -c [ --config-file ] arg  Configuration file.
  -k [ --config-key ] arg   Configuration key.
```

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. Publish decorated frames to SINK.

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).

OPTIONS:

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

CONFIGURATION:
  -p [ --position-sources ] arg  The name of position server(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.

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]} }
{samples: [ [0, {Position1}, {Position2}, ..., {PositionN} ],
            [1, {Position1}, {Position2}, ..., {PositionN} ],
             :
            [T, {Position1}, {Position2}, ..., {PositionN} ]] }
```

where each position object is defined as:

```
{ ID: String,                 | A string matching an ID from the header sources
  unit: String,               | 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 tagk
```

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:

```
{ ID: "Example",
  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 [OPTIONS]
   or: record [CONFIGURATION]

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

CONFIGURATION:
  -n [ --filename ] arg         The base file name to which to source name will
                                be appended
  -f [ --folder ] arg           The path to the folder to which the video
                                stream and position information will be saved.
  -d [ --date ]                 If specified, YYYY-MM-DD-hh-mm-ss_ will be
                                prepended to the filename.
  -p [ --positionsources ] arg  The name of the server(s) that supply object
                                position information.The server(s) must be of
                                type SMServer<Position>

  -i [ --imagesources ] arg     The name of the server(s) that supplies images
                                to save to video.The server must be of type
                                SMServer<SharedCVMatHeader>
```

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 -i raw

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

Position Network Socket

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

Signature

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

Usage

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

TYPE
  udp: User datagram protocol.

OPTIONS:

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

CONFIGURATION:
  -h [ --host ] arg         Remote host to send positions to.
  -p [ --port ] arg         Port on which to send positions.
  --server                  Server-side socket sychronization. Position data
                            packets are sent whenever requestedby a remote
                            client. TODO: explain request protocol...
  -c [ --config-file ] arg  Configuration file.
  -k [ --config-key ] arg   Configuration key.
```

Example

TODO

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:

```
OAT_USE_FLYCAP=Off // Compile with support for Point Grey Cameras
OAT_USE_OPENGL=Off // Compile with support for OpenGL rendering
OAT_USE_CUDA=Off   // Compile with NVIDIA GPU accerated processing
```

See the Dependencies sections to make sure you have the required dependencies to support each of these options if you plan to set them to `On`.

To complete installation, add the following to your `.bashrc`. This makes Oat command available within your user profile:

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

Dependencies

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 We have had success with Boost versions >= 1.54. To install Boost, use your package manager,

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

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. To compile OpenCV with ffmpeg support, use:

```
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, 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 support, the performance benefits will be automatic. For OpenGL, you need to set a flag when building Oat to see the performance gains. See the Installation section for details.

Note: If you have NVIVIA 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 read the installation instructions since it is a multistep process.To compile OpenCV with CUDA support, add the `-DWITH_CUDA=ON` flag in the cmake command below.

To 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/3.0.0-rc1.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
```

RapidJSON and cpptoml

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`

To install both of these libraries, starting in the project root directory, run:

```
cd lib
./update_libs.sh
```

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 Greg 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
```

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

TODO

- [ ] Networked communication with remote endpoints that use extracted positional information
  - ~~Strongly prefer to consume JSON over something ad hoc, opaque and untyped~~
  - Multiple clients
    * UDP with single endpoint
      · ~~Client version (sends data without request)~~
      · Server version (pipeline is heldup by client position requests)
    * TCP/IP with single endpoint
      · Client version (sends data without request)
      · Server version (pipeline is heldup by client position requests)
    * Broadcast over UDP?
- [ ] Cmake/build improvements
  - ~~Global build script to make all of the programs in the project~~
  - ~~CMake managed versioning~~
  - ~~Option for building with/without point-grey support~~
  - ~~Author/project information injection into source files using either cmake or doxygen~~
  - Put boost in a more standard location
  - Clang build
  - Windows build?
- [ ] Travis CI
  - Get it building using the improvements to CMake stated in last TODO item
- [ ] Frame and position server sample synchronization
  - [ ] Dealing with dropped frames
    * Right now, I poll the camera for frames. This is fine for a file, but not necessarily for a physical camera whose acquisitions is governed by an external, asynchronous clock
    * Instead of polling, I need an event driven frame server. In the case of a dropped frame, the server **must** increment the sample number, even if it does not serve the frame, to prevent offsets from occurring.
  - [ ] Pull-based synchronization with remote client.
    * By virtue of the sychronization architecture I'm using to coordinate samples between SOURCE and SINK components, I get both push and pull based processing chain updates for free. This means if I use `oat-posisock` in server mode, and it blocks until a remote client requests a position, then the data processing chain update

will be sychronized to these remote requestions. The exception are `pure SOURCE` components, which have internal ring-buffers (`oat-frameserve` and `oat-positest`). These components will fill their internal buffers at a rate driven by their internal clock regardless of remote requests for data processing chain updates. If remote sychronization is required, this is undesirable.

    * Ideally, the sychronization and buffering architecture of `pure SOURCE` components should be dependent on some globally enfornced sychronization strategy. This might be accomplished via the creation of a master, read only clock, existing in shmem, that dictates sychronization strategy and keeps time. Instead of each `pure SOURCE` keeping its own, asychronous clock (e.g. if two cameras are started at different times, they can have different absolute sample numbers at the same real-world time), this centrial clock would be in charge. This would obviate all issues I'm having with sample number propogation through multi-SOURCE components.

  – In anycase, the `pure SOURCE` components, esspecially roat-frameserve', are by far the most primative components in the project at this point and need refactoring to deal with these two issues in a reasonable way.

- [ ] shmem type checking by clients, exit gracefully in the case of incorrect type
- e.g. a framefilter tries to use a position filter as a SOURCE. In this case, the framefilter needs to realize that the SOURCE type is wrong and exit.
- [ ] GigE interface cleanup
  – The PGGigeCam class is a big mess. It has has tons of code redundancy.
  – 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?
  – 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.
- [ ] Exception safety for all components and libs
  – frameserve
  – ~~framefilt~~
  – ~~posidet~~
  – ~~posicom~~
  – ~~posifilt~~
  – ~~positest~~
  – ~~record~~

  - ~~view~~
  - ~~decorate~~
  - There are a bunch of unsafe exit conditions in the shmem library, especially having to do with EXITs after boost interprocess exceptions during shmem segment setup.

- [ ] Use smart pointers to ensure proper resource management.
  - ~~Start with `recorder`, which has huge amounts of heap allocated objects pointed to with raw pointers~~

- [ ] Decorator is primitive
  - Size of position markers, sample numbers, etc do not change with image resolution
  - How are multi-region tags displayed using the -R option?

- [ ] When a position sink decrements the client reference count, positest deadlocks instead of continuing to serve. Problem with SMServer?
- [ ] `oat-calibrate` is a complete hack. It would be best to move this component out of oat-frameserve source directory and make it its own utility component that can listen to a frame-stream from oat-frameserve
- [ ] For config file specification, there should be a single command line option with two values (file, key) instead of a single option for each.