Tyelab
Currently Existing Bruker-2P Program Execution + Some Ideas 2022

Technical Docs can be found here:

https://bruker-control.readthedocs.io/en/latest/about/index.html

Repository can be found here:

https://github.com/Tyelab/bruker_control

Solid boxes mean implemented.

Dashed boxes mean yet to be implemented or awaiting improvement

User Guide can be found here:

https://bruker-control.readthedocs.io/en/latest/instructions/index.html

Local Directory Structure

Local drive is a 1.8TB SSD dedicated to collecting raw data

```
E:
└── teamname_projectname
    ├── config
    │   └── config.json
    ├── microscopy
    │   └── recording session
    │       ├── .ENV for microscope settings
    │       ├── .XML for image metadata
    │       ├── .RAW binaries for images
    │       └── .RAW binaries for behavior
    ├── video
    │   └── .mp4 via H264 encoding
    ├── yoked
    │   └── control
    │       └ .json of trial_arrays
    │   └── experimental
    │       └ .json of trial_arrays
    └──zstacks
        └── recording_session_zstack#
            ├ .ENV for microscope settings
            ├ .XML for image metadata
            └ .RAW binaries for images
```

Server Directory Structure

Note: This is what's Austin's project has currently

```
/snlkt/
    └── teamname_projectname
        ├── subjects
        │   └── SUBJECT_ID
        │       ├── subject_id.json
        │
        ├── 2p
        │   ├── config
        │   ├── processed
        │   └── raw
        │       └── SUBJECT_ID
        │           └── DATE
        │               ├── config.json
        │               ├── microscopy
        │               ├── video.mp4
        │               └── zstacks
```

Would be best if user didn't have to upload the sketch themselves, instead did it via Arduino CLI subprocess. See Issue #71

https://github.com/Tyelab/bruker_control/issues/71

Subject folder is where subject metadata is held.

See:
https://bruker-control.readthedocs.io/en/latest/configurations/configs.html#example-subject-metadata

Additional metadata for imaging plane parameters could look something simple like this in .json, .yml, etc (includes multiple lasers as we have them at NLW and new Bruker will have it as well). Info could be found by parsing out numerous .ENV and .XML files from Bruker, but accessing them this way would be simpler to use as well as easier to read/maintain instead of having to search through Bruker's XML.

```
{
  "plane_1":

    {
      location: -300.00, # Z-axis position,
      pockels_1: 500, # First laser power, used for one calcium indicator
      pockels_2: 600, # Second laser power, used for second calcium indicator
      pmts_1: 750, # Channel 1 PMTs, used to record from first laser
      pmts_2: 650, # Channel 2 PMTs, used to record from second laser
    },

  "plane_2":

    {
      .
      .
      .
    },

  "plane_n"
      .
      .
      .
}
```

Adopting this kind of metadata file would allow the program to know how many planes it needs to take, which laser/PMT settings to uses on an animal by animal basis or even plane by plane basis if necessary. Could also automatically use microscope's API to move down to different planes. Something like this could be written easily:

```
next_plane_position = imaging_plane_metadata[n]

pl.SendScriptCommands("-SetMotorPosition '-Z' '{}'").format(next_plane_position)
```

Config folder is where behavior configurations are held.

See:
https://bruker-control.readthedocs.io/en/latest/configurations/configs.html#example-project-config-behavior

Potential Re-Design in Class-Based Code Refactor
https://github.com/Tyelab/bruker_control/blob/class_in_session/configs/project_config.json

See Pydantic datamodels validation for Class-Based Refactor here:

https://github.com/Tyelab/bruker_control/blob/class_in_session/main/datamodels.py

Naming Conventions, File Formats, Etc

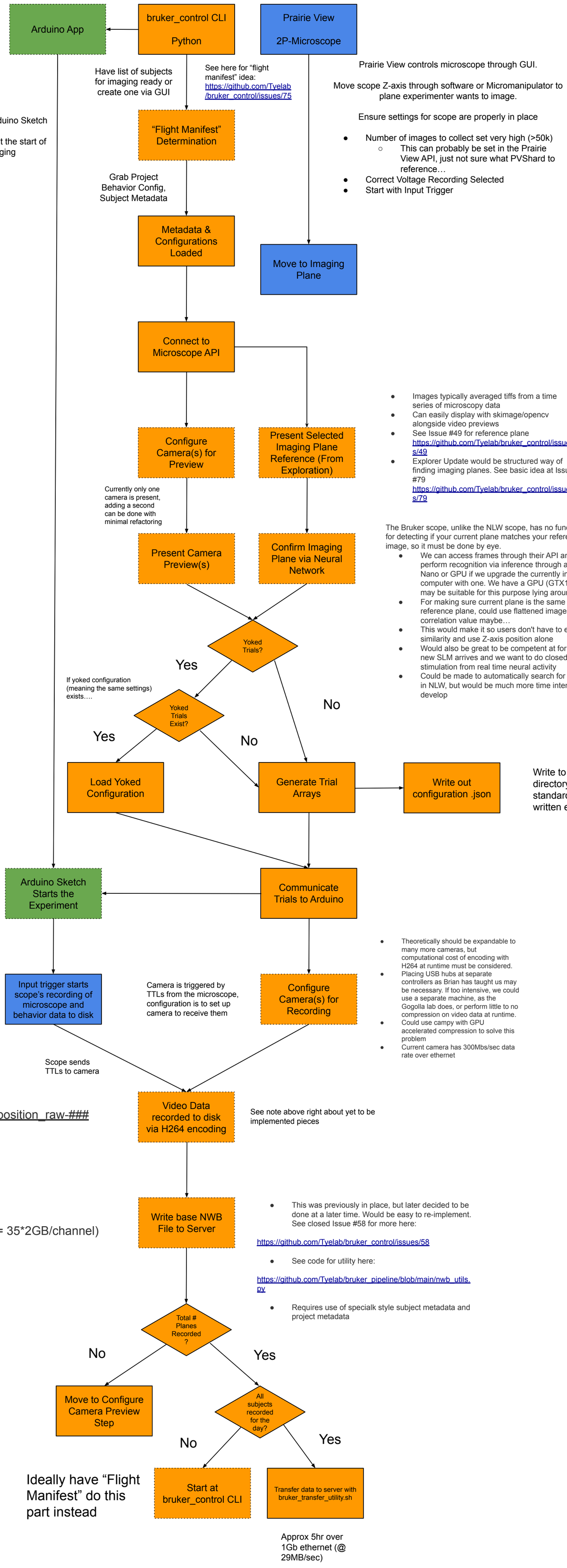
Much of this documentation resides on Confluence at the moment, but it should be migrated to RTD. The essence is:

YYYYMMDD_subjectid_plane#_planeposition

Recording types, directories, etc have their docs on Confluence and will be migrated to RTD or here…

Start of Day's Imaging Requires:

1. Load Arduino Sketch
2. Open Prairie View
3. Use bruker_control CLI



Upload team's Arduino Sketch

Only done once at the start of day's imaging

Prairie View controls microscope through GUI.

Move scope Z-axis through software or Micromanipulator to plane experimenter wants to image.

Ensure settings for scope are properly in place

- Number of images to collect set very high (>50k)
  - This can probably be set in the Prairie View API, just not sure what PVShard to reference…
- Correct Voltage Recording Selected
- Start with Input Trigger

- Images typically averaged tiffs from a time series of microscopy data
- Can easily display with skimage/opencv alongside video previews
- See Issue #49 for reference plane https://github.com/Tyelab/bruker_control/issues/49
- Explorer Update would be structured way of finding imaging planes. See basic idea at Issue #79 https://github.com/Tyelab/bruker_control/issues/79

The Bruker scope, unlike the NLW scope, has no functionality for detecting if your current plane matches your reference image, so it must be done by eye.

- We can access frames through their API and perform recognition via inference through a Jetson Nano or GPU if we upgrade the currently in place computer with one. We have a GPU (GTX1050) that may be suitable for this purpose lying around in lab.
- For making sure current plane is the same as reference plane, could use flattened images and do correlation value maybe…
- This would make it so users don't have to eyeball similarity and use Z-axis position alone
- Would also be great to be competent at for when the new SLM arrives and we want to do closed loop stimulation from real time neural activity
- Could be made to automatically search for plane, as in NLW, but would be much more time intensive to develop

- Theoretically should be expandable to many more cameras, but computational cost of encoding with H264 at runtime must be considered.
- Placing USB hubs at separate controllers as Brian has taught us may be necessary. If too intensive, we could use a separate machine, as the Gogolla lab does, or perform little to no compression on video data at runtime.
- Could use campy with GPU accelerated compression to solve this problem
- Current camera has 300Mbs/sec data rate over ethernet

- This was previously in place, but later decided to be done at a later time. Would be easy to re-implement. See closed Issue #58 for more here:

https://github.com/Tyelab/bruker_control/issues/58

- See code for utility here:

https://github.com/Tyelab/bruker_pipeline/blob/main/nwb_utils.py

- Requires use of specialk style subject metadata and project metadata

Time Series Directory
yyyymmdd_subjectid_plane#_planeposition_raw-###

VoltageRecording.xml
VoltageRecordingRAW (<2GB)
VoltageRecordingFilelist.txt
MicroscopyMetadata.env
MicroscopyRecording.xml
MicroscopyRecordingFilelist.txt
MicroscopyRecordingRAW (30 min = 35*2GB/channel)