Stytra Documentation

Release 0.8

Vilim Štih and Luigi Petrucco, @portugueslab

Mar 18, 2019

Contents:

Stytra is a package to build and run behavioral experiments.

CHAPTER 1

Installation guide

Stytra was developed and tested using Python 3.7 installed as part of the Anaconda Python distribution. Other Python versions have not been tested. Make sure you have the latest version of Anaconda installed before proceeding with the installation. Installation with custom python environments, Miniconda, or in Anaconda virtual environments could be achieved but might give dependencies issues. The following instructions have been tested and work for an installation in the Anaconda root environment.

1.1 Installing stytra

Stytra relies on opencv for some of its fish tracking functions. If you don't have it installed, open the Anaconda prompt and type:

```
conda install opencv
```

If you are using Windows, git (used for tracking software versions) might not be installed. Git can also be easily installed with conda:

```
conda install git
```

This should be everything you need to make ready before installing stytra.

Note: PyQt5 is not listed as an explicit requirement because it should come with the Anaconda package. If you are not using Anaconda, make sure you have it installed and updated before installing Stytra!

The simplest way to install Stytra is with pip:

```
pip install stytra
```

You can verify the installation by running one of the examples in stytra examples folder. To run a simple looming stimulus experiment, you can type:

```
python -m stytra.examples.looming_exp
```

If the GUI opens correctly and pressing the play button starts the stimulus: congratulations, installation was successful! If it crashes, check if you have all dependencies correctly installed. If it still does not work, open an issue on the Stytra github page.

1.1.1 Editable installation

On the other hand, if you want to modify the internals of stytra or use the unreleased features, clone or download stytra from github and install it with:

```
pip install path_to_stytra/stytra
```

If you want to be able to change the stytra code and use the changed version, install using the -e argument:

```
pip install -e path_to_stytra/stytra
```

Now you can have a look at the stytra *Examples gallery*, or you can start *Configuring a computer for Stytra experiments*. In the second case, you might want to have a look at the camera APIs section below first.

Note: Stytra might raise an error after quitting because of a bug in the current version of pyqtgraph (a package we are using for online plotting). If you are annoyed by the error messages when closing the program you can install the develop version of pyqtgraph from their github repository. The problem will be resolved once the next pyqtgraph version is released.

1.2 Installing camera APIs

1.2.1 xiCam: Ximea

Download the Ximea SDK software pacakge for your operating system, during the installation wizard make sure that you select the python API checkbox. After installation, copy the python wrapper API (in the folder where you installed XIMEA, ...XIMEAAPIPythonv3ximea) into the Python site-packages folder (for anaconda, usually the folder ...anaconda3Libsite-packages)

1.2.2 pymba: AVT

Go to the Allied Vision software webpage and download and install the Vimba SDK. Then install the python wrapper pymba. You can install it from source:

```
pip install git+https://github.com/morefigs/pymba.git
```

or, if using 64bit windows, you can grab the installation file from here. open the terminal in the folder where you downloaded it and install:

```
pip install pymba-0.1-py3-none-any.whl
```

1.2.3 spinnaker: Point Grey / FLIR

Go the the FLIR support website, download the SDK and the Python API.

1. **Install the SDK, by chosing the camera and OS, and then downloading** e.g. Spinnaker 1.15.0.63 Full SDK - Windows (64-bit) — 07/27/2018 - 517.392MB or the equivalent for your operating system
2. **Install the python module** pip install "path_to_extracted_zip/spinnaker_python-1.15.0.63-cp36-cp36m-win_amd64.whl"

(with the file with the appropriate OS and Python versions)

1.2.4 National Instruments framegrabber with Mikrotron camera

Install the NI vision SDK. For the Mikrotron MC1362 camera, you can use this camera file. The camera file usually needs to be put into C:UsersPublicPublic DocumentsNational InstrumentsNI-IMAQData After putting the camera file there, is should be selected for the image acquisition device in NI MAX.

CHAPTER 2

Examples gallery

You don't need to get acquainted with the full feature set of stytra to start running experiments. Here and in the stytra/examples directory, we provide a number of example protocols that you can use to get inspiration for your own. In this section, we will illustrate general concepts of designing and running experiments with examples.

All examples in this section can be run in two ways: copy and paste the code in a python script and run it from your favorite IDE, or simply type on the command prompt:

```
python -m stytra.examples.name_of_the_example
```

2.1 Create a new protocol

To run a stytra experiment, we simply need to create a script were we define a protocol, and we assign it to a `Stytra` object. Running this script will create the Stytra GUI with controls for editing and running the protocol.

The essential ingredient of protocols is the list of stimuli that will be displayed. To create it, we need to define the `Protocol.get_stim_sequence()` method. This method returns a list of `Stimulus` objects which will be presented in succession.

In stytra.examples.most_basic_exp.py we define a very simple experiment:

Listing 1: ../../stytra/examples/most_basic_exp.py

```
from stytra import Stytra, Protocol
from stytra.stimulation.stimuli.visual import Pause, FullFieldVisualStimulus

# 1. Define a protocol subclass
class FlashProtocol(Protocol):
    name = "flash_protocol"  # every protocol must have a name.

    def get_stim_sequence(self):
        # This is the method we need to write to create a new stimulus list.
        # In this case, the protocol is simply a 1 second flash on the entire screen
```

(continues on next page)

(continued from previous page)

```
        # after a pause of 4 seconds:
        stimuli = [
            Pause(duration=4.),
            FullFieldVisualStimulus(duration=1., color=(255, 255, 255)),
        ]
        return stimuli


if __name__ == "__main__":
    # This is the line that actually opens stytra with the new protocol.
    st = Stytra(protocol=FlashProtocol())
```

It is important to note that stimuli should be instances, not classes!

Try to run this code or type in the command prompt:

```
python -m stytra.examples.most_basic_exp
```

This will open two windows: one is the main control GUI to run the experiments, the second is the screen used to display the visual stimuli. In a real experiment, you want to make sure this second window is presented to the animal. For details on positioning and calibration, please refer to *Calibration*

For an introduction to the functionality of the user interface, see *Stytra user interface*. To start the experiment, just press the play button: a flash will appear on the screen after 4 seconds.

2.1.1 Parametrise the protocol

Sometimes, we want to control a protocol parameters from the interface. To do this, we can define protocol class attributes as Param. All attributes defined as 'Param's will be modifiable from the user interface.

For a complete description of Params inside stytra see *Parameters in stytra*.

Listing 2: ../../stytra/examples/flash_exp.py

```
from stytra import Stytra, Protocol
from stytra.stimulation.stimuli.visual import Pause, FullFieldVisualStimulus
from lightparam import Param


class FlashProtocol(Protocol):
    name = "flash_protocol"  # every protocol must have a name.

    def __init__(self):
        super().__init__()
        # Here we define these attributes as Param s. This will automatically
        #  build a control for them and make them modifiable live from the
        # interface.
        self.period_sec = Param(10., limits=(0.2, None))
        self.flash_duration = Param(1., limits=(0., None))

    def get_stim_sequence(self):
        # This is the
        stimuli = [
            Pause(duration=self.period_sec - self.flash_duration),
            FullFieldVisualStimulus(
                duration=self.flash_duration, color=(255, 255, 255)
```

(continues on next page)

(continued from previous page)

```
            ),
        ]
        return stimuli


if __name__ == "__main__":
    st = Stytra(protocol=FlashProtocol())
```

Note that Parameters in Protocol param are the ones that can be changed from the GUI, but all stimulus attributes will be saved in the final log, both parameterized and unparameterized ones. No aspect of the stimulus configuration will be unsaved.

2.2 Define dynamic stimuli

Many stimuli may have quantities, such as velocity for gratings or angular velocity for windmills, that change over time. To define these kind of stimuli Stytra use a convenient syntax: a param_df pandas DataFrame with the specification of the desired parameter value at specific timepoints. The value at all the other timepoints will be linearly interpolated from the DataFrame. The dataframe has to contain a *t* column with the time, and one column for each quantity that has to change over time (*x*, *theta*, etc.). This stimulus behaviour is handled by the `Stimulus` class.

In this example, we use a dataframe for changing the diameter of a circle stimulus, making it a looming object:

Listing 3: ../../stytra/examples/looming_exp.py

```
import numpy as np
import pandas as pd

from stytra import Stytra
from stytra.stimulation import Protocol
from stytra.stimulation.stimuli import InterpolatedStimulus, CircleStimulus
from lightparam import Param


# A looming stimulus is an expanding circle. Stimuli which contain
# some kind of parameter change inherit from InterpolatedStimulus
# which allows for specifying the values of parameters of the
# stimulus at certain time points, with the intermediate
# values interpolated

# Use the 3-argument version of the Python type function to
# make a temporary class combining two classes


class LoomingStimulus(InterpolatedStimulus, CircleStimulus):
    name = "looming_stimulus"


# Let's define a simple protocol consisting of looms at random locations,
# of random durations and maximal sizes

# First, we inherit from the Protocol class
class LoomingProtocol(Protocol):

    # We specify the name for the dropdown in the GUI
```

(continues on next page)

(continued from previous page)

```
    name = "looming_protocol"

    def __init__(self):
        super().__init__()

        # It is convenient for a protocol to be parametrized, so
        # we name the parameters we might want to change,
        # along with specifying the the default values.
        # This automatically creates a GUI to change them
        # (more elaborate ways of adding parameters are supported,
        # see the documentation of lightparam)

        # if you are not interested in parametrizing your
        # protocol the the whole __init__ definition
        # can be skipped

        self.n_looms = Param(10, limits=(0, 1000))
        self.max_loom_size = Param(60, limits=(0, 100))
        self.max_loom_duration = Param(5, limits=(0, 100))
        self.x_pos_pix = Param(10, limits=(0, 2000))
        self.y_pos_pix = Param(10, limits=(0, 2000))

    # This is the only function we need to define for a custom protocol
    def get_stim_sequence(self):
        stimuli = []

        for i in range(self.n_looms):
            # The radius is only specified at the beginning and at the
            # end of expansion. More elaborate functional relationships
            # than linear can be implemented by specifying a more
            # detailed interpolation table

            radius_df = pd.DataFrame(
                dict(
                    t=[0, np.random.rand() * self.max_loom_duration],
                    radius=[0, np.random.rand() * self.max_loom_size],
                )
            )

            # We construct looming stimuli with the radius change specification
            # and a random point of origin within the projection area
            # (specified in fractions from 0 to 1 for each dimension)
            stimuli.append(
                LoomingStimulus(
                    df_param=radius_df, origin=(self.x_pos_pix, self.y_pos_pix)
                )
            )

        return stimuli


if __name__ == "__main__":
    # We make a new instance of Stytra with this protocol as the only option:
    s = Stytra(protocol=LoomingProtocol())
```

2.2.1 Use velocities instead of quantities

For every quantity we can specify the velocity at which it changes instead of the value itself. This can be done prefixing *vel_* to the quantity name in the DataFrame. In the next example, we use this syntax to create moving gratings. What is dynamically updated is the position *x* of the gratings, but with the dictionary we specify its velocity with *vel_x*.

Listing 4: ../../stytra/examples/gratings_exp.py

```
import numpy as np
import pandas as pd

from stytra import Stytra
from stytra.stimulation import Protocol
from stytra.stimulation.stimuli import MovingGratingStimulus
from lightparam import Param
from pathlib import Path


class GratingsProtocol(Protocol):
    name = "gratings_protocol"

    def __init__(self):
        super().__init__()

        self.t_pre = Param(5.)  # time of still gratings before they move
        self.t_move = Param(5.)  # time of gratings movement
        self.grating_vel = Param(-10.)  # gratings velocity
        self.grating_period = Param(10)  # grating spatial period
        self.grating_angle_deg = Param(90.)  # grating orientation

    def get_stim_sequence(self):
        # Use six points to specify the velocity step to be interpolated:
        t = [
            0,
            self.t_pre,
            self.t_pre,
            self.t_pre + self.t_move,
            self.t_pre + self.t_move,
            2 * self.t_pre + self.t_move,
        ]

        vel = [0, 0, self.grating_vel, self.grating_vel, 0, 0]

        df = pd.DataFrame(dict(t=t, vel_x=vel))

        return [
            MovingGratingStimulus(
                df_param=df,
                grating_angle=self.grating_angle_deg * np.pi / 180,
                grating_period=self.grating_period,
            )
        ]


if __name__ == "__main__":
    # We make a new instance of Stytra with this protocol as the only option
    s = Stytra(protocol=GratingsProtocol())
```

You can look in the code of the windmill_exp.py example to see how to use the dataframe to specify a more complex motion - in this case, a rotation with sinusoidal velocity.

Note: If aspects of your stimulus change abruptly, you can put twice the same timepoint in the param_df, for example: param_df = pd.DataFrame(dict(t = [0, 10, 10, 20], vel_x = [0, 0, 10, 10])

2.2.2 Visualise with stim_plot parameter

If you want to monitor in real time the changes in your experiment parameters, you can pass the stim_plot argument to the call to stytra to add to the interface an online plot:

Listing 5: ../../stytra/examples/plot_dynamic_exp.py

```
from stytra import Stytra


if __name__ == "__main__":
    from stytra.examples.gratings_exp import GratingsProtocol

    # We make a new instance of Stytra with this protocol as the only option:
    s = Stytra(protocol=GratingsProtocol(), stim_plot=True)
```

2.3 Stimulation and tracking

2.3.1 Add a camera to a protocol

We often need to have frames streamed from a file or a camera. In the following example we comment on how to achieve this when defining a protocol:

Listing 6: ../../stytra/examples/display_camera_exp.py

```
from stytra import Stytra
from stytra.stimulation.stimuli import Pause
from pathlib import Path
from stytra.stimulation import Protocol


class Nostim(Protocol):
    name = "empty_protocol"

    # In the stytra_config class attribute we specify a dictionary of
    # parameters that control camera, tracking, monitor, etc.
    # In this particular case, we add a stream of frames from one example
    # movie saved in stytra assets.
    stytra_config = dict(
        camera=dict(video_file=str(Path(__file__).parent / "assets" /
                                   "fish_compressed.h5")))

    #  For a streaming from real cameras connected to the computer, specify camera␣
↪type, e.g.:
    # stytra_config = dict(camera=dict(type="ximea"))
```

(continues on next page)

(continued from previous page)

```
    def get_stim_sequence(self):
        return [Pause(duration=10)]  # protocol does not do anything


if __name__ == "__main__":
    s = Stytra(protocol=Nostim())
```

Note however that usually the camera settings are always the same on the computer that controls a setup, therefore the camera settings are defined in the user config file and generally not required at the protocol level. See *Configuring a computer for Stytra experiments* for more info.

2.3.2 Add tracking to a defined protocol

To add tail or eye tracking to a protocol, it is enough to change the *stytra_config* attribute to contain a tracking argument as well. See the experiment documentation for a description of the available tracking methods.

In this example, we redefine the previously defined windmill protocol (which displays a rotating windmill) to add tracking of the eyes as well:

Listing 7: ../../stytra/examples/tail_tracking_exp.py

```
from pathlib import Path
from stytra import Stytra
from stytra.examples.gratings_exp import GratingsProtocol


class TrackingGratingsProtocol(GratingsProtocol):
    name = "gratings_tail_tracking"

    # To add tracking to a protocol, we simply need to add a tracking
    # argument to the stytra_config:
    stytra_config = dict(
        tracking=dict(embedded=True, method="tail"),
        camera=dict(
            video_file=str(Path(__file__).parent / "assets" / "fish_compressed.h5")
        ),
    )


if __name__ == "__main__":
    s = Stytra(protocol=TrackingGratingsProtocol())
```

Now a window with the fish image an a ROI to control tail position will appear, and the tail will be tracked! See *Configuring tracking of embedded fish* for instructions on how to adjust tracking parameters.

2.4 Closed-loop experiments

Stytra allows to simple definition of closed-loop experiments where quantities tracked from the camera are dynamically used to update some stimulus variable. In the example below we create a full-screen stimulus that turns red when the fish is swimming above a certain threshold (estimated with the vigour method).

Listing 8: ../../stytra/examples/custom_visual_exp.py

```
from stytra import Stytra, Protocol
from stytra.stimulation.stimuli import VisualStimulus
from PyQt5.QtCore import QRect
from PyQt5.QtGui import QBrush, QColor
from pathlib import Path


class NewStimulus(VisualStimulus):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.color = (255, 255, 255)

    def paint(self, p, w, h):
        p.setBrush(QBrush(QColor(*self.color)))  # Use chosen color
        p.drawRect(QRect(0, 0, w, h))  # draw full field rectangle

    def update(self):
        fish_vel = self._experiment.estimator.get_velocity()
        # change color if speed of the fish is higher than threshold:
        if fish_vel < -5:
            self.color = (255, 0, 0)
        else:
            self.color = (255, 255, 255)


class CustomProtocol(Protocol):
    name = "custom protocol"  # protocol name

    stytra_config = dict(
        tracking=dict(method="tail", estimator="vigor"),
        camera=dict(
            video_file=str(Path(__file__).parent / "assets" / "fish_compressed.h5")
        ),
    )

    def get_stim_sequence(self):
        return [NewStimulus(duration=10)]


if __name__ == "__main__":
    Stytra(protocol=CustomProtocol())
```

2.5 Freely-swimming experiments

For freely swimming experiments, it is important to calibrate the camera view to the displayed image. This is explained in *Calibration*. Then, we can easily create stimuli that track or change depending on the location of the fish. The following example shows the implementation of a simple phototaxis protocol, where the bright field is always displayed on the right side of the fish, and a centering stimulus is activated if the fish swims out of the field of view. Configuring tracking for freely-swimming experiments is explained here *Configuring tracking of freely-swimming fish*

Listing 9: ../../stytra/examples/phototaxis.py

```
from stytra import Stytra
from stytra.stimulation.stimuli import (
    FishTrackingStimulus,
    HalfFieldStimulus,
    RadialSineStimulus,
    FullFieldVisualStimulus,
)
from stytra.stimulation.stimuli.conditional import CenteringWrapper

from stytra.stimulation import Protocol
from lightparam import Param
from pathlib import Path


class PhototaxisProtocol(Protocol):
    name = "phototaxis"
    stytra_config = dict(
        display=dict(min_framerate=50),
        tracking=dict(method="fish", embedded=False, estimator="position"),
        camera=dict(
            video_file=str(Path(__file__).parent / "assets" / "fish_free_compressed.h5
↪"),
            min_framerate=100,
        ),
    )

    def __init__(self):
        super().__init__()
        self.n_trials = Param(120, (0, 2400))
        self.stim_on_duration = Param(10, (0, 30))
        self.stim_off_duration = Param(10, (0, 30))
        self.center_offset = Param(0, (-100, 100))
        self.brightness = Param(255, (0, 255))

    def get_stim_sequence(self):
        centering = RadialSineStimulus(duration=self.stim_on_duration)
        stimuli = []
        stim = type("phototaxis", (FishTrackingStimulus, HalfFieldStimulus), {})
        for i in range(self.n_trials):
            stimuli.append(
                CenteringWrapper(
                    stim_false=stim(
                        duration=self.stim_on_duration,
                        color=(self.brightness,) * 3,
                        center_dist=self.center_offset,
                    ),
                    stim_true=centering,
                )
            )
            stimuli.append(
                FullFieldVisualStimulus(
                    color=(self.brightness,) * 3, duration=self.stim_off_duration
                )
            )
```

(continues on next page)

(continued from previous page)

```
        return stimuli


if __name__ == "__main__":
    s = Stytra(protocol=PhototaxisProtocol())
```

2.6 Defining custom Experiment classes

New Experiment objects with custom requirements might be needed; for example, if one wants to implement more events or controls when the experiment start and finishes, or if custom UIs with new plots are desired. In this case, we will have to sublcass the stytra `Experiment` class. This class already has the minimal structure for running an experimental protocol and collect metadata. Using it as a template, we can define a new custom class.

2.6.1 Start an Experiment bypassing the Stytra constructor

First, to use a custom Experiment we need to see how we can start it bypassing the `Stytra` constructor class, which by design deals only with standard Experiment classes. This is very simple, and it is described in the example below:

Listing 10: ../../stytra/examples/no_stytra_exp.py

```
from stytra.experiments import Experiment
from stytra.stimulation import Protocol
import qdarkstyle
from PyQt5.QtWidgets import QApplication
from stytra.stimulation.stimuli import Pause, Stimulus


# Here ve define an empty protocol:
class FlashProtocol(Protocol):
    name = "empty_protocol"  # every protocol must have a name.

    def get_stim_sequence(self):
        return [Stimulus(duration=5.),]


if __name__ == "__main__":
    # Here we do not use the Stytra constructor but we instantiate an experiment
    # and we start it in the script. Even though this is an internal Experiment
    # subtype, a user can define a new Experiment subclass and start it
    # this way.
    app = QApplication([])
    app.setStyleSheet(qdarkstyle.load_stylesheet_pyqt5())
    protocol = FlashProtocol()
    exp = Experiment(protocol=protocol,
                     app=app)
    exp.start_experiment()
    app.exec_()
```

2.6.2 Customise an Experiment

To customize an experiment, we need to subclass `Experiment`, or the existing subclasses `VisualExperiment` and `TrackingExperiment`, which deal with experiments with a projector or with tracking from a camera, respectively. In the example below, we see how to make a very simple subclass, with an additional event (a mask waiting for an OK from the user) implemented at protocol onset. For a description of how the `Experiment` class work, refer to its documentation.

Listing 11: ../../stytra/examples/custom_exp.py

```
from stytra.experiments import Experiment
from stytra.stimulation import Protocol
import qdarkstyle
from PyQt5.QtWidgets import QApplication
from stytra.stimulation.stimuli import Stimulus
from PyQt5.QtWidgets import QMessageBox
from PyQt5.QtWidgets import QPushButton


# Here ve define an empty protocol:
class FlashProtocol(Protocol):
    name = "empty_protocol"  # every protocol must have a name.

    def get_stim_sequence(self):
        return [Stimulus(duration=5.),]


class CustomExperiment(Experiment):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        self.start = False

    def start_protocol(self):
        self.start = False

        msgBox = QMessageBox()
        msgBox.setText('Start the protocol when ready')
        msgBox.setStandardButtons(QMessageBox.Ok)
        ret = msgBox.exec_()
        super().start_protocol()



if __name__ == "__main__":
    # Here we do not use the Stytra constructor but we instantiate an experiment
    # and we start it in the script. Even though this is an internal Experiment
    # subtype, a user can define a new Experiment subclass and start it
    # this way.
    app = QApplication([])
    app.setStyleSheet(qdarkstyle.load_stylesheet_pyqt5())
    protocol = FlashProtocol()
    exp = CustomExperiment(protocol=protocol,
                           app=app)
    exp.start_experiment()
    app.exec_()
```

CHAPTER 3

Stimulation

One of the main purposes of stytra is to provide a framework to design and run sequences of stimuli to be presented to the fish.

3.1 Stimuli and Protocols in stytra

The `Stimulus` class constitutes the building block for an experiment in stytra. A sequence of Stimuli is bundled together and parameterized by the `Protocol` class. See Create stimulus sequence for a description of how to create a protocol in stytra.

The `ProtocolRunner` class is used to keep track of time and set the Stimuli in the Protocol sequence with the proper pace.

3.2 Stimuli examples

3.2.1 Full-field luminance

```
    def get_stim_sequence(self):
        lum = pd.DataFrame(dict(t=[0, 1, 2], luminance=[0.0, 1.0, 0.0]))
        return [
            DynamicLuminanceStimulus(df_param=lum, clip_mask=(0.0, 0.0, 0.5, 0.5)),
            DynamicLuminanceStimulus(df_param=lum, clip_mask=(0.5, 0.5, 0.5, 0.5)),
        ]
```

3.2.2 Gratings

```
    def get_stim_sequence(self):
        Stim = type("stim", (InterpolatedStimulus, GratingStimulus), dict())
        return [
            Stim(df_param=pd.DataFrame(dict(t=[0, 2], vel_x=[10, 10], theta=np.pi /␣
↪4)))
        ]
```

3.2.3 OKR inducing rotating windmill stimulus

```
    def get_stim_sequence(self):
        Stim = type(
            "stim", (InterpolatedStimulus, WindmillStimulus), {}  # order is␣
↪important!
        )
        return [Stim(df_param=pd.DataFrame(dict(t=[0, 2, 4], theta=[0, np.pi / 8,␣
↪0])))]
```

3.2.4 Seamlessly-tiled image

```
    def get_stim_sequence(self):
        Stim = type("stim", (SeamlessImageStimulus, InterpolatedStimulus), {})
        return [
            Stim(
                background="caustics.png",
                df_param=pd.DataFrame(dict(t=[0, 2], vel_x=[10, 10], vel_y=[5, 5])),
            )
        ]
```

3.2.5 Radial sine (freely-swimming fish centering stimulus)

```
    def get_stim_sequence(self):
        return [RadialSineStimulus(duration=2, period=10, velocity=5)]
```

CHAPTER 4

Configuring a computer for Stytra experiments

By default, Stytra checks the user folder (on Windows usually C:/Users/user_name, ~ on Unix-based systems) for the stytra_setup_config.json file. You can put default settings for the current computer in it, specifying the e.g. saving format, camera type and ROI, full-screen stimulus display and anything else that is specified when instantiating `Stytra`.

An example is provided below:

stytra_setup.config.json

```
{
"display": {"full_screen": true},
"dir_save": "J:/_Shared/experiments",
"dir_assets": "J:/_Shared/stytra_resources",
"log_format": "hdf5",
"camera": {"type": "ximea", "rotation":-1, "roi":[0, 0, 784, 784]},
"tracking": {"method":"fish"},
"embedded" : false
}
```

CHAPTER 5

Data and metadata saving

5.1 Data saving classes in Stytra

All streaming data (tracking, stimulus state) is collected by subclasses of the `Accumulator` Accumulators collect named tuples of data and timing of data points. If the data format changes, the accumulator resets.

All other data (animal metadata, configuration information, GUI state etc. is collected inside the Experiment class via tha `DataCollector`

5.2 Configuring Stytra for updating external database:

In addition to the JSON file, the metadata can be saved to a database, such as MongoDB. For this, an appropriate database class has to be created and passed to the Stytra class. This example uses PyMongo

Example:

```
from stytra.utilities import Database, prepare_json
import pymongo


class PortuguesDatabase(Database):
    def __init__(self):
        # in the next line you have to put in the IP address and port of the
        # MongoDB instance
        self.client = pymongo.MongoClient("mongodb://192.???.???.???:????")
        # the database and collection are created in MongoDB before
        # the first use
        self.db = self.client.experiments
        self.collection = self.db["experiments"]

    def insert_experiment_data(self, exp_dict):
        """ Puts a record of the experiment in the default lab MongoDB database
```

(continues on next page)

(continued from previous page)

```
        :param exp_dict: a dictionary from the experiment data collector
        :return: the database id of the inserted item
        """

        # we use the prepare_json function to clean the dictionary
        # before inserting into the database

        db_id = self.collection.insert_one(
            prepare_json(exp_dict, eliminate_df=True)
        ).inserted_id
        return str(db_id)
```

CHAPTER 6

Calibration

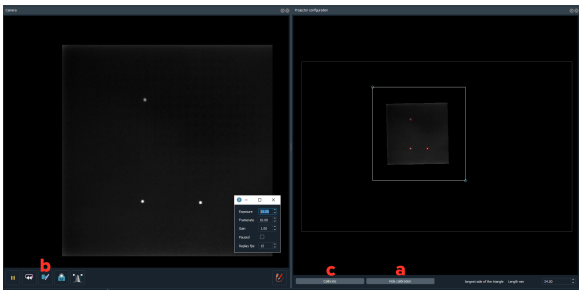
6.1 Positioning and calibrating the monitor for visual stimuli

To calibrate the monitor for your experiment, first position the black stimulus screen on the monitor you are using for the experiment. Then, hit the show calibration button and drag around the ROI in the stytra GUI until the red rectangle covers the area you want to use for the stimulus and the cross is at the center. Finally, specify in the spin box the final size of the lateral edge of the calibrator in centimeters.

The calibration is saved in the last_stytra_config.json file, so once you have done it it maintains the same calibration for all subsequent experiments.

6.2 Calibration of the camera and monitor

To calibrate the camera image to the displayed image, the Circle Calibrator is used (it is enabled automatically for freely-swimming experiments).



After Stytra starts, turn off the IR illumination and remove the IR filter in front of the camera. Then, click the display calibration pattern button (a) and move the display window on the projector so that the 3 dots are clearly visible. Sometimes the camera exposure has to be adjusted as well (b) so that all 3 dots are visible. Due to screen or projector framerates, usually setting the camera framerate to 30 and the exposure to 10ms works well.

Then, click calibrate (c) and verify that the location of the camera image in the projected image makes sense. If not, try adjusting camera settings and calibrating again.

CHAPTER 7

Triggering a Stytra protocol

Stytra is designed to be used in setups where the presentation of stimuli to the animal needs to be synchronized with an acquisition program running on a different computer, e.g. controlling a two-photon microscope. To this end, the triggering module provides classes to ensure communication with external devices to time the beginning of the experiment. Two methods are already supported in the triggering library:

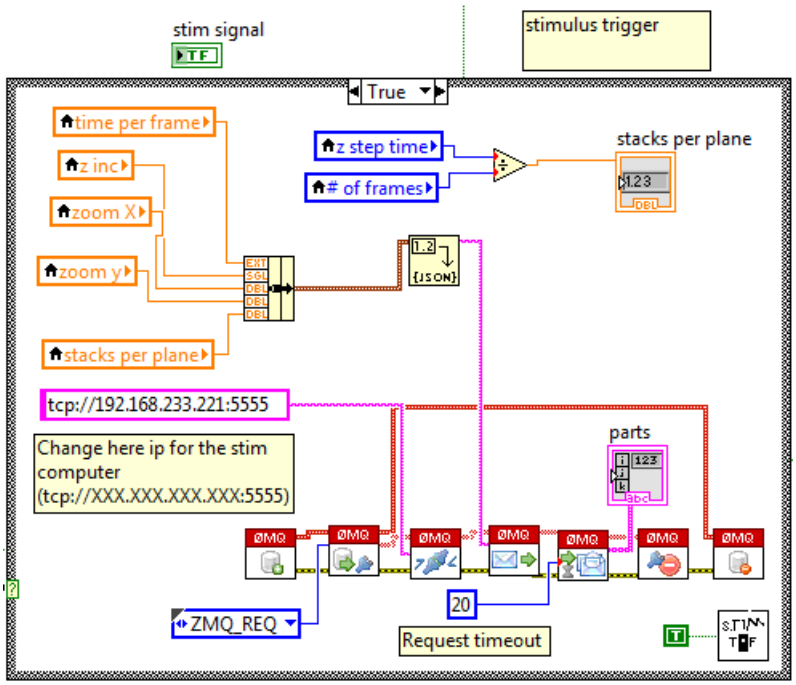
7.1 TTL pulse triggering on a Labjack/NI board and serial ports.

In the first simple configuration, Stytra simply waits for a TTL pulse received on a Labjack or a NI board to start the experiment.

7.2 ZeroMQ

Stytra employs the ZeroMQ library to synchronize the beginning of the experiment through a message coming from the acquisition computer over the local network. ZeroMQ is supported in a number of programming languages and environments including LabView, and the exchange of the synchronizing message can easily be added to custom-made or open-source software. The messages can also be used to communicate to Stytra data such as the microscope configuration that will be logged together with the rest of experiment metadata.

A common framework to build custom software for hardware control is LabView. In our laboratory, a LabView program is used to control the scanning from the two-photon microscope. Below we report a screenshot of a very simple subVI that can be used together with Stytra for triggering the start of the stimulation. A ZMQ context is created, and than used to send a json file with the information about microscope configuration over the network to the ip of the computer running Stytra, identified by its IP. Stytra uses by default port 5555 to listen for triggering messages.



7.3 Additional methods

The triggering module is also designed to be expandable. It is possible to define new kinds of triggers, which consists of a processes that continuously checks a condition. To define a new trigger, e.g., starting the acquisition when a new file is created in a folder, it is enough to write a method that uses the python standard library to monitor folder contents.

CHAPTER 8

Parameters in stytra

Various aspects of the experiments are parametrised using the lightparam package. For basic use patterns you can refer to the README of the package.

CHAPTER 9

A note on coordinate systems in Stytra

Stytra follows the common convention for displaying images on screens: the x axis increases to the right and the y axis increases downward, with (0,0) being the upper right corner. For the recorded coordinates, the same holds. The angles correspondingly increase clockwise.

CHAPTER 10

Hardware description

Below we provide a description of the setups in use in the lab together with a full list of components that can be used to assemble them.

Two configurations of our setups are described: the first one is for detailed kinematic tracking of eyes and tail in a fish with head restrained in agarose, the second for tracking freely swimming fish in a petri dish.

Finally, we present a cheap version of the behavioral setup that can be easily built for about 700 euros, and easily assembled using cardboard, laser-cut parts or other custom-made enclosures.

10.1 Head-restrained fish setup

This configuration requires high magnification, provided by a 50 mm macro objective. On the other side, illumination be provided only in a very small field and can be accomplished by with a simple single IR-LED.

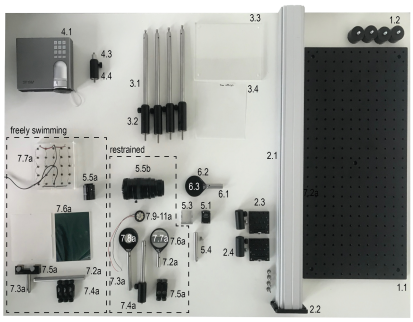
10.2 Freely swimming fish setup

This configuration uses a camera with larger field of view and a custom-built LED box for illuminating homogeneously a large area.

10.3 List of components

Below we provide a list of all components required for building the two setups. Indicative prices in euros (Jul 2018) and links to supplier pages are provided as well.

Note: Many parts of the setup, such as the base, the stage and the holders can easily be replaced with custom solutions.



10.3.1 Head-restrained setup

Table 1: Components for embedded configuration behavioral setup

	Component	Manufacturer	Part No	Specs	Amount	Price (euros)	Link	Notes	Replacement
Breadbord									
1.1	Aluminum bread-bord	Thorlabs	MB3045/M		1	170	https://www.thorlabs.com/thorproduct.cfm?partnumber=MB3045/M		
1.2	Feet	Thorlabs	AV4/M		1	20.86	https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=6421		
Rail									
2.1	Construction rail	Thorlabs	XT66-750		1	80	https://www.thorlabs.com/thorProduct.cfm?partNumber=XT66-750		

Continued on next page

Table 1 – continued from previous page

	Component	Manufacturer	Part No	Specs	Amount	Price (euros)	Link	Notes	Replacement
2.2	Construction rail mount	Thorlabs	XT66P1		1	32.25	https://www.thorlabs.com/thorproduct.cfm?partnumber=XT66P1		
2.3	Rail carriage	Thorlabs	XT66P2/M		2	62.25	https://www.thorlabs.com/thorproduct.cfm?partnumber=XT66P2/M		
2.4	Post holder	Thorlabs	PH20/M		2	6.33	https://www.thorlabs.com/thorproduct.cfm?partnumber=PH20/M		
Stage									
3.1	Post	Thorlabs	TR250/M		4	8.12	https://www.thorlabs.com/thorproduct.cfm?partnumber=TR250/M	can be smaller for embedded prep	
3.2	Post holder	Thorlabs	PH75/M		4	7.44	https://www.thorlabs.com/thorproduct.cfm?partnumber=PH75/M#ad-image-0		

Continued on next page

Table 1 – continued from previous page

	Component	Manufacturer	Part No	Specs	Amount	Price (euros)	Link	Notes	Replacement
3.3	Acrilic stage	custom	133555		1	10	https://www.modulor.de/acrylglas-gs-transparent-farblos-6-00-x-250-x-500-m.html		
3.4	Screen	custom							
Projector									
4.1	Projector	Asus	P3E		1	534	https://www.asus.com/us/Projectors/P3E/		
4.2	Display cable (HDMI)	Any							
4.3	Post	Thorlabs	TR40/M		1	4.52	https://www.thorlabs.com/thorproduct.cfm?partnumber=TR40/M		
4.4	Post holder	Thorlabs	PH40/M		1	6.56	https://www.thorlabs.com/thorproduct.cfm?partnumber=PH40/M#ad-image-0		
Camera									
5.1	Camera	Ximea	MQ013MG-ON	Python 1300	1	580	https://www.ximea.com/products/usb3-vision-cameras-xiq-line/mq013mg-on		PointGrey Blackfly S Mono 0.4 MP USB3 Vision (Sony IMX287) https://eu.ptgrey.com/blackfly-s-mono-04-mp-usb

Continued on next page

Table 1 – continued from previous page

	Component	Manufacturer	Part No	Specs	Amount	Price (euros)	Link	Notes	Replacement
5.2	Camera cable	e.g. Ximea	CBL-U3-3M0	USB 3, 3m passive	1	17	https://www.ximea.com/en/products/usb3-vision-compliant-cameras-xiq/xiq-usb-30-accessories/1-m-usb-30-passive-cable		
5.3	Camera holder	custom			1	0			
5.4	Post	Thorlabs	TR75/M		1	4.93	https://www.thorlabs.com/thorproduct.cfm?partnumber=TR75/M		
5.5b	Camera objective	Navitar	TC.5028	C mount	1	590	https://navitar.com/products/imaging-optics/telecentric/video-telecentric/		
IR filter									
6.1	Post	Thorlabs	TR75/M		1	4.93	https://www.thorlabs.com/thorproduct.cfm?partnumber=TR75/M		
6.2	Lens mount	Thorlabs	LMR2/M		1	23	https://www.thorlabs.com/thorproduct.cfm?partnumber=LMR2/M		

Continued on next page

Table 1 – continued from previous page

	Component	Manufacturer	Part No	Specs	Amount	Price (euros)	Link	Notes	Replacement
6.3	IR filter	Edmund Optics	66-106	830 nm LP	1	69	https://www.edmundoptics.de/optics/optical-filters/longpass-edge-filters/rg-830-50mm-dia.-longpass-filter/		
Illumination									
7.1	Power supply	Conrad	ESPS-1500	Voltcraft	1	15	https://www.conrad.com/ce/en/product/1380523/Main		
							PSU-adjustable-voltage——VOLTCRAFT——ESPS-1500——3-Vdc-45-Vdc-5-Vdc		
7.2b	Post	Thorlabs	TR150/M		1	8.12	https://www.thorlabs.com/thorproduct.cfm?partnumber=TR150/M		
7.3b	Post	Thorlabs	TR50/M		2	8.12	https://www.thorlabs.com/thorproduct.cfm?partnumber=TR50/M		
7.4b	Post holder	Thorlabs	PH40/M		1	6.56	https://www.thorlabs.com/thorproduct.cfm?partnumber=PH40/M#ad-image-0		

Continued on next page

Table 1 – continued from previous page

	Component	Manufacturer	Part No	Specs	Amount	Price (euros)	Link	Notes	Replacement
7.5b	Right-angle clamp	Thorlabs	RA90/M		2	8.87	https://www.thorlabs.com/thorProduct.cfm?partNumber=RA90/M		
7.6b	Lens mount	Thorlabs	LMR2/M		1	23	https://www.thorlabs.com/thorproduct.cfm?partnumber=LMR2/M		
7.7b	cold mirror	Edmund Optics	#64-450		1	75	https://www.edmundoptics.de/optics/optical-mirrors/hot-cold-mirrors/45deg-aoi-50.0mm-diameter-cold-mirror/		
7.8b	LED holder	Thorlabs	SMR1/M		1	17	https://www.thorlabs.com/thorproduct.cfm?partnumber=SMR1/M		
7.9b	Cap for LED	Thorlabs	SM1CP2M		1	16	https://www.thorlabs.com/thorproduct.cfm?partnumber=SM1CP2M	machined to accomodate wires	

Continued on next page

Table 1 – continued from previous page

	Component	Manufacturer	Part No	Specs	Amount	Price (euros)	Link	Notes	Replacement
7.10	high power LED	RS Components	e.g. 796-1772	850 nm LED	1	10	https://de.rs-online.com/web/p/led-ir/7961772/	max 1 A power	LED-tech, Osram Black 850 nm (LT-2418) (https://www.led-tech.de/de/OSRAM-Black-Series-850n
7.11	LED pad	LED-tech	LT-2418		10	0.4	https://www.led-tech.de/de/Waermeleitklebepad-fuer-16mm-Star		
7.12	buck pack	Digikey	RCD-24-1.00/W/X3		1	22	https://www.digikey.com/product-detail/en/recom-power/RCD-24-1.00-W-X3/945-1131-ND/2256311	buck pack that has max 1 A power	wired and dimmable using PWM and/or analogue in
					Total	2586			

10.3.2 Freely-swimming setup

Table 2: Components for freely swimming configuration behavioral setup

	Component	Manufacturer	Part No.	Specs	Amount	Price (euros)	Link	Notes	Replacement
Breadbord									
1.1	Aluminum breadbord	Thorlabs	MB3045/M		1	170	https://www.thorlabs.com/thorproduct.cfm?partnumber=MB3045/M		

Continued on next page

Table 2 – continued from previous page

	Component	Manufacturer	Part No.	Specs	Amount	Price (euros)	Link	Notes	Replacement
1.2	Feet	Thorlabs	AV4/M		1	20.86	https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=6421		
Rail									
2.1	Construction rail	Thorlabs	XT66-750		1	80	https://www.thorlabs.com/thorProduct.cfm?partNumber=XT66-750		
2.2	Construction rail mount	Thorlabs	XT66P1		1	32.25	https://www.thorlabs.com/thorproduct.cfm?partnumber=XT66P1		
2.3	Rail carriage	Thorlabs	XT66P2/M		2	62.25	https://www.thorlabs.com/thorproduct.cfm?partnumber=XT66P2/M		
2.4	Post holder	Thorlabs	PH20/M		2	6.33	https://www.thorlabs.com/thorproduct.cfm?partnumber=PH20/M		
Stage									

Continued on next page

Table 2 – continued from previous page

	Component	Manufacturer	Part No.	Specs	Amount	Price (euros)	Link	Notes	Replacement
3.1	Post	Thorlabs	TR250/M		4	8.12	https://www.thorlabs.com/thorproduct.cfm?partnumber=TR250/M		
3.2	Post holder	Thorlabs	PH75/M		4	7.44	https://www.thorlabs.com/thorproduct.cfm?partnumber=PH75/M#ad-image-0		
3.3	Acrilic stage	custom			1	10	https://www.modulor.de/acrylglas-gs-transparent-farblos-6-00-x-250-x-500-m.html	From item n. 0133555 of modulor	
3.4	Screen	custom							
Projector									
4.1	Projector	Asus	P3E		1	534	https://www.asus.com/us/Projectors/P3E/		
4.2	Display cable (HDMI)	Any							
4.3	Post	Thorlabs	TR40/M		1	4.52	https://www.thorlabs.com/thorproduct.cfm?partnumber=TR40/M		

Continued on next page

Table 2 – continued from previous page

	Component	Manufacturer	Part No.	Specs	Amount	Price (euros)	Link	Notes	Replacement
4.4	Post holder	Thorlabs	PH40/M		1	6.56	https://www.thorlabs.com/thorproduct.cfm?partnumber=PH40/M#ad-image-0		
Camera									
5.1	Camera	Ximea	MQ013MG-ON	Python 1300	1	580	https://www.ximea.com/products/usb3-vision-cameras-xiq-line/mq013mg-on		Camera Point-Grey Blackfly S Mono 0.4 MP USB3 Vision (Sony IMX287) 305 https://eu.ptgrey.com/blackfly-s-mono-04-mp-usb3
5.2	Camera cable	e.g. Ximea	CBL-U3-3M0	USB 3, 3m passive	1	17	https://www.ximea.com/en/products/usb3-vision-compliant-cameras-xiq/xiq-usb-30-accessories/1-m-usb-30-passive-cable		
5.3	Camera holder	custom			1	0			
5.4	Post	Thorlabs	TR75/M		1	4.93	https://www.thorlabs.com/thorproduct.cfm?partnumber=TR75/M		

Continued on next page

Table 2 – continued from previous page

	Component	Manufacturer	Part No.	Specs	Amount	Price (euros)	Link	Notes	Replacement
5.5a	Camera lens	Edmund Optics	59-872	C mount 35mm	1	295	https://www.edmundoptics.com/imaging-lenses/fixed-focal-length-lenses/35mm-c-series-fixed-focal-length-lens/#specs		
IR filter									
6.1	Post	Thorlabs	TR75/M		1	4.93	https://www.thorlabs.com/thorproduct.cfm?partnumber=TR75/M		
6.2	Lens mount	Thorlabs	LMR2/M		1	23	https://www.thorlabs.com/thorproduct.cfm?partnumber=LMR2/M		
6.3	IR filter	Edmund Optics	66-106	830 nm LP	1	69	https://www.edmundoptics.de/optics/optical-filters/longpass-edge-filters/rg-830-50mm-dia.-longpass-filter/		
Illumination									
7.1	Power supply	Conrad	ESPS-1500	Voltcraft	1	15	https://www.conrad.com/ce/en/product/1380523/Main PSU-adjustable-voltage——VOLTCRAFT——ESPS-1500——3-Vdc-45-Vdc-5-Vdc		

Continued on next page

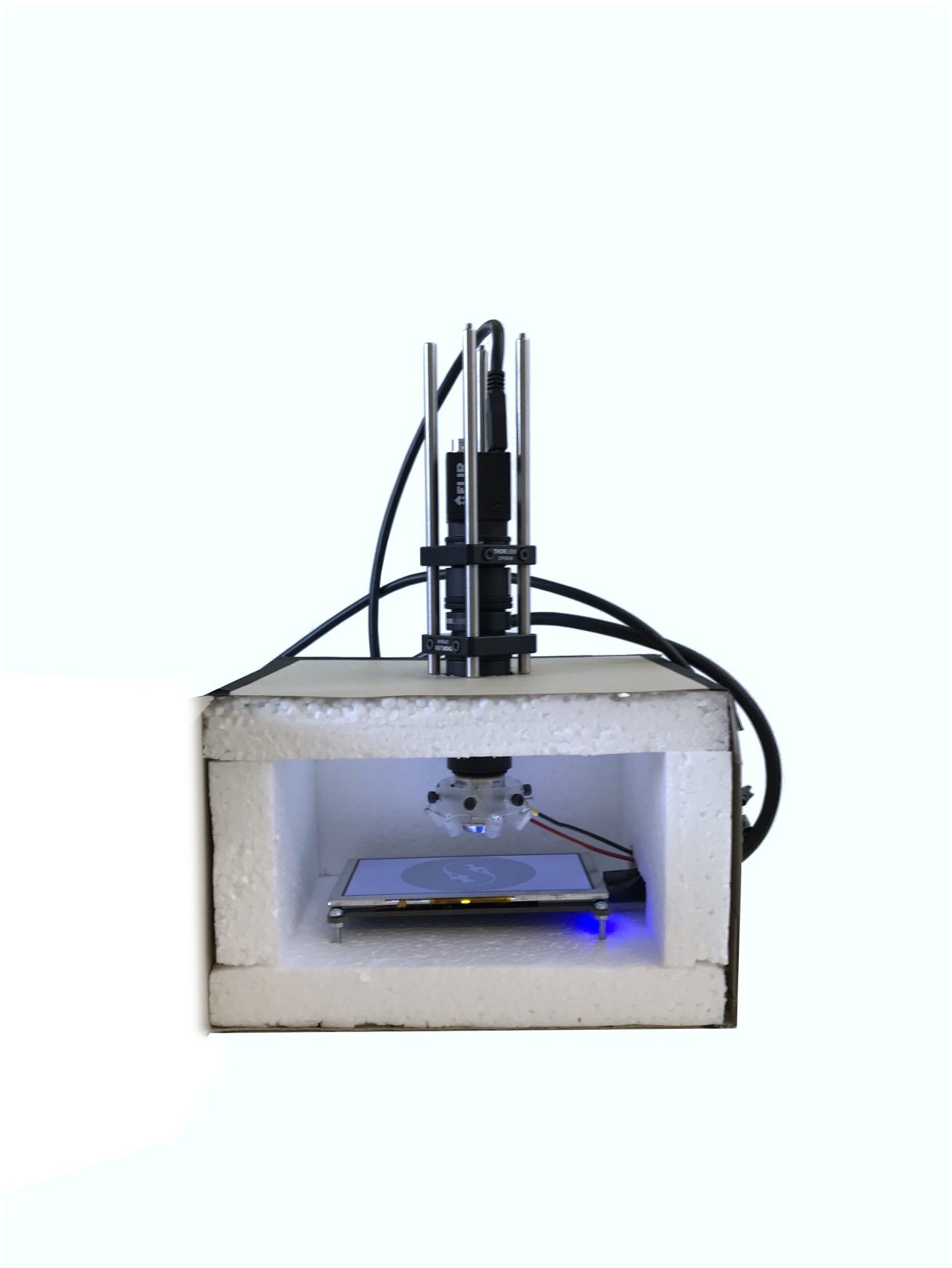
Table 2 – continued from previous page

	Component	Manufacturer	Part No.	Specs	Amount	Price (euros)	Link	Notes	Replacement
7.2a	Post	Thorlabs	TR50/M		1	8.12	https://www.thorlabs.com/thorproduct.cfm?partnumber=TR50/M		
7.3a	Post	Thorlabs	TR150/M		1	8.12	https://www.thorlabs.com/thorproduct.cfm?partnumber=TR150/M		
7.4a	Right-angle clamp	Thorlabs	RA90/M		3	8.87	https://www.thorlabs.com/thorProduct.cfm?partNumber=RA90/M		
7.5a	Mirror holder	Edmund Optics	#54-997		1	70	https://www.edmundoptics.com/optomechanics/optical-mounts/optical-filter-mounts/40mm-sq.-fixed-filter-holder/		
7.6a	Cold mirror	Edmund Optics	#62-633		1	20	https://www.edmundoptics.com/optics/optical-mirrors/hot-cold-mirrors/45deg-aoi-12.5mm-sq-cold-mirror/		
7.7a	LED box	custom			1				
					Total	2199			

10.4 Low-cost behavioral setup

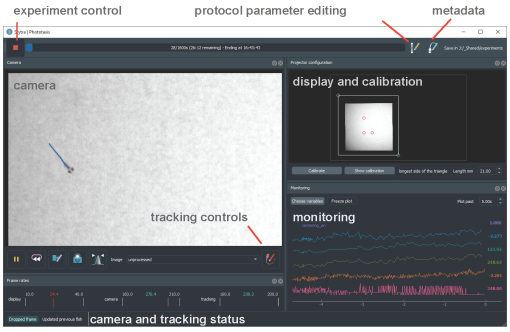
A very cheap version of the behavioral setup can be built by replacing the projector with small LED display and the camera lens with a fixed focal length objective. The dimensions of this setup are quite small and parts can be kept in

place with a basic custom-made frame that can be laser-cut or even made out of cardboard.



CHAPTER 11

Stytra user interface



The toolbar on top controls running of the protocols: starting and stopping, progress display, opening a dialog for protocol settings, changing the metadata and save destination.

The rest of the interface is reconfigured depending on the experiment type. Each panel can be moved separately and closed. To reopen a closed panel, you can right-click on the title bar of any panel and a list of all available panels will appear.

The camera panel buttons are for:

- pausing and starting the camera feed
- activating the replay (for a region selected when the camera is paused). Refer to *Replaying the camera feed to refine tracking* section for details.
- adjusting camera settings (framerate, exposure and gain)
- capturing the current image of the camera (without the tracking results superimposed
- turning on and off auto-scaling of the image brightness range.
- selection box to display the image at a particular stage in the tracking pipeline
- button for editing the tracking settings

The framerate display widget shows current framerates of the stimulus display, camera and tracking. If minimum framerates for display or tracking are configuresd, the indicators turn red if the framerate drops. These are configured in the stytra_config dict for a protocol or setup_config.json file in the following sections:

```
stytra_config = dict(
        display=dict(min_framerate=50),
        camera=dict(min_framerate=100),
    )
```

The monitoring widget shows changing variables relating to the stimulus, tracking or estimation of the animal state for closed-loop stimulation use

The status bar shows diagnostic messages from the camera or tracking.

CHAPTER 12

Image processing pipelines

Image processing and tracking pipelines are defined by subclassing the `Pipeline` class. The pipelines are defined as trees of nodes, starting from the camera image with each node parametrized using lightparam. The image processing nodes are subclasses of `ImageToImageNode` whereas the terminal nodes are `ImageToDataNode`

Attributes of pipelines are:

- a tree of processing nodes, along
- (optional) a subclass of the camera window which displays the tracking overlay
- (optional) an extra plotting window class

the nodes can be set as attributes of the class, with names that are arbitrary except for how they are used by the display and plotting classes (see the `stytra.experiments.fish_pipelines` for examples)

12.1 Processing nodes

There are two types of nodes: `ImageToImageNode` and *ImageToDataNode <stytra.tracking.pipelines.ImageToDataNode>*

Nodes must have: - A name

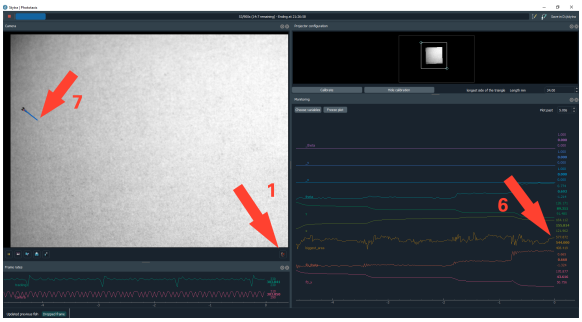
- A _process function which contains optional parameters

as keyword arguments, annotated with Params for everything that can be changed from the user interface. The _process function **has to** output a `NodeOutput` named tuple (from stytra.tracking.pipeline) which contains a list of diagnostic messages (can be empty), and either an image if the node is a ImageToImageNode or a NamedTuple if the node is a ImageToDataNode

Optionally, if the processing function is stateful (depends on previous inputs), you can define a reset function which resets the state.

CHAPTER 13

Configuring tracking of freely-swimming fish



1. Open the tracking settings window
2. Input the number of fish in the dish
3. Determine the parameters for background subtraction bglearning_rate and bglearn_every The diagnostic display can be invoked by putting display_processed to different states
4. Once you see the fish nicely, adjust the thresholded image, so that the full fish, but nothing more, is white bgdif_threshold
5. Adjust the eye threshold so that the eyes and swim bladder are highlighted (by changing the display_processed parameter) threshold_eyes
6. Adjust the target area: look at the biggest_area plot, if the background is correctly subtracted and a fish is in the field of view, the value should equal the current area of the fish. Choose a range that is comfortably around the current fish are
7. Adjust the tail length: the red line tracing the tail should not go over the actual tail.
8. If the fish jumps around too much, adjust the prediction uncertainty.

CHAPTER 14

Configuring tracking of embedded fish

1. Ensure that the exposure time is not longer than 1.5 miliseconds, otherwise the tracking will not be correct for fast tail movements
2. Open the tracking settings window
3. Invert the image if the tail is dark with respect to the background
4. Set the camera display to filtered and adjust clipping until the fish is the only bright thing with respect to the background, which has to be completely black.
5. Make the image as small as possible (with image_scale) as long as the tail is mostly more than 3px wide and filter it a bit (usually using filter_size=3)
6. Adjust the number of tail segments, around 30 is a good number. Usually, not more than 10 n_output_segments are required

7) Tap the dish of the fish so that it moves, and ensure the tail is tracked correctly. You can use the replay function to ensure the whole movement is tracked smoothly

8. To ensure the tracking is correct, you can enable the plotting of the last bout in the windows

CHAPTER 15

Replaying the camera feed to refine tracking

The replay functionality allows a frame-by-frame view of the camera feed during a period of interest (e.g. a bout or a struggle). After an interesting event happens and you can see it in the plot, pause the camera with the pause button. Use the two gray bars in the plots, select the time-period of interest. Then, enable the replay with the button underneath the camera, and unpause the camera feed. Now, the selected slice of time is replayed, and the framerate of the replay can be adjusted in the camera parameters. To go back to the live feed, toggle the replay button.

CHAPTER 16

Modules

stytra The root module, contains the Stytra class for running the experiment (selecting the appropriate experiment subtypes and setting the parameters)

stytra.experiments The controller classes organizing different kinds of experiments (with and without behavioral tracking, closed loop stimulation and with video recording). The classes put together everything required for a particular kind of experiment

stytra.gui Defines windows and widgets used for the different experiment types

stytra.hardware Communication with external hardware, from cameras to NI boards

stytra.triggering Communication with other equipment for starting or stopping experiments

stytra.metadata Classes that manage the metadata

stytra.stimulation Definitions of various stimuli and management of experimental protocols

stytra.calibration Classes to register the camera view to the projector display and set physical dimensions

stytra.tracking Fish, eye and tail tracking functions together with appropriate interfaces

CHAPTER 17

Indices and tables

- genindex
- modindex
- search

Index

S