autoFISH -

automated imaging with a fluidics system

- **Before getting started, try to read the entire documentation to get a feeling for how this package works. Some things might otherwise be confusing first.**
- **TESTED FOR WIN 10 only**: most microscope control packages work only under Windows.

Overview

Here, we provide a brief overview of how autofish works and what components are needed. The subsequent chapters provide a more detailed description.

In short, autofish is a Python package that performs sequential fluidic runs on a microscope, alternating between complex buffer exchanges and automate image acquisition.

- The **used fluidics system** is home-made based on widely available and cheap components. It is strongly based on published approaches (Boettiger et al., 2016; Moffitt and Zhuang, 2016).
- **Image acquisition** is either done with the Python package PycroManager (Pinkard et al., 2021) or by a simple synchronization with a shared text file.

Python controls the entire workflow, which also allows to include additional data-processing steps if required, e.g. some image analysis to verify data integrity or to already launch pre-processing steps to gain time.

Configurations: demo and standard experiments

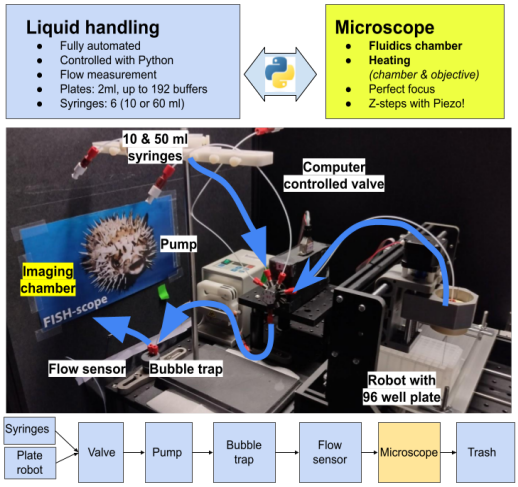
We provide **demo configuration** files that allow testing the different modules without connecting to actual hardware. This allows testing a proper installation of autofish. Demo configurations can be found in the **`demo` subfolder**. We also refer to these demo tests in the respective sections.

We also provide the configurations we use on our system (Nikon Ti) to perform standard experiments **here**.

Fluidics system

We use a custom-build fluidics system to perform the liquid handling. It allows to automatically perform buffer exchanges.

We provide detailed building plans in the dedicated document found **here**.



Microscope control

We provide several options to start acquisitions on a microscope.

1. Image acquisition can be performed with the Python package **PycroManager** (Pinkard et al., 2021). This allows launching multi-D acquisition events with Micromanager 2.

2. Synchronization can also be done with a **shared text file**. Two options exists :
    a. **Content** of the text file: content is set to 1 once the acquisition should be started. The acquisition software must set the value back to 0 once the imaging is terminated. Then the next fluidic run is started.
    b. **Existence** of text file **:** autoFISH creates this text file to indicate that the acquisition should be started. Once the acquisition is terminated, the acquisition software must delete the file to indicate that imaging is terminated.
3. **TTL synchronization.** This option allows to communicate with an acquisition software via TTL pulses, e.g. for communication with LEICA microscopes. The current implementation relies on a microcontroller (Arduino, Rasberry Pi) to send / receive the TTL pulses. These controllers are cheap and easy to setup.

New contributions

We develop autofish in a modular and extensible fashion. This design should make it relatively easy to add new hardware components, e.g. a different pump or a different acquisition routine.

If you want to integrate your own development in *autofish*, you can create a pull request on GitHub and after verification, we can integrate these. Alternatively, you can also contact us to discuss how to best integrate your contribution.

Software installation

Code is hosted on GitHub: https://github.com/fish-quant/autofish

Below is a recommended workflow for getting started, other ways exist as well.

Install miniconda and autoFISH

This you only have to do once.

1. Download the latest **version of miniconda** (Python 3.X) from https://docs.conda.io/en/latest/miniconda.html

2. Open Anaconda terminal and Create dedicated environment:
   `conda create --name autofish python=3.9`
3. Activate environment:
   `conda activate autofish`
4. Install autofish
   `pip install -i https://test.pypi.org/simple/ autofish`

Starting autofish

1. Activate environment:
   `conda activate autofish`
2. Start user interface
   `autofish`

For developers

We use Visual studio code for developments, but of course any Python IDE works. Installation link can be found **here**.

Change default terminal to Command Prompt

Under Windows, we had some issues when using PowerShell as a Terminal, when changing this to Command Prompt, things worked out fine. To do this, perform the following steps in Visual studio code

1. Open Visual Studio Code.
2. Press CTRL + SHIFT + P to open the Command Palette.
3. Search for “Terminal: Select Default Profile”
4. Select “Command prompt”

Setting up Visual Studio code

1. Launch Visual studio code.
2. From the menu `File > Add folder to workspace`
3. Select folder containing the Python code.

4. Double click on the file **`coordinator_gui.py`** located in the folder **`autofish`**.
5. Running this script will open the user-interface of autofish by pressing on the Run symbol (arrow) above the code editor window.
6. The first time, you have to specify:
    a. … the conda environment you defined above.
    b. … if you would like to install the Python extension: accept.
7. You can save this workspace to a file, which can then be opened for subsequent usage, and which will also remember the conda environment: *File* > *Save workspace as*

A typical autoFISH experiment

We provide different simple user interfaces to define both the microscopy and the fluidics. These are explained in dedicated sections below. The entry point is a **launch pad** (opening with the command `autofish`), permitting access to these control interfaces, and then start complete run. A typical workflow consists of

A typical autoFISH experiment

Below, we described a typical workflow for an autofish experiment. All steps are explained in more detail in the following sections.

1. Open **fluidics interface** to define the fluidics run. This includes specifying the sequence of buffer exchanges, but also preparing the system by priming the tubing. Please note that you can also run selected cycles without an acquisition.
2. Specify acquisition in the **acquisition software**, e.g. micromanager.
3. Setup acquisition in autofish
4. Initiate the controller to permit a fully automated run.
5. Run the full fluidics run.

Considerations

- Only the first round contains a **DAPI** stain. Only for this round the DAPI image is acquired.
- **Imaging settings** are defined after the first round, since here cells can be seen.
- For the remaining rounds, the experiment is automated.

Preparation

1. Get **samples** ready:
    a. Mount imaging chamber.
    b. Prepare all buffers.
2. Prepare **autofish** system:
    - Define experimental settings in config file.
    - Start fluidics flow sensor and start logging into file.
    - Start autofish and initiate (config files, zero robot)
    - Prime lines and check flow (replace tubing if necessary)
    - Hardware (easy to forget)
    - Temperature control (both objective and chamber)
    - Vacuum pump

Actual experiment

1. Run first hybridization round (including DAPI)
2. Micromanager: define acquisition.
    a. Include DAPI channel.
    b. Save position list.
3. Autofish: setup imaging
    a. Adjust microscope config file: channels and z-stacks.

    b. Open PycroManager module in autofish
        i. Load config files (position, microscope)
        ii. Specify folder to save images.
        iii. Perform acquisition for first round.
4. Start automated runs.
    a. Initiate controller.
    b. Launch all runs.

Turning of system

1. Clean system : flush all fluidic lines with water.
2. Turn off hardware.
3. Detach tubing from peristaltic pump.

Other scenarios

Acquisition specification after the first run

Often the **acquisition might be defined after the first fluidics run,** i.e. because no labeling was performed on the bench. Here, you can run the first run manually from the fluidics interface, define your acquisition and acquire the images for this first run, before going into automated mode.

Acquisition of additional channels

If you acquire **additional channels** in the first round, e.g. DAPI, we recommend **acquiring them last.** Like this, the numeric channel identifiers will not get mixed up, e.g. FISH will always be the first channel to be acquired.

Microscopy control

In the pulldown menu of the launchpad, you can select which type of imaging synchronization you would like to use.

PycroManager

Here, the acquisition is performed with micromanager 2 (Edelstein et al., 2014) with the Python package PycroManager (Pinkard et al., 2021). This requires installation of

- Latest nightly build of micromanager 2 from **here**.

Micromanager, PycroManager, headless

We refer to the excellent documentation of PycroManager, for how to get started and verify that the installation works properly.

PycroManager allows **headless connections** to micromanager. This means, no open instance of micromanager is required, and only essential elements of the micromanager core are started from within Python. In our hands, this allowed for more stable acquisitions.

Typical workflow

1. We usually open a micromanager instance to **set up the acquisition**:
    a. Finding adequate **exposure times** for the different channels.
    b. Specifying the **different positions** that should be acquired. These positions are then exported as a position list for import into autofish. ToDo: show screen shot.
2. At this stage, micromanager can be closed if a headless acquisition is performed.
3. Update the microscope configuration file (see below) to specify the z-range and the channels that should be acquired.
4. In the Python autoFISH GUI, select `pycromanager` from the dropdown menu and press `Specify acquisition`. This will open the window shown below:

    a. Load configuration file.
    b. Load position list.
    c. Specify folder to save images.
    d. Connect to micromanager.
    e. Create the acquisition event.

f. [Optional] launch a test acquisition, this can also be used to acquire the images of the first fluidics run.

5. The interface can now be closed (unless you want to perform different acquisitions for the first round as for the remaining ones)..

Microscope configuration

All acquisition parameters, expect the position lists, are specified in the file `microscope_config.yaml`

Some of the parameters are specific to your microscope, such as the name of the XY and Z drive, the provided names are for an inverted Nikon Ti. You might need to adapt this for your microscope.

A typical file is shown below:

```yaml
Microscope:
    type: 'NIKON_TI'
    xy_drive_name: 'TIXYDrive'
    Core-TimeoutMs: 1500
    stage-speed: 6
    z_drive_name: 'TIZDrive'
    channel_group: 'channels'
    channels: ['Cy3_75', 'DAPI_5']
    channel_exposures_ms: [100,50]
    channel_blank: DAPI_0
    z_start: 0
    z_end: 6
    z_step: 0.4
    mm_app_path: 'C:\Program Files\Micro-Manager-2.0-nightly'
    mm_config_file: 'MMConfig_demo.cfg'
```

The different fields are described next. Most of these are specific for a NikonTi widefield microscope. Adding support for other microscopes can be achieved my modification of the autoFISH codebase.

Command	Action
`type`	Specify your microscope. This will essentially determine how the position list is read (see function `load_position_list` in imager.py) Currently implemented are ● **Demo**: for test purposes only with demo MM config file. Small position list is created. Allows testing the PycroManager installation. ● **NIKON_TI**: for an inverted Nikon Ti.
`xy_drive_name` `z_drive_name`	Names of the XY and Z drives. Will depend on your microscope. These are used to interpret the loaded positions lists.
`stage_speed`	Speed at which the stage is moving. For NikonTi from 1 (fast) to 8 (slow). We use slower speeds when larger displacements are performed, e.g. for Ibididi multi-channel slides. This helps to keep the autofocus system

	engaged.
`Core-TimeoutMs`	Time-out for Core after which an error will be shown. Default is 5000, we set this to higher values when the stage moves slowly.
`channel_group`	Name of channel group containing the channels that will be acquired.
`channels`	List of all channels that will be acquired.
`channel_exposures_ms`	List of exposure times (in ms) of the acquired channels specified in channel.
`channel_blank`	[**optional**] Channel name used for a blank acquisition. When specified one additional image will be taken at the first position without a z-stack using this channel. This can be necessary when the light source is not automatically turned off after acquisition.

`z_start`
`z_end`
`z_step`
	Positions of z-stack relative to specified Z-position in position list.

Only needed for headless start of micromanager:

`mm_app_path`	Path to micromanager
`mm_config_file`	Name of micromanager config file

Position list

The different positions that should be acquired are stored in a .pos file. In autofish, these positions are then read into python as XYZ coordinates (`imager.py` > `pycromanager` > `load_position_list`). This routine might need to be adapted to your microscope.

Demo

You can load the demo file `microscope_config__demo.yaml`.

You have two different options.

1. Open micromanager, and connect normally.
2. Update the micromanager path and name of the config file and connect headless.

Known issues

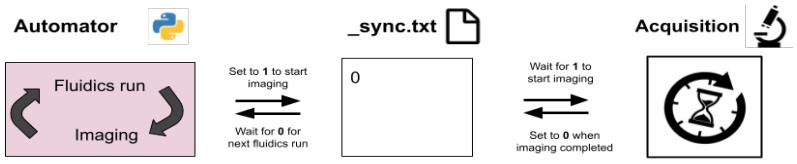
If you see error messages complaining about ZMQ, a likely explanation is that you don't have the latest nightly build of micromanager and/or the latest version of PycroManager. Try to download the latest micromanager version, and update PycroManager:

- 1. Open conda terminal
- 2. Activate environment: `conda activate autofish3`
- 3. Update pycromanager: `` `pip install -U pycromanager ``

Text file synchronization

Here, a shared text file is used to establish communication between autofish and the image acquisition software. Implementations exist in two flavors:

1. This text file **contains** 1 to indicate that the acquisition should be launched, and it must be set back to 0 by the acquisition software once the imaging is terminated.
2. Text file is **created** to indicated that the acquisition is launched and must be deleted to indicated by the acquisition software once imaging is terminated.



Typical workflow

1. In the main interface select `text file sync` from the dropdown menu and press 'Microscope'
2. In the interface select the text file that you want to see to communicate with the acquisition software.
3. Create the acquisition object by pressing on the corresponding button.
4. You can then close the interface.

Demo

If you want to test alternating sequences of fluidics run and imaging, you could load the demo config for the fluidics. When you launch the full run, you can manually set the sync file back to 0 to indicate that the imaging is completed.

TTL synchronization

Here, image synchronization is obtained with TTL pulses. To achieve this, we use a widely available microcontroller (Arduino). These controllers can send/receive TTL pulses. autoFISH communicates with this controller via the serial port.

- To start acquisition **autoFISH** sends the string '`start`' and will then wait until it receives the string '`finished`'.
- On the Arduino an event loop is running, reading the serial port until the string '`start`' is received. It will then send an output TTL trigger. It will then wait for an input TTL trigger, and once received send the string '`finished`' via the serial port to autoFISH.

Setting up an Arduino is very simple (see instructions for construction of the fluidics system for more details). The code running on the Arduino is here.

Adding your own acquisition module

The code of autofish is modular, and you can add your own acquisition routine. However, this will require some coding. As a guidance, you can already look at how the existing two imaging routines are implemented.

- You will need to add a new Class in **`imager.py`** (using Microscope) as a parent class. As a minimum, this class needs to provide a function called **`acquire_images`**. This function will launch the desired acquisition routine.
- In **`coordinator.py`**, you will need to add your new acquisition class to the function **`run_all_rounds`** to correctly handle the potential input parameters.
- In **`coordinator_gui.py`**, you will need to add you class to the dropdown-menu of the launch GUI (**`make_window_control`**), and then write the GUI for your calls that collects the relevant user inputs.

Setting up the fluidics system

We describe next how you can set up the fluidic system and define a sequence of steps that will be performed automatically. The fluidics system consists of at least a **pump** and the **pipette robo**t, optionally it can further contain a computer-controlled valve and flow sensor

Configuration files

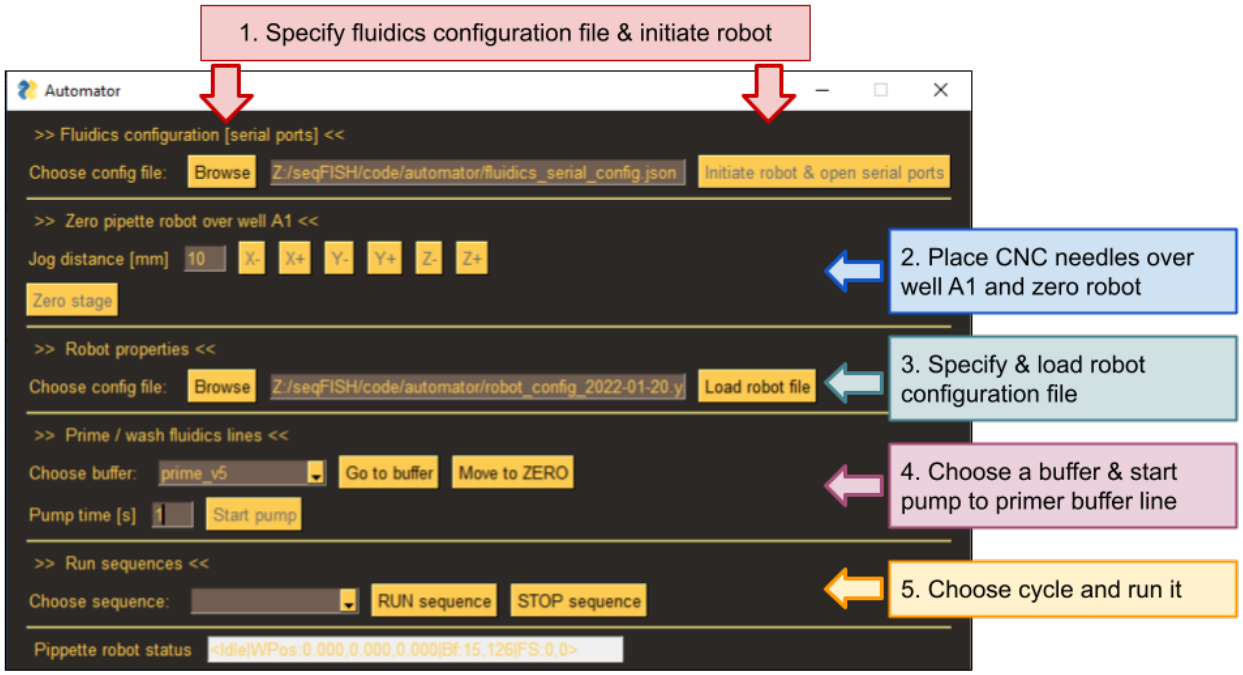
For a user, 2 files are important which define how Python can interact with the different components and how the actual experiment is defined.

- **system_config.json**: specifies how to communicate with the different hardware components (pump, valve, CNC robot). This must be adapted to a given fluidics system only once, and then remains unchanged. For more details see the dedicated section in the Appendix.

- **experiment_config.yaml**: specifies where the buffers are and the sequence of a run. Has to be adapted for each experiment. For more details see dedicated section about experiment specifications.

Basic usage of fluidic system

The fluidics system can be controlled and initiated via the user interface shown below. We next detail a typical workflow, more details on the config files are provided in subsequent sections.



1. **Make sure hardware specifications** stored in ***system_config.json*** are correct.
2. Update **robot configuration** *file robot_config,yaml* for your experiment.
3. Turn on the fluidics system (pump, robot, (if present) valve, (if present) start flow measurement.
4. Start autofish, this will show the GUI.

5. **Initiate system** (Step 1 in GUI, red).
    a. Press `Browse` to select fluidics specification file.
    b. Press `Initiate robot & …` to initiate communication with the different components.
6. **Zero CNC robot over well A**1 (Step 2 in GUI, blue)
    a. Move robot with jog buttons for indicated distance in mm. Center over well A1 (a few mm above)
    IMPORTANT: don't move too much, robot doesn't know the limits of the different motors.
    b. Once needle is centered over A1, press `Zero stage`
7. Load robot specification file (Step 3 in GUI, green). This will verify which buffers are used for the sequential runs, estimate the total run time, and calculate the position of each well. Some information is provided in the terminal, more details in the log file.
8. **Prime buffer lines** (without the chamber attached) (Step 4, purple)
    a. Select the buffer you want to prime from the pulldown menu. Press `Go to buffer`.
    b. Press `Start pump`, which will turn on the pump for indicated duration in seconds. Pump can only be **started once**. To start again, the buffer has to be selected again.
9. **Connect chamber and prime**
    a. Fill with buffer
    b. Chase bubbles.
    c. Mount on support
10. [Optional] If you want to run a selected round, select it from the drop-down menu, and press `RUN sequence` (Step 5, orange)
    a. Progress will be shown in the terminal. A more detailed log is created in a file.
    b. To stop the run, press `STOP sequence`
    Once round is terminated, it will be removed from the dropdown list.

Experiment specification: experiment_config.yaml

This file contains information about the buffers, the sequence of buffer exchange steps and the layout of the plate robot.

- Stored as a yaml file
- Strict formatting: spacings and indents must be respected.
- File contains three sections defining the entire experiment. Each section starts with the respective keyword followed by a ". Elements of each section have an indent of 4 spaces.
    - **buffers:** name and position of all buffers.
    - **sequence:** sequence of steps the robot has to perform for each run.
    - **well_plate:** properties of the plate robot.

buffers

Here, the names and position of all buffers are specified. **Two types of buffers** can be specified:

- **General purpose buffers**: they are the same for each round. Typically stored in syringes.
- **Hybridization round specific buffers**: they change during each round.

    - These are typically buffer with readout oligos, or specific wash buffers. Several such buffers can be defined for a round.
    - The buffers are declared in the 'sequence' section and terminate with an **ii**, e.g., `w_ii`. In the buffer list, they are specified with a unique name for each round, e.g., `w_r1` for round `r1`.
    - **At least one** hybridization specific buffer must be specified. Autofish analyses the settings file and then automatically determines the round identifies.
- **Runs are listed and executed in the order the buffer list is provided.**

Names

- Each name must be **unique**.
- Round-specific buffers are indicated with an **ii** at the end of their name.
    - They must be unique as well, e.g. using `w_ii` and `w_stringent_ii` will lead to conflicts since the string w_ can be matched to both buffers.
- Please note that after the name, you have to add ":" and then the position.

Examples

`buffer: wash`	general purpose buffer "wash"
`buffer: w_r1`	round specific buffer `w_` for `r1` (is defined as `w_ii` in the protocol sequence)
`buffer: h_r1`	round specific buffer `h_` for `r1` (is defined as `h_ii in` the protocol sequence)
`buffer: w_r1`	round specific buffer `w_` for `r2` (is defined as `w_ii in` the protocol sequence)

Positions

Positions are provided with a tuple with 3 elements:

[valve-id, plate-id, plate-pos]

- **valve-id**: which port of computer-controlled valve, e.g., with 8 entries.
    - 0: no valve used (because no valve in fluidics system)
    - >0: port entry number
- **plate-id**: several plates could be put on the robot
    - 0: no plate -> position as to be specified with XYZ coordinates (see below)
    - 1: plate 1 -> position must be specified with the well number (A1, …)
- **plate-pos**:
    - For absolute positions you must specify the position relative to 0 position when homing the robot. To go to position (X=135mm, Y=36mm, Z=-39mm), you must specify `X135_Y36_Z-39.` Important are the underscores, and the minus (-) for the Z coordinate since the robot will move down.
    - For the well position, you can use A1, A2, … The program calculated the absolute position based on the well_plate specification that you provided.

Note: For buffers on the syringe stand, plate-id and plate-pos must be set to null.

Example

We defined two buffers, the general buffer `wash_valve4` which can be found on the valve 4, and the buffer `w_r1`, which will be used in an iterative run which is on the first 96 well plate, on valve 6 in well A1:

```
buffers:
    wash_valve4: [4,null,null]
    w_r1: [6,1,A1]
```

valve_out

This is an optional field to specify outlet valves. We use these valves to direct the flow to different samples, e.g. different channels on an Ibidi multi-channel slide. The provided list contains the position IDs of each valve. In the example below, an outlet valve with 3 positions (5,6,7) is used. The

```
valve_out:
    positions: [5,6,7]
```

sequence

Here, the sequence of steps to be executed is specified. Each step starts with a dash and the steps are executed in the order listed from top to bottom. The following steps are supported.

Command	Action	Example
`buffer`	Select the specified buffer. Buffer names are listed with their position in the dedicated section “buffer” explained below. Buffer names containing ‘ii’ are buffers that are hybridization round specific and can be selected (more below).	`- buffer: wash_valve4`
`pump`	Activate the pump for the indicated duration in seconds.	`- pump: 180`
`pump_valve_out`	Activate the pump for the indicated duration for each of the specified outlet valves. Requires the definition of `valve_out` (see above). Number of parameters of durations must match the number of specified valve positions.	`- pump_valve_out: [180,30,30]`
`pause`	Take a break for the specified duration in seconds.	`- pause: 1200`
`round`	Specify **conditional** steps that are performed only for the indicated identifiers for a round (e.g. to perform a DAPI stain). Note that command has an additional indent. Multiple rounds can be defined and must be listed separated by a “,” and nothing else.	`- - round: r1,r4` `  - buffer: dapi_ii`
`zero_plate`	Moves plate to position 0. This can be useful if during an experiment a plate should be changed (can be combined with `wait` to stop the system)	`- zero_plate:`
`wait`	Waits for the user to press ENTER. Can be used if some changes must be done to the system.	`- wait:`

Example 1 : select the buffer `wash` and the activate the pump for 90s, you can write:

```
sequence:
    - buffer: wash
    - pump: 180
```

Example 2 : the system has three outlet valves. The following command will select the first valve and activate the pump for 180s, then the second and third valve with a pump duration of 30s, respectively. The longer duration for the first valve allows to fill the entire system, while the subsequent durations are shorter since the buffer has to fill only the lines from the outlet valve to the specimen.

```
sequence:
    - pump_valve_out: [180,30,30]
```

Conditional steps

You can define conditional steps, which will be only executed for a certain hybridization run. For example, you want to include a DAPI wash for the first round. For this you have to use an indented sequence as shown below, where the first action is “round” followed by the identifier of the round, where these steps are to be performed. Please also note how the dashes are used. You can use all actions detailed above, including round-specific buffers, which will then only be used for these conditional runs.

In the **example** below, an additional buffer (`dapi_ii`) is used for rounds r1 and r4.

```
  - - round: r1,r4
    - buffer: dapi_ii
    - pump: 180
    - pause: 180
```

well_plate

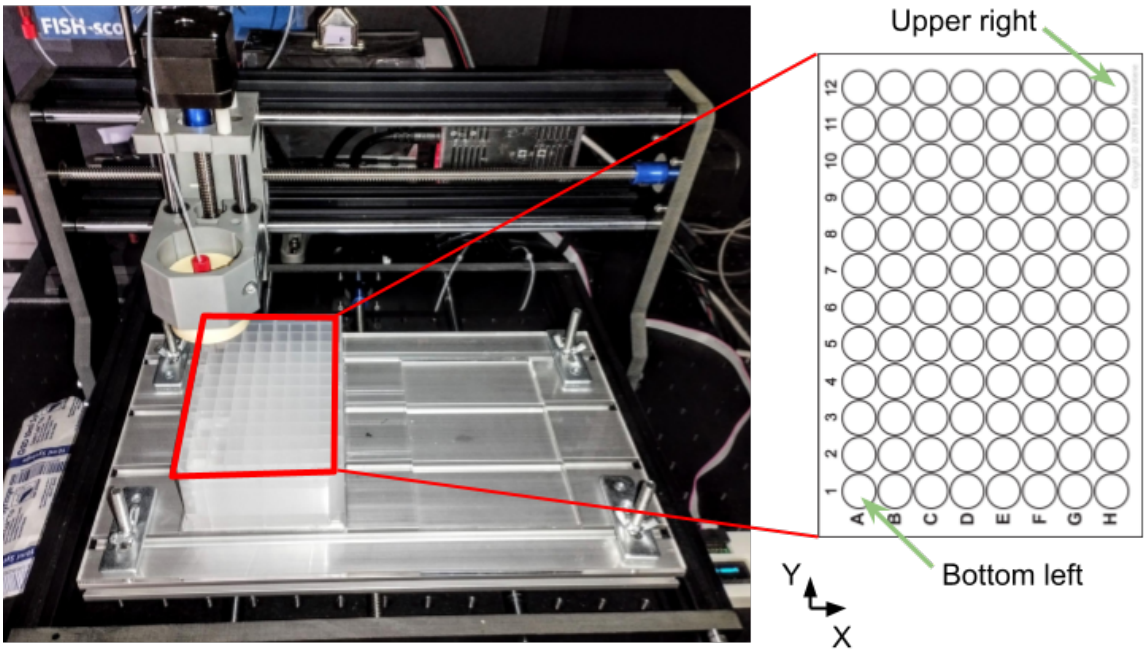
Here the specifics of the used well plate are specified. This usually has to be done once for a given configuration.

Command	Action	Default
`trop_right`	Position of upper right well (center)	`x: 63` `y: 99`
`bottom_left`	Position of bottom left well (center)	`x: 0` `y: 0`
`columns`	Number of columns	`8`
`rows`	Number of rows	`12`
`well_spacing`	Distance between well	`9`
`z_base`	How much robot will go down in z [mm]	`-37`

~~feed~~	~~Speed at which robot moves [not implemented yet]~~	~~500~~

Information will be used to automatically calculate the positions of each well (this can also include a tilt, i.e. a plate that is not aligned with the X, Y axis of the robot). The assumed orientation of the plate is indicated in the figure below. This means

- Columns are letters (A-G for a 96 well plate)
- Rows are numbered (1-12 for a 96 well plate)



Flow sensor

As an optional step, flow rates can be measured, and compared to the expected flow when the pump is active. By defining tolerance limits, the fluidics system can be automatically stopped, when the measured flow is outside of this tolerance.

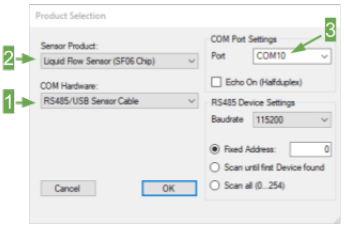
Sensirion SLF3x Flow Sensor

We use a flow sensor from Sensirion (SO-SLF3S-0600F-EVAL-KIT). The evaluation kit connects it directly to a USB port. Rather than reading the flowrates directly into Python, we use the provided Viewer software available on the Sensirion website. This software allows you to log the flow rates into a CSV file. In autofish, we read this csv file and extract the expected volume for the specified pump duration.

In the system config file, you then must specify the location of this log file. Also, before activating the pump, make sure that you record flow into the file. The image below shows how to start the viewer software and start logging to a file (in this example the flow sensor is on COM10):

1. Start software “USB RS485 Sensor Viewer”

2. In interface, select (order is important!):
   a. COM Hardware: **`RS485/USB Sensor Cable`**
   b. Sensor Product: **`Liquid Flow Sensor (SF06 Chip)`**
   c. COM port: **`COM10`**



3. In new interface, start logging by
   a. Measurement control
      i. Sampling interval: **`50ms`**
      ii. To start measurement, press button **`Run`**
   b. Data logging
      i. Select file: **`C:\Temp\DataLog.csv`** (default)
      ii. To start logging, press button **`Start logging`**

Adding new components

The Python code is modular and allows adding new components.

Fluidics system

Communication with the fluidic system is handled by automator.py. Here, the class **`Robot`** controls the fluidics system, e.g. performing sequence of buffer exchange steps.

Then each component is specified with its own class.

- `flowsensor`
- `plateController`
- `pumpController`
- `valveController`

Each base class defines the **required methods** that have be available. Each new component is then implemented as a new implementation of this class, e.g. for the implementation of the Longer peristaltic pump, we created the class `LongerBT100` derived from the `pumpController` class.

Once a new class is implemented, the corresponding function `assign_XYZ` in the Robot class has to be amended to include the new component, where `XYZ` is either `sensor`, `valve`, `pump`, `plate`. Here, depending of the name used of this class in the `system_config` file, the initiation of the object is defined. For instance,

- The longer pump is specified in system config file as type ‘LONGER BT100’.
- in the `assign_pump` class, the initiation of the pump object is performed with the `LongerBT100` class when the type corresponds to ‘LONGER BT100’

Microscope communication

We implemented the class `Microscope` that has one required method '`aquire_image`'. Different classes can be derived from this class to initiate communication with the microscope. For instance, `pycroManager` to interact with Micromanager, or `TTL_sync` to initiate synchronization via a TTL pulse.

If new classes are implemented, they also must be considered in the GUI to be avaiable in the pull-down menu (function `autofish_gui.py`).

Appendix

System configuration: system_config.json

In this json file, the communication protocol with the different hardware components is specified. Currently four different components are permitted, for each one a parent Python class is defined in autofish, specifying the minimal requirements for required methods. Then a child class can be defined for a specific hardware.

- **`pump`**: defines a class for a pump. It must support at least these methods:
  - **`start`**: start the pump. /Users/fmueller/Library/CloudStorage/OneDrive-InstitutPasteurParis/code/GitHub/development/fish-quant.github.io
  - **`stop`**: stop the pump.
- **`valve_in:`** computer-controlled valve. Used to selected buffer (input valve)
- **`valve_out:`** computer-controlled valve. Used to selected different outputs (output valve)
- **`plate`**: plate robot permitting to access many buffers, either in 96 well plates or in larger reservoirs.
- **`flow_sensor`**: sensor to measure flow rates.

The only required field is **`'type'`**, permitting to identify which exact component is connected. All other fields are component specific. For instance, for components connected via the serial port, information about the COM port and serial port protocol (baud rate) is provided.

Below, the configuration file for our fluidics system is shown:

```
{
    "pump": {
        "test_command": "1#\r",
        "response": "REGLO DIGITAL",
        "type": "REGLO DIGITAL",
        "baudrate": 9600,
        "COM": "COM10",
        "Revolution": "CCW",
        "Flowrate": 0.5
    },
    "valve_in": {
        "test_command": "/1h30001R\r",
        "response": "/0",
        "type": "HAMILTON MVP",
```

```
        "baudrate": 9600,
        "COM": "COM11"
    },
    "plate": {
        "wake_up": "\r\n\r\n",
        "test_command": "$I\n",
        "response": "VER:",
        "type": "CNCRouter3018PRO",
        "baudrate": 115200,
        "COM": "COM8"
    },
    "flow_sensor": {
        "type": "Sensirion_csv",
        "log_file": "C:\\Temp\\DataLog.csv",
        "kernel_size": 20,
        "flow_min": 20
    }
}
```

Serial port communication

- https://pythonhosted.org/pyserial/
- https://github.com/pyserial/pyserial
- https://www.com-port-monitoring.com/ (be careful to download the free version)

References

Boettiger AN, Bintu B, Moffitt JR, Wang S, Beliveau BJ, Fudenberg G, Imakaev M, Mirny LA, Wu C, Zhuang X. 2016. Super-resolution imaging reveals distinct chromatin folding for different epigenetic states. *Nature* **529**:418–422. doi:10.1038/nature16496

Edelstein AD, Tsuchida MA, Amodaj N, Pinkard H, Vale RD, Stuurman N. 2014. Advanced methods of microscope control using μManager software. *J Biol Methods* **1**:e10. doi:10.14440/jbm.2014.36

Moffitt JR, Zhuang X. 2016. RNA Imaging with Multiplexed Error-Robust Fluorescence In Situ Hybridization (MERFISH). *Methods Enzymol* **572**:1–49. doi:10.1016/bs.mie.2016.03.020

Pinkard H, Stuurman N, Ivanov IE, Anthony NM, Ouyang W, Li B, Yang B, Tsuchida MA, Chhun B, Zhang G, Mei R, Anderson M, Shepherd DP, Hunt-Isaak I, Dunn RL, Jahr W, Kato S, Royer LA, Thiagarajah JR, Eliceiri KW, Lundberg E, Mehta SB, Waller L. 2021. Pycro-Manager: open-source software for customized and reproducible microscope control. *Nat Methods* **18**:226–228. doi:10.1038/s41592-021-01087-6