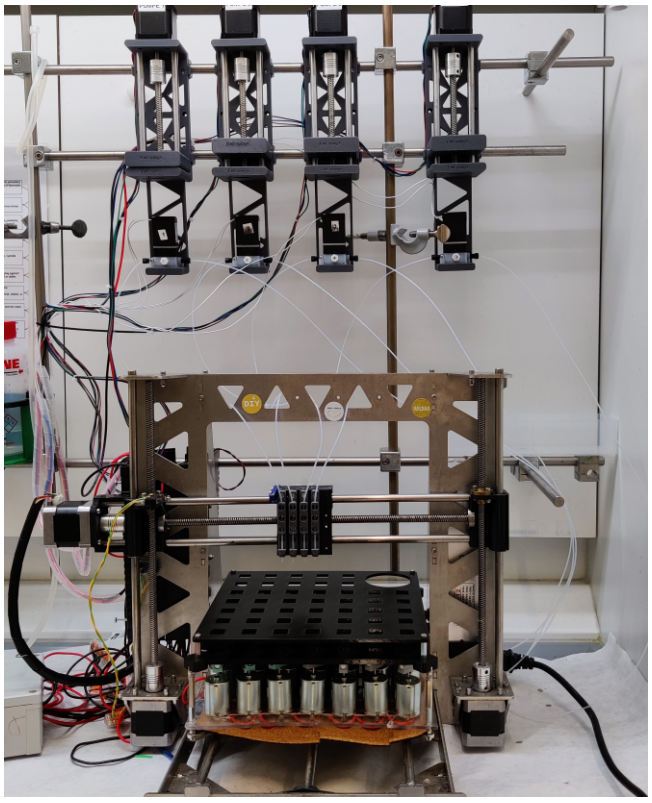
# Guide to the Automatic Synthesis Machine

Jakob Saugbjerg, Thorbjørn Jensen

31/08, 2023

Lahn automation group AU



# 1 Introduction

The Automatic Synthesis Machine (ASM) is designed for automated batch synthesis work and other tasks involving mixing of liquids and solutions. The ASM consists of four main parts that needs preparation before use.

- A 3D printer framework
- A dosage module
- A module for holding the reaction vessels
- A stirring module

The different parts have been designed to be assembled with ease and function in modules, allowing each part to be prepared independently. Four modules have been predeveloped; a stirring module, a combined atmosphere control and heating module, and a dosage module, with CAD drawings available on our GitHub (see each folder for a short description of each module[1]).

A GUI has been developed in Python to work as the interface for the ASM. The GUI comes with predefined setup files, but can easily be expanded
The GUI serves both as the program operating the ASM, but also as the program for protocol creation. The protocol creation is predefined with methods of dosing; a standard method where the desired are dosed in the connected order one vial at the time, a project specific method called NEST_standard_method that doses the three first solution in all vials, followed by dosing of a fourth solution, a calibration method designed for calibration of pumps and a custom order dosing that allows the user to choose the desired order of solutions as well as delays between solution dosing. Additionally, a range of different modules for holding reaction vessels has been created with accompanying coordinate files (see section 2 for description of the files and how to create your own).

# 2 Calibration of Machine

For the ASM to function as intended, the machine should be calibrated in two parts.

1. Coordinate calibration

[1] https://github.com/AUautosynth/Auto-Synthesis/tree/Release-version

2. Dosage calibration

**1.** The calibration of the coordinates for a new reaction module is fairly easy. It requires the open source software Pronterface to control the machine. Place a needle or another narrow object in the holder (see section XX for dosage module) and connect to the ASM through Pronterface. Now, move the ASM (G28 for homing and G1 for movements[2]) to the coordinates for the first reaction vessel and note the center position of said vessel. If the reaction module is symmetric or with constant spacing between vessels, the remaining coordinates can be calculated through the spacing (coodinates are given in millimeters from origin). **Be sure to determine the waste position as well**. If, however, the reaction module is uneven or asymmetric, it is recommended to confirm the coordinates for each vessel and note it. The module file is simply a text file with vital information for the module functionality. An example is included in Figure 2.1.

| | |
|---|---|
| vessel volume: | 3 mL |
| number of vessels: | 45 |
| waste coordinates(x,y): | 180,180 |
| x-coordinate | y-coordinate |
| 199 | 16 |
| 171 | 16 |
| 142 | 16 |
| 113 | 16 |
| 84 | 16 |
| 56 | 16 |
| 28 | 16 |
| 199 | 44 |
| ... | ... |
| 28 | 158 |
| 142 | 186 |
| 113 | 186 |
| 84 | 186 |
| 56 | 186 |
| 28 | 186 |

**Fig. 2.1:** Example of a module text file containing all necessary information regarding the reaction module.

**If creating a new module file,** the file should contain two tab separated columns with the same information as shown in Figure 2.1.

**2.** Using a module with a confirmed coordinate set, the ASM program includes a calibration method to develop pump characteristics to save. This should be done when introducing a new syringe size and when working with liquids with significantly different properties from the liquids used in existing calibrations. To run the calibration method, a new protocol should be created. **1)** Open the ASM program and press create. **2)** Choose the **pump_calibration** method, set the number of solutions to **1**, choose the reaction module used for the calibration

[2]https://marlinfw.org/docs/gcode/G000-G001.html

(we recommend a module that holds fairly large vessels, say 5 mL snapcap vials) set the number of replicates to **3** for statistical significance, and choose the number of different data points desired with the number of experiments. **3)**

# 3 Running experiments

To run experiments, the ASM program serves as central platform. First, an experimental protocol should be created with the desired experimental values. **1)** To do this, open the program and press create. **2)** choose the desired dosing method, the number of solution needed for the experiment, the chosen reaction module, the number of replicates desired for each experiments and number of experiments wanted (Figure 3.2, left). **3)** Pressing the "Confirm Setup" button, the program takes you to the next page for setting the volume, the pump characteristic, the pump speed, whether the solution is viscous or not, set the waste coordinates for each solution, and the gantry position.

The gantry position is the position of each needle on the x-axis carriage (see Figure 3.1). Depending on what type of needle fixture chosen, the gantry position is essential for each solution being dosed correctly. Changing the gantry position does not affect the dosing order or solution numbering.

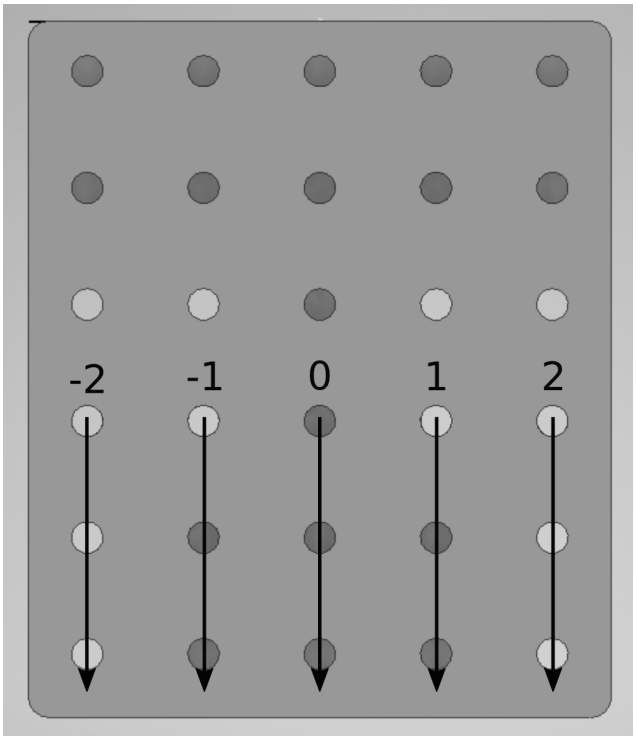



**Fig. 3.1:** Indication of the different gantry positions on the x-axis carriage. Each number indicates the position to put into the GUI when creating a protocol.

The waste coordinates are default set as waste coordinates for the chosen module, but in case a

precious solution should not be mixed into the general waste, the coordinate set for a reaction vessel can be put in (Figure 3.2, right).

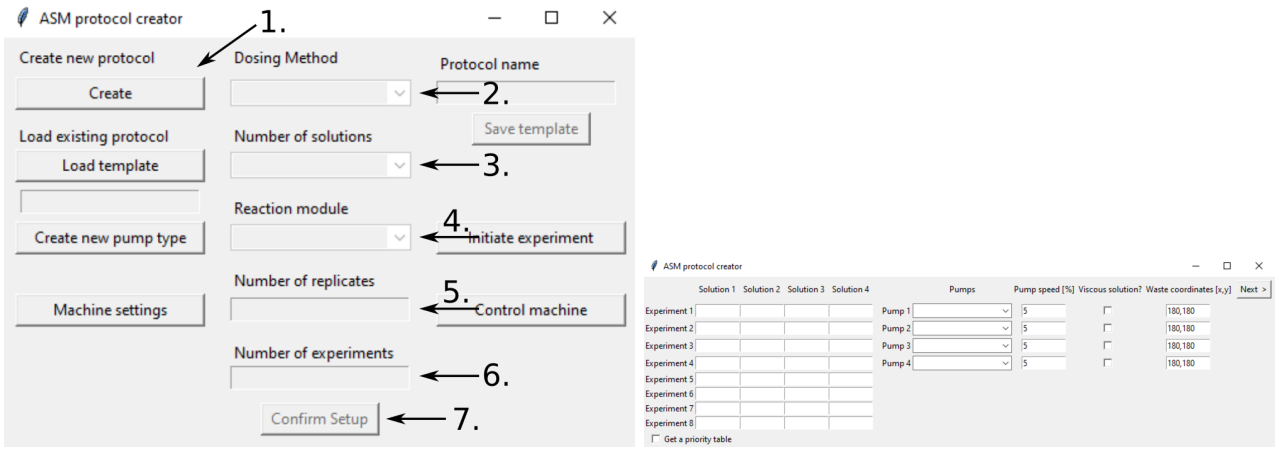



**Fig. 3.2**

When filling out the experimental data (see Figure 3.2, right), each experiment should be given a volume for each solution. If none of a solution is needed for an experiment, the entry field can be left empty or filled with a 0. Once filled completely, press the button "Next ". This closes the window. The experimental data is now created as variables internally, and the protocol can be saved using the empty text field and pressing "Save Protocol".

To initiate the experiment, press the "Load Protocol" button in the main program and find the .txt file containing the desired protocol. Then press the "Initiate Experiment" button, which will bring you to the experimental window (Figure 3.3).


**Fig. 3.3**

Now connect the correct solution to the pump designated to said solution, and attach the syringe in the correct to the pump.

From here, the only thing left to do is choose the correct USB port, turn on the ASM, randomize the experimental order (using the "Randomize position" button) and press the "Start

Experiment" button. From here, the experiment should run itself.

# 4 Module overview and assembly

The following section contains a brief overview of the different modules predeveloped for the ASM, including an assembly guide for critical parts.

## 4.1 Dosage module

The dosage module contains three main parts; **1)** a pump with a connecting valve (Figure 4.1, left). The pump fits all syringe sizes (Braun two component green ones). **2)** A x-axis carriage that holds a needle fixture (Figure 4.1, right), and **3)** a needle fixture making sure the needles are held in place at a constant position throughout experiments (Figure 4.1, right). The x-axis carriage is designed so that different versions of needle fixtures can be mounted. Technical drawings of all three parts can be found at the GitHub page.

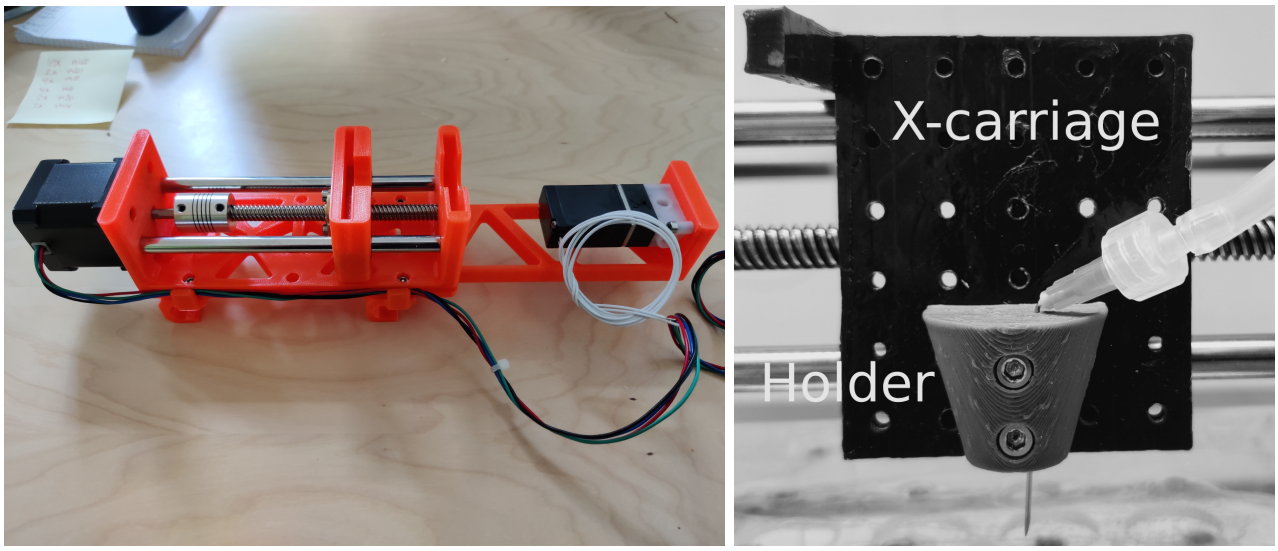
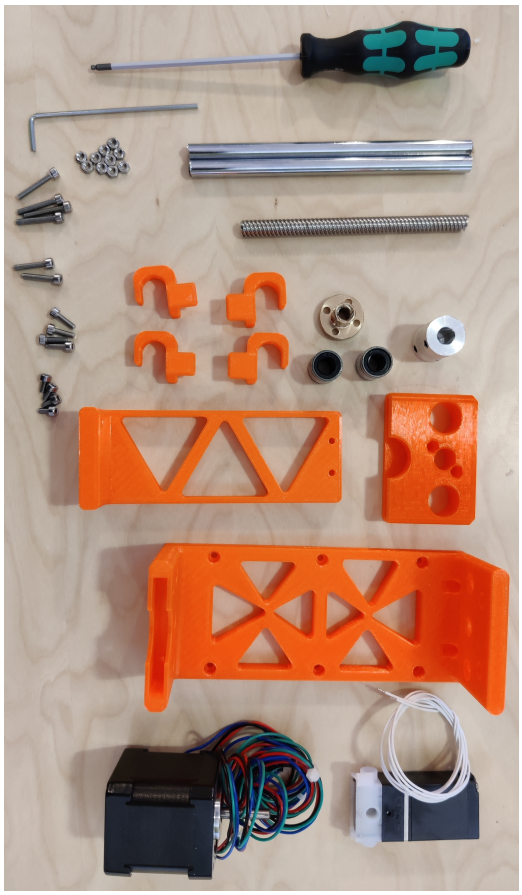



**Fig. 4.1:** The figure shows the assembled pump with the attached valve (left) and the x-axis carriage with an attached needle fixture (right)

### 4.1.1 Pump assembly

To assemble the pump, the following items are needed:



- 3D printed pump kit
- 2 140x8 mm rods
- 1 M8 lead screw (115 mm) and 1 brass 8 mm lead screw nut
- 2 LM8SUU ball bearings
- 1 flexible coupling - 5 mm to 8 mm
- 10x 3 mm nuts
- 4x M20 3 mm bolts
- 2x M16 3 mm bolts
- 4x M10 3 mm bolts
- 4x M6 3 mm bolts
- A 2 mm and a 2.5 mm hex key
- A NEMA 17 stepper motor
- Burkert 6606 valve (ETFE inner)

**1)** Insert two 3 mm nuts into the pump sledge and install the lead screw nut with the bolt holes on top of the sledge, using 2 M20 screws (Figure 4.2)

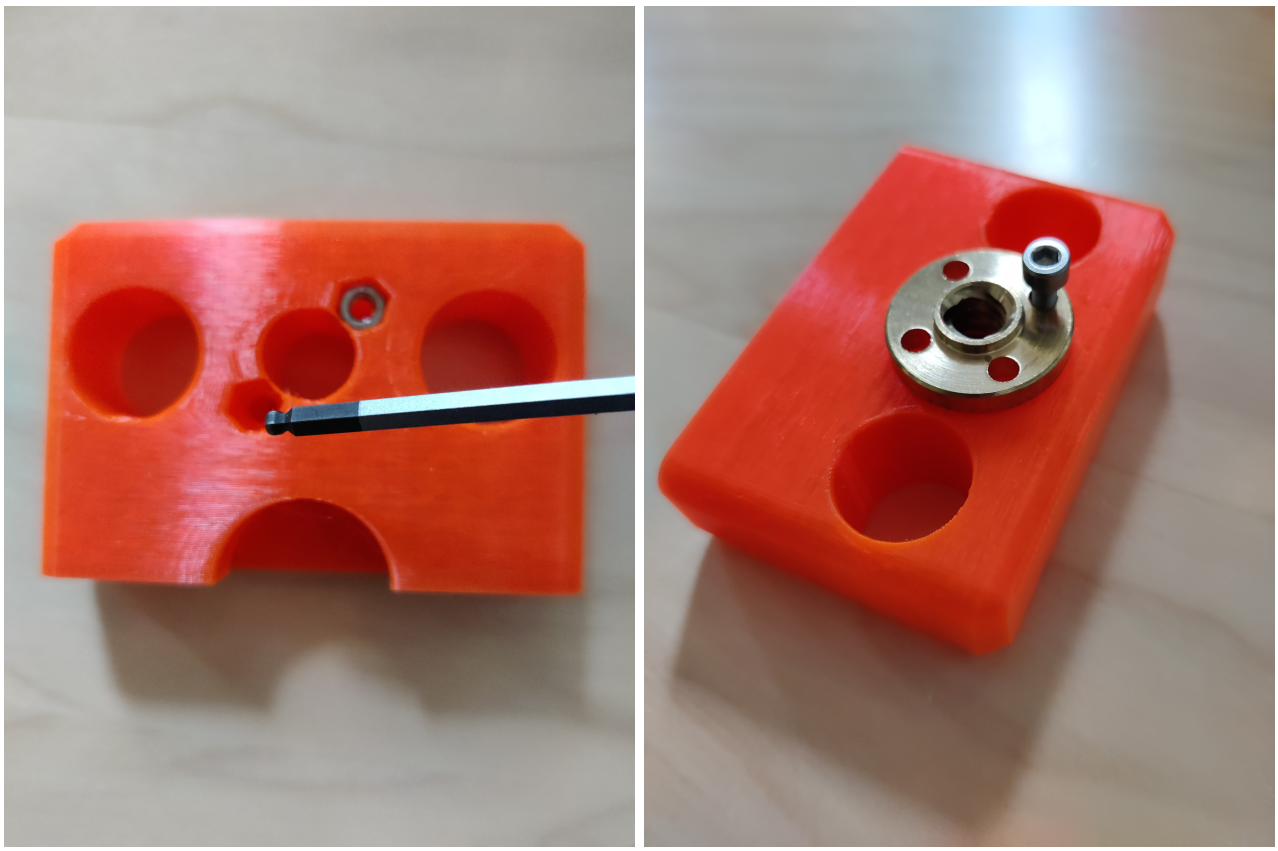


**Fig. 4.2**

**2)** Press the LM8SUU ball bearings into the sledge (Figure 4.3), from the side opposite the lead screw nut. Note; this is a tight fit, force can be necessary.

**Fig. 4.3**

**3)** Connect the flexible connector to the M8 lead screw. Some adjustment of the connector might be needed to make sure the lead screw fit correctly in the final assembly. Be sure to fit the connector evenly to avoid bending. Screw the lead screw into the pump sledge from the lead screw nut side (Figure 4.4).

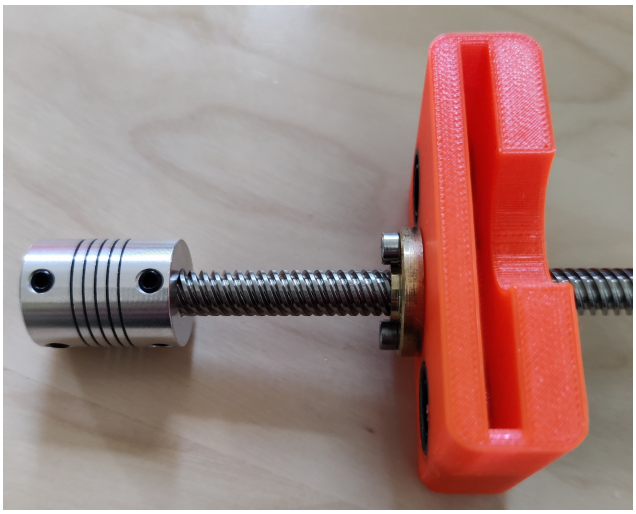


**Fig. 4.4**

**4)** Grab the pump body and press the 8 mm smooth rods half way into the pump body from the top (Figure 4.5). This is a tight fit, it might be necessary to twist the rod into place.

**Fig. 4.5**

**5)** Place the pump sledge with the attached lead screw into the pump body (Figure 4.6). It

might be needed to screw the lead screw a bit out in order to make the sledge fit with the smooth rods. Press the smooth rods completely down.

**Fig. 4.6**

**6)** Attach the motor to the top of the pump body. Make sure that the wires protrude in the direction of the ridge in the pump body (Figure 4.7, left). Screw the motor tight to the pump body with the 4 M6 screws (Figure 4.7, middle). Now fix the motor shaft in the flexible lead screw connector. Make sure that the flat part of the shaft aligns with one of the connector screws (Figure 4.7, right). You should now be able to rotate the flexible connector, causing the sledge to move up and down.

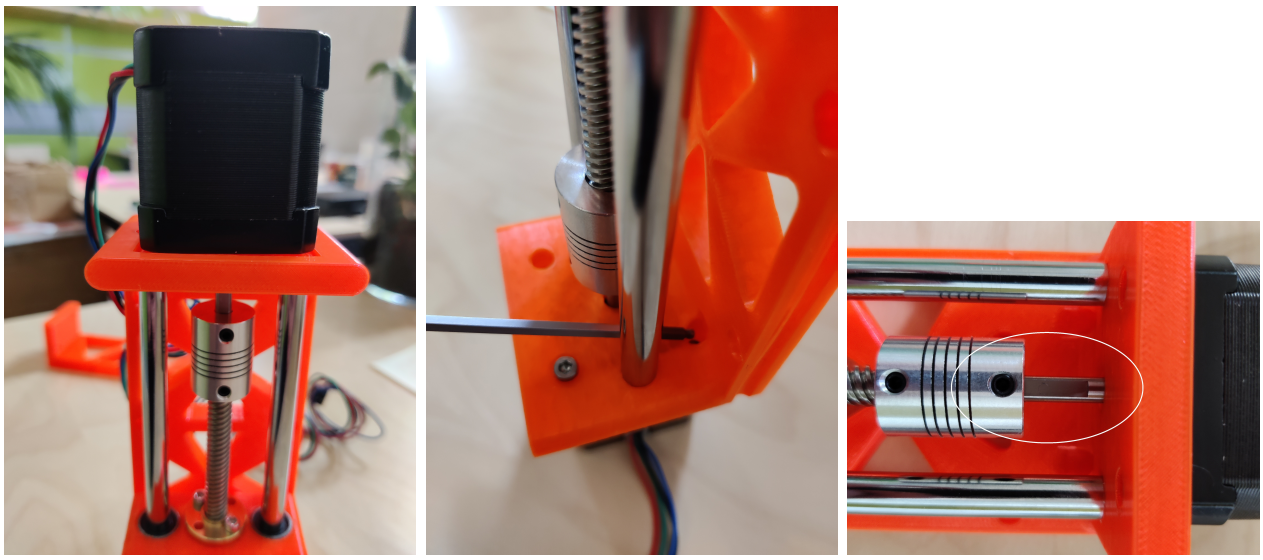


**Fig. 4.7**

**7)** Take the four hooks from the pump kit and push in the 3 mm nuts into the hexagonal holes (Figure 4.8, left). Install the hooks on the pump body in the top and bottom holes on the pump body, using the 4 M10 screws. The hooks should be installed so that the pump can hang with the motor facing upwards. When installing the left hooks, make sure to put the wires from the motor into the pump body ridge and fixate the wires with the hooks (Figure 4.8, right).

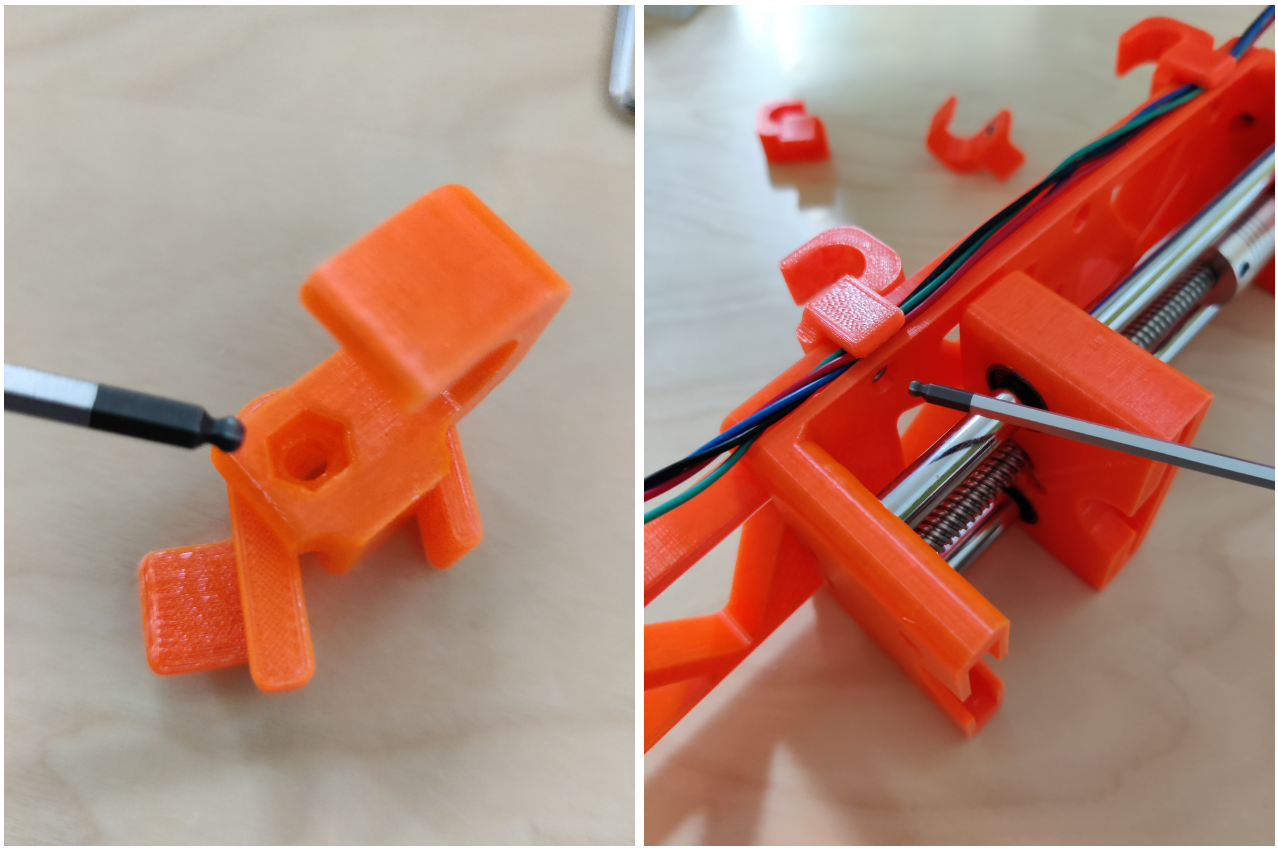


**Fig. 4.8**

**8)** In the bottom of the pump body, push in two 3 mm nuts into the small holes (Figure 4.9, left). Now turn your attention to the valve fitter. Push in two 3 mm nuts into the hexagonal

holes in the valve fitter (Figure 4.9, right).

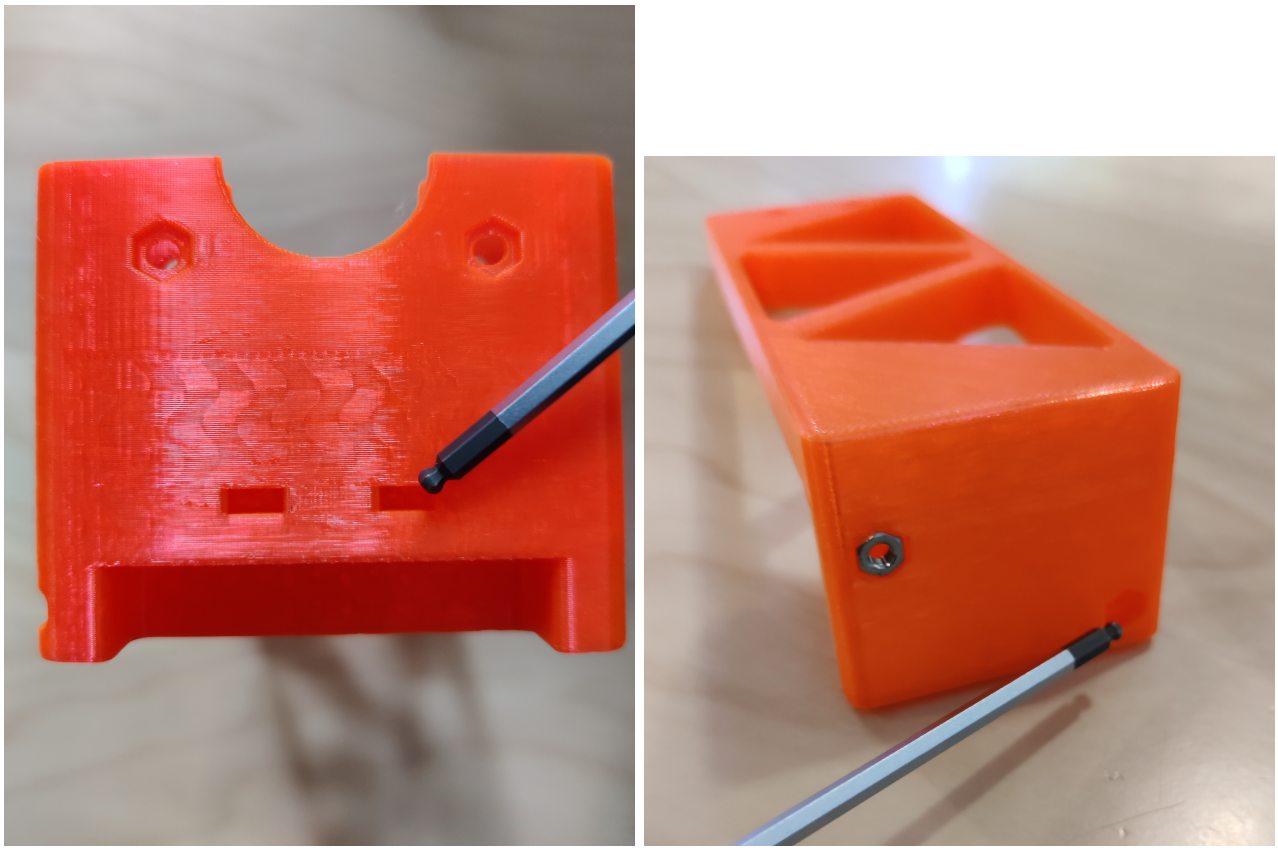


**Fig. 4.9**

**9)** Now attach the valve fitter to the indent in the pump body using the remaining two M20 screws (Figure 4.10, left). Finally, install the valve to the valve fitter using the two M16 screws (Figure 4.10, right). All done!

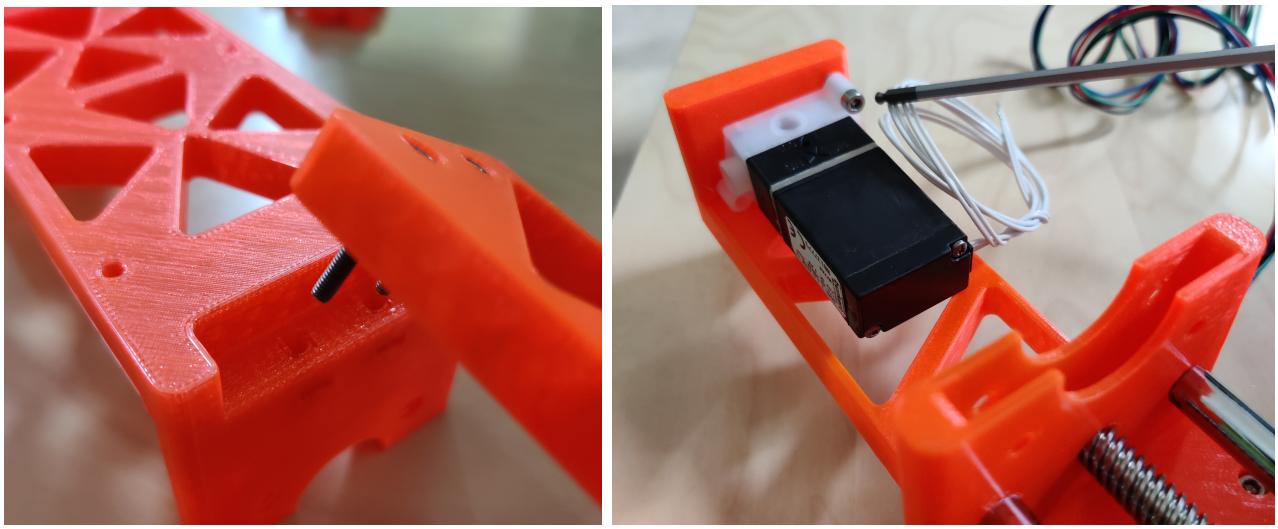


**Fig. 4.10**

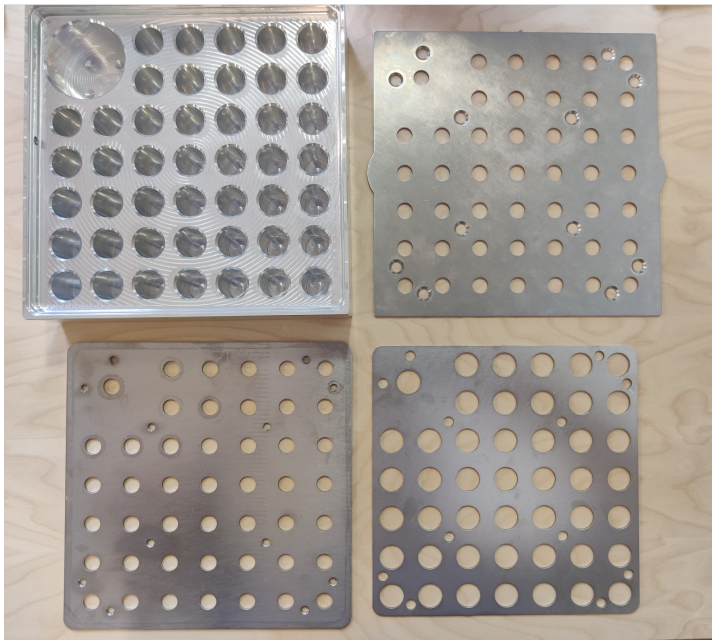
For the pump to operate, syringe fitters needs to be 3D printed as well. The fitters comes in pairs of two, with one part fitting into the pump sledge and one part fitting into the pump body. All designed fittings are available at the GitHub page.

## 4.2 Heating and atmosphere control module

The heating and atmosphere control (HAC) module contain four parts to function as intended. **1)** An aluminum block designed to hold up to 45 snap cap vials (5 mL) and a waste container, **2)** a stainless steel lid with holes aligned with the placement of each vial and waste. The lid comes in three parts, a bottom, a middle and a top piece, all connected with slanted screws, **3)** small silicone discs and a silicone rim that fits into the lid holes, ensuring a sealed module, and **4)** a 220x220 mm 220 V silicone heating pad, mountable on the bottom of the aluminum block.

### 4.2.1 HAC module assembly

For the HAC module, the following parts are required:



- 1 stainless steel lid
- 1 aluminum block
- 12 slanted screws
- 46 silicone discs*
- 1 silicone rim*
- 1 200x200 mm silicone heating mat (220 V, 200 W)

*Description on how to make the silicone discs and rim are available in Section 4.2.2

**1)** The lid is assembled by placing the middle section on the bottom section, aligning the screw holes (Figure 4.11, left). The silicone discs are placed in the middle section and the silicone rim placed along the edge of the middle section (Figure 4.11, right).

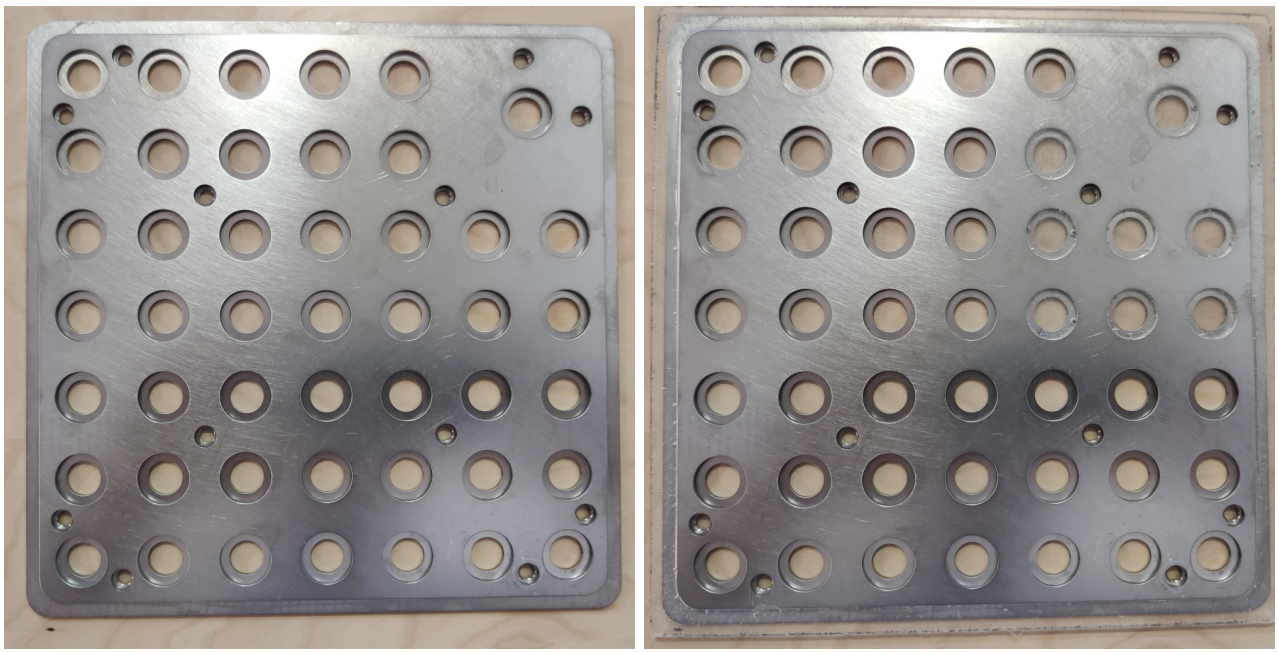


**Fig. 4.11**

**2)** The top part of the lid is placed on the middle section, again aligning the holes. Each screw is fastened, making sure not to over tighten (Figure 4.12, left)! On the bottom of the aluminum block, the heating mat is installed by removing the adhesive cover and attaching the mat precisely. Make sure the wires are protruding through the port in the bottom (Figure 4.12, right).

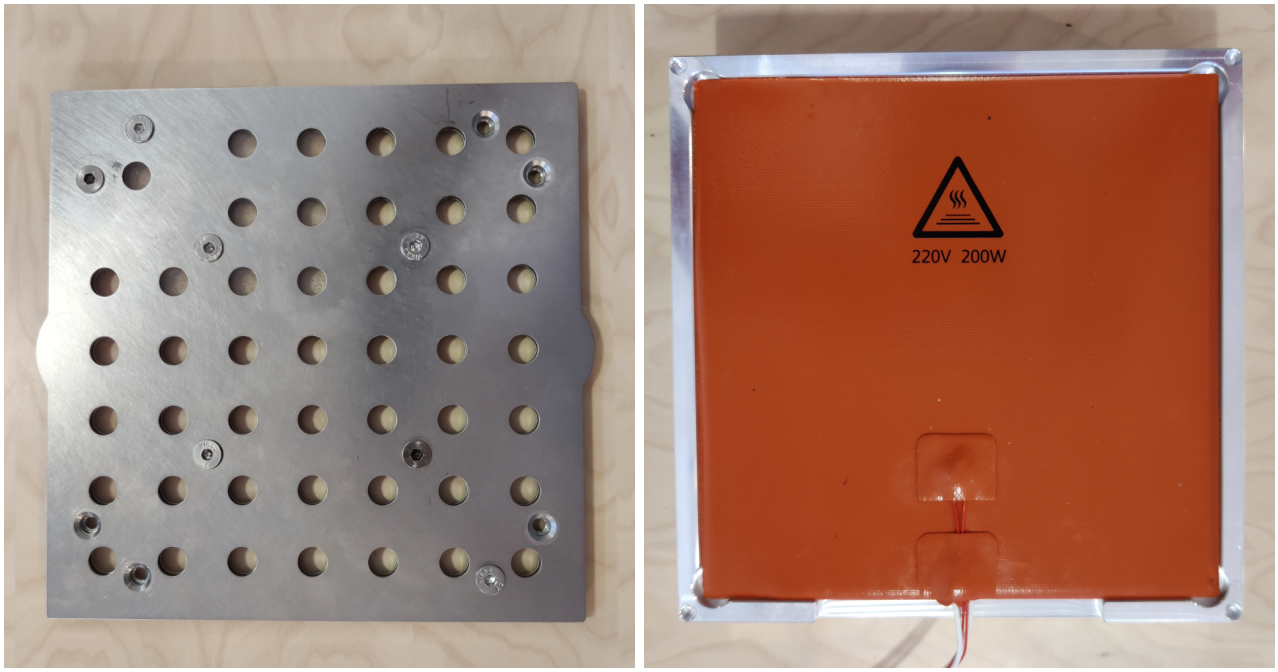



**Fig. 4.12**

This finishes the HAC module assembly. The HAC module comes with small features for convenience. **1)** on the back of the module a hole is found for a 1/4" thread, allowing a tube fitting to be installed for easy $N_2$ inlet (Figure 4.13, black circle). **2)** an additional 1/4" thread

hole is found on the back piece, going into the aluminum block itself (Figure 4.13, red circle). This allow for installment of a thermocouple for temperature monitoring. **3)** on each side, two 3 mm holes are found, allowing a plastic (or similar) screen to be installed (Figure 4.13, yellow circles).

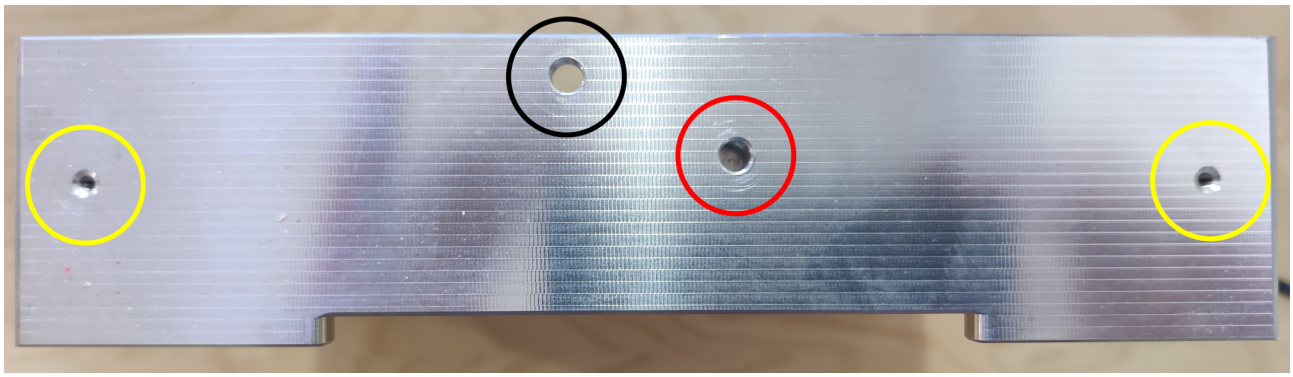


**Fig. 4.13**

### 4.2.2 Silicone sealing

I order to finish the HAC module, silicone discs and and an edge rim should be molded. Luckily, the HAC module has an accompanying mold for exactly this purpose. First, mix your favorite silicone in the color of your choice (Figure 4.14, left, we chose white). We do not have a specific shore value to aim for, but aim above 30 shore (shore A). Proceed to pull vacuum on the mixed silicone to get rid of the worst air bubbles (Figure 4.14, right).

**Fig. 4.14**

When satisfied with the amount of air removed from the silicone mix, proceed to pour the mix onto the mold (Figure 4.15, left). Take a flat piece of plastic, wood or metal and distribute the silicone in each hole, as well as the ridge along the edge (Figure 4.15, right). Leave to cure

and pop each small disc out as well as the edge piece. Now everything needed to seal the HAC module is done.

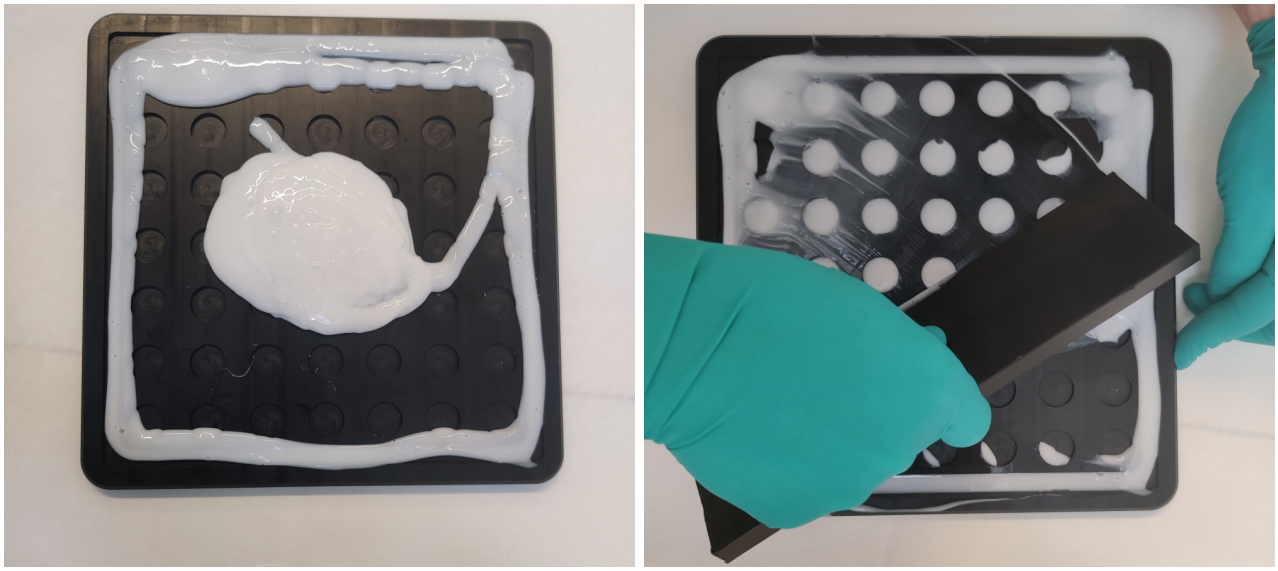
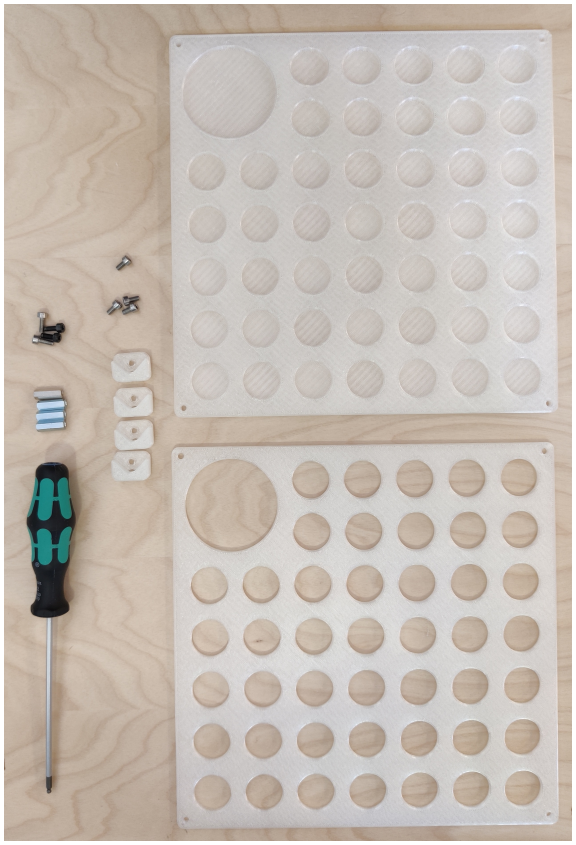


**Fig. 4.15**

## 4.3 Generic module

Additionally, a generic module was developed for convenience. The module functions as a tray for the desired reaction vessel and fits on the ASM reaction bed. The reason for calling it generic, is that the module is 3D printable and the shape of the hole in the 3D file is easily modifiable, thus making it vessel generic in nature. The tray measures 200x200 mm in order to be printed on most 3D printers, and is fitted with corner pieces in order to fit on the 220x220 mm frame of the ASM.

For the generic module, the following is needed:



- 1 module top
- 1 module bottom
- 4 corner pieces
- 4 M6 3 mm screws
- 4 M8 3 mm screws
- 4 15 mm screw sleeve spacers

In order to assemble the module, fit the four M8 screws into the corner pieces (Figure 4.16, left). Attach the corner pieces to the module bottom in each corner, using the screw sleeve spacers (Figure 4.16, right).

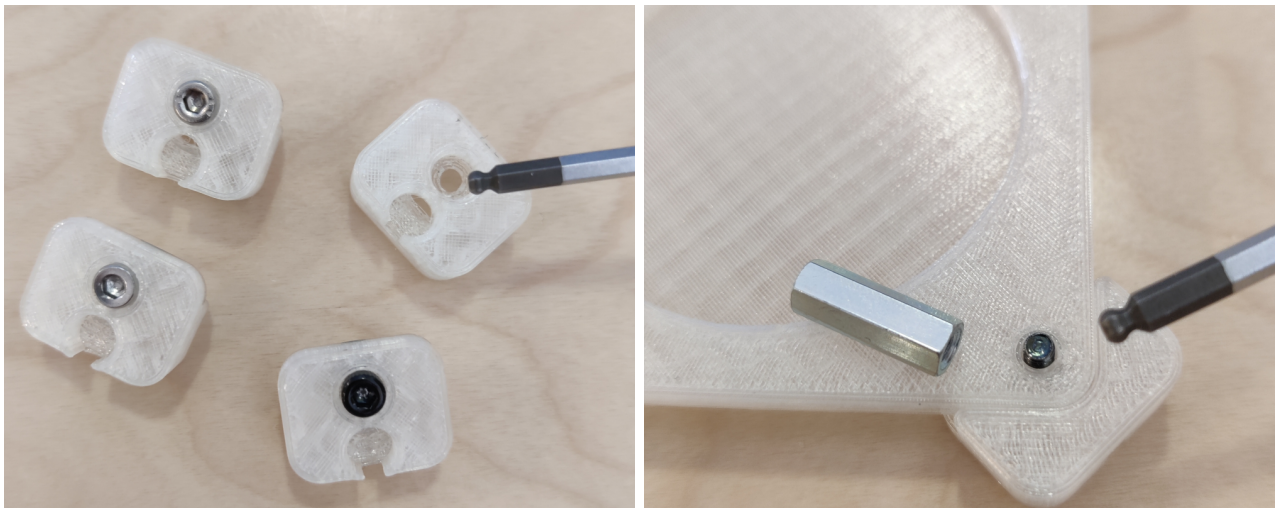


**Fig. 4.16**

The top of the module is then attached to the bottom by using the M6 screw to fasten the top through the screw sleeves (Figure 4.17, left). Make sure to align the holes properly.

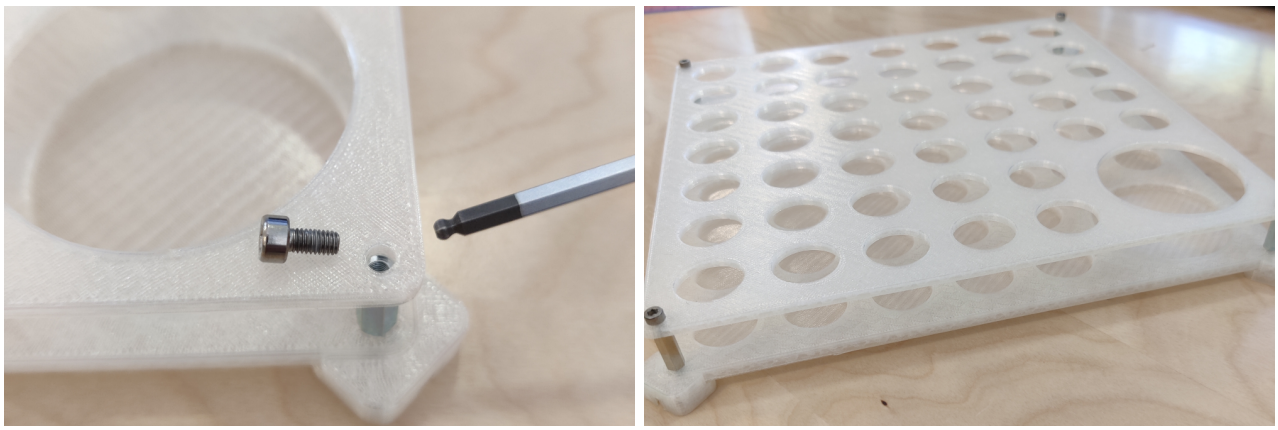


**Fig. 4.17**

# 5 Code documentation

## 5.1 General overview

The source code is written in python and consists of a main module running the ASM package. To work with the code as a developing tool the following python packages are needed:

- tkinter
- pandas
- numpy
- serial

For the ASM to work you will have to use the printer functions module, which has the functions as shown below.

**Listing 5.1:** start sequence

```
def start_sequence(comport=None):
    if comport is None:
        port = serial.tools.list_ports.comports()
        print("Connecting to " + port[0].device)
        arduino = serial.Serial(port[0].device, 250000)
    elif comport is not None:
        print("Connecting to", comport)
        arduino = serial.Serial(comport, 250000)

    config_data = {"X": 0, "Y": 0, "Z": 0, "arduino": arduino}
    settings.configure_printer(config_data)
    # time.sleep(2)
    settings.arduino.write("G28 \n".encode())  # homing command
```

```
    # time.sleep(15)
    print("home")
    move_to(z=settings.safety_height)
```

The start sequence establish a connection between the python code and the Marlin software through the serial port on the computer. Afterwards, it ensures that every module has the newly created "arduino" object throughout every module by using configure printer function from the settings module. It also sets the X, Y and Z axis to 0 and homes the machine through the G28 G-code.

Now the ASM is ready to be used. This gives two options, either you can use the Excel sheet we have created to create already established dosing methods or you can create your own. We will firstly look what it takes to create you own methods through our python package. The ASM is easily controlled through the other functions in the printer functions module. These being:

**Listing 5.2:** movement functions

```
def move_to(x=settings.X, y=settings.Y, z=settings.Z):
    if x is None:
        x = settings.X
    if y is None:
        y = settings.Y
    if z is None:
        z = settings.Z
    if z != settings.Z:
        sleep = math.sqrt((z - settings.Z) ** 2)
    else:
        sleep = math.sqrt(((x - settings.X) ** 2 + (y - settings.Y) ** 2))
    settings.arduino.write("G1 X{} Y{} Z{} \n".format(x, y, z).encode())
    time.sleep((sleep / settings.feedrate) * 2 + 1)
    # Update the printer head position
    settings.X, settings.Y, settings.Z = x, y, z


def move_relative_to(x=settings.X, y=settings.Y, z=settings.Z):
    if x is None:
        x = settings.X
    if y is None:
        y = settings.Y
```

```
23      if z is None:
24          z = settings.Z
25      if z != settings.Z:
26          sleep = math.sqrt((z - settings.Z) ** 2)
27      else:
28          sleep = math.sqrt(((x - settings.X) ** 2 + (y - settings.Y) ** 2))
29      settings.arduino.write("G91".encode())
30      settings.arduino.write("G1 X{} Y{} Z{} \n".format(x, y, z).encode())
31      settings.arduino.write("G90".encode())
32      time.sleep((sleep / settings.feedrate) * 2 + 1)
33      # Update the printer head position with the new relative movement
34      if x is not None:
35          settings.X += x
36      if y is not None:
37          settings.Y += y
38      if z is not None:
39          settings.Z += z
```

These two functions work similarly. These functions are used by calling the desired movement as such: move_to(x=100, y=100), which will make the ASM header move to X100 Y100 in the absolute coordinate system. The relative movement function is just that, it will make the 3D printer move relative to its current position. The last of the functions in the printer functions module is the end sequence, it powers off the steppermotors and disconnects the ASM from the serial connection and can be seen below.

**Listing 5.3:** End sequence

```
1
2   def end_sequence(pump_objects=None):
3       move_to(x=100, y=100)
4       time.sleep(5)
5       if pump_objects is not None:
6           for pump in pump_objects:
7               pump.unload()
8       settings.arduino.write("M81 \n".encode())  ##Power off
9       settings.arduino.close()
10      print("Thank you for choosing Jakob & Thorbjoern inc.")
11  end_sequence()
```

To get the ASM to dose anything you will have to establish pump objects. Multiple pump classes have been created and more are hopefully on the way. The pump objects are created

by assigning the pump characteristic to it as well as the extruder number, syringe size and the position on the dosing header. They have multiple functions in them so that they can load, dose, unload and be calibrated. Creating a new pump class can be done when you have some basic knowledge on the python workflow. To show you an example of how to create you own method take a look below

**Listing 5.4:** Example of a custom made method

```
1
2  from library.printer_functions import start_sequence, end_sequence, move_to
3  from pumps.valvepump import ValvePump
4
5  dosing_list = [1, 1, 2, 2]
6
7  start_sequence()
8
9  Toluene_pump = ValvePump(extruder_number=0, syringe_size=10, gantry_position
       =0, a_speed=0.005, b_vol=0.0014, a=0.3229, b=-0.1907)
10 Toluene_pump.load()
11
12 i = 1
13 for volume in dosing_list:
14     move_to(x=25*i, y=100)
15     Toluene_pump.dose(volume=volume)
16     i += 1
17
18 Toluene_pump.unload()
19
20 end_sequence()
```

This script starts of by importing the minimum number of necessary functions and classes. Then it creates the variable dosing list, which contains different integers and are the volumes we want to dose. The ASM is initialized by the start sequence and a valve pump object is created and assigned the necessary information. These characteristics is for toluene and thus we name our object Toluene pump. The toluene pump is then loaded and the dosing list is iterated through by a for loop. In the for loop we move to the 4 positions where we have placed our beakers, X25, X50, X75 and X100 all with the same Y coordinate, Y100. We then dose the volume that we have assigned in the dosing list to the corresponding beaker. We finish off the script by unloading the Toluene pump and then disconnect from the ASM by using the end sequence function. This is a simple method, which shows how to create custom methods and

learn the python package.

## 5.2 Python modules

The main python module is where the main window is declared. It is from this window that the rest of the modules are called. The settings module have been created to handle the fact that module wide globals cannot be created in python. Therefore, every module which should handle some kind of common varaible like the experimental dataframe. First the variable is loaded from the settings module, then the variable is used for whatever purpose, and then lastly if any operations has been done to the variable the new modified version is saved into the settings module. Thus, a pseudo-global module wide variable exists. Going forward it would be a good idea to make the hardcoded variables with actual values into somekind of .txt document, where it gets loaded from. Atleast if the idea is that it should be a .exe program in the end.
The rest of the GUI stems from the following modules: Experimental window, machine control, protocol creator, pump constructor, and settings window. The machine control module is a GUI window to make small movements with the machine, the purpose is to replace the use of pronterface. The pump constructor is a walkthrough of how to create a new type of pump with all the necessary data it needs. The protocol creator module is a GUI window where you type in the data for the protocol you want to create I.E. your experiment data. The settings window is a GUI window where you can change some general settings. The experimental window module is a GUI window which runs the protocol you have chosen. A lot of nested functions are within the main function of the experimental window function, these are necessary to make sure the values in the tkinter widget get inherited. While it is perhaps the biggest of the modules, the intricate details of how it handles protocol execution is crucial to know and understand to be able to develop further on it and to potentially debug the GUI. The rest of the modules are miscellaneous operators. Like the method functions which is a collection of the functions the methods use like the start sequence, end sequence, and move to function. These are very important to understand, what they do and how they are created. The file explorer is merely a collection of some common used tkinter widgets functions for the comboboxes and the sorts. Protocol manager is another very important module to understand. This handles the saving and loading of the protocols. How they are saved should also be how they are loaded. It is very important to ensure that the datatypes are correct after loading, since a wrong data type could result in an error. The last modules are the pump classes. These should also be understood

completely. Pumps are defined in the system as classes, which takes a lot of different arguments. We highly recommend taking the time to read and understand the sequence of gcode that the pumps have, since this would be the basis for creating new functions for the pumps or creating new pumps in general. Two different kind of pumps exists currently and that is because there is a difference in what output pin is used for each machine board. The BTT octopus uses fan output (M106) while the arduino mega with RAMPSX uses hotend output (M104).

## 5.3 Protocol

Protocols are created when running an experiment from user input. Protocols are .txt files, which contains all the necessary data to construct the three primary arguments which are needed to run the ASM; experiment-data, setup-data, and pump-objects. experiment-data is a dataframe containing all the necessary dosing information. It contains how much of each solution should be in each experiment in addition to the vial's X and Y coordinate. A picture of a dataframe can be seen below (figure 5.1.

| | Solution 1 | Solution 2 | x-coordinate | y-coordinate |
|---|---|---|---|---|
| Experiment 1 | 3.00000 | 5.00000 | 199 | 16 |
| Experiment 2 | 2.00000 | 4.00000 | 171 | 16 |
| Experiment 1 | 3.00000 | 5.00000 | 142 | 16 |
| Experiment 2 | 2.00000 | 4.00000 | 113 | 16 |
| | nan | nan | 84 | 16 |
| | nan | nan | 56 | 16 |
| | nan | nan | 28 | 16 |
| | nan | nan | 199 | 44 |
| | nan | nan | 171 | 44 |
| | nan | nan | 142 | 44 |
| | nan | nan | 113 | 44 |
| | nan | nan | 84 | 44 |
| | nan | nan | 56 | 44 |
| | nan | nan | 28 | 44 |
| | nan | nan | 199 | 72 |
| | nan | nan | 171 | 72 |
| | nan | nan | 142 | 72 |

**Fig. 5.1:** Example of a dataframe

It should be noted that every value inside the dataframe is a float, even the empty "nan" values are merely empty float values. The setup-data is a dictionary containing an informational overview about the experiment. This includes the name of the experiment, the protocol, the amount of pumps needed, and the amount of experiments. Lastly, the pump-objects is a list containing each pump object. The data to construct the pump objects are also retrieved from

the protocol. An example of a protocol can be seen below (figure 5.2

```
Dosing Method: standard_method
Reaction module: Cuvette module
Number of reaction vessels: 45
Max Volume: 3.0
Waste: [180, 180]
Beaker height: 5
Safety height: 10
Vial height: 3
Machine speed: 50
Solution 1 info: h
Solution 2 info: w
Pump 1 speed: 5.0
Pump 2 speed: 5.0
Viscous solution 1: 0
Viscous solution 2: 0
Gantry position 1: -1
Gantry position 2: 0
Waste coordinate 1: 180.0,180.0
Waste coordinate 2: 180.0,180.0
Pump 1: 1 mL Braun (water) no b
Pump 2: 1 mL Braun (water)
index   Solution 1      Solution 2      x-coordinate    y-coordinate
Experiment 1    3.0     5.0     199     16
Experiment 2    2.0     4.0     171     16
Experiment 1    3.0     5.0     142     16
Experiment 2    2.0     4.0     113     16
        nan     nan     84      16
        nan     nan     56      16
        nan     nan     28      16
        nan     nan     199     44
        nan     nan     171     44
        nan     nan     142     44
        nan     nan     113     44
        nan     nan     84      44
        nan     nan     56      44
        nan     nan     28      44
        nan     nan     199     72
        nan     nan     171     72
        nan     nan     142     72
        nan     nan     113     72
        nan     nan     84      72
        nan     nan     56      72
```

**Fig. 5.2:** Example of a protocol

As can be seen, there is first all the machine settings, followed by the pump settings needed to create pump objects and then finally the values needed to construct the experimental dataframe.

## 5.4 Method

The three methods which have been made at the point of writing is the standard (S), custom/-timed (CT), and calibration (C).

### 5.4.1 Calibration method

The purpose of C is to calibrate new pumps usually in the form of calibrating an already existing pump with a new type of syringe. The code is therefor also the simplest since it only needs to dose from a single pump signal into each vial, a flowchart of the code can be seen in figure 5.3.

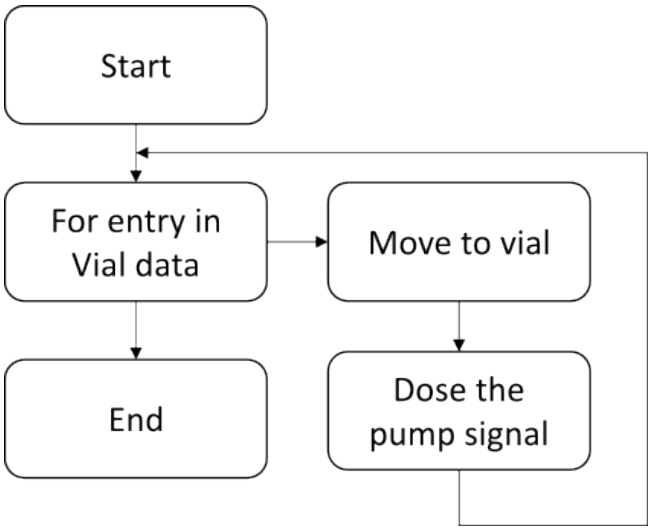



**Fig. 5.3:** Flowchart of calibration method

From the flowchart it can be seen that the code only consists of a for-loop where it goes over each experiment and doses into a vial depending on the pump signal. The experiment data differs only on a meta level in the form that instead of giving a volume, it gives a pump signal that should be dosed. This also means that the user should know for themselves what range of pump signals are best to investigate.

### 5.4.2 Standard method

The S is based on the simplest experiment setup where you need to add 1-5 solutions together. Thus, the code consists of two for-loops as can be seen from figure 5.4.

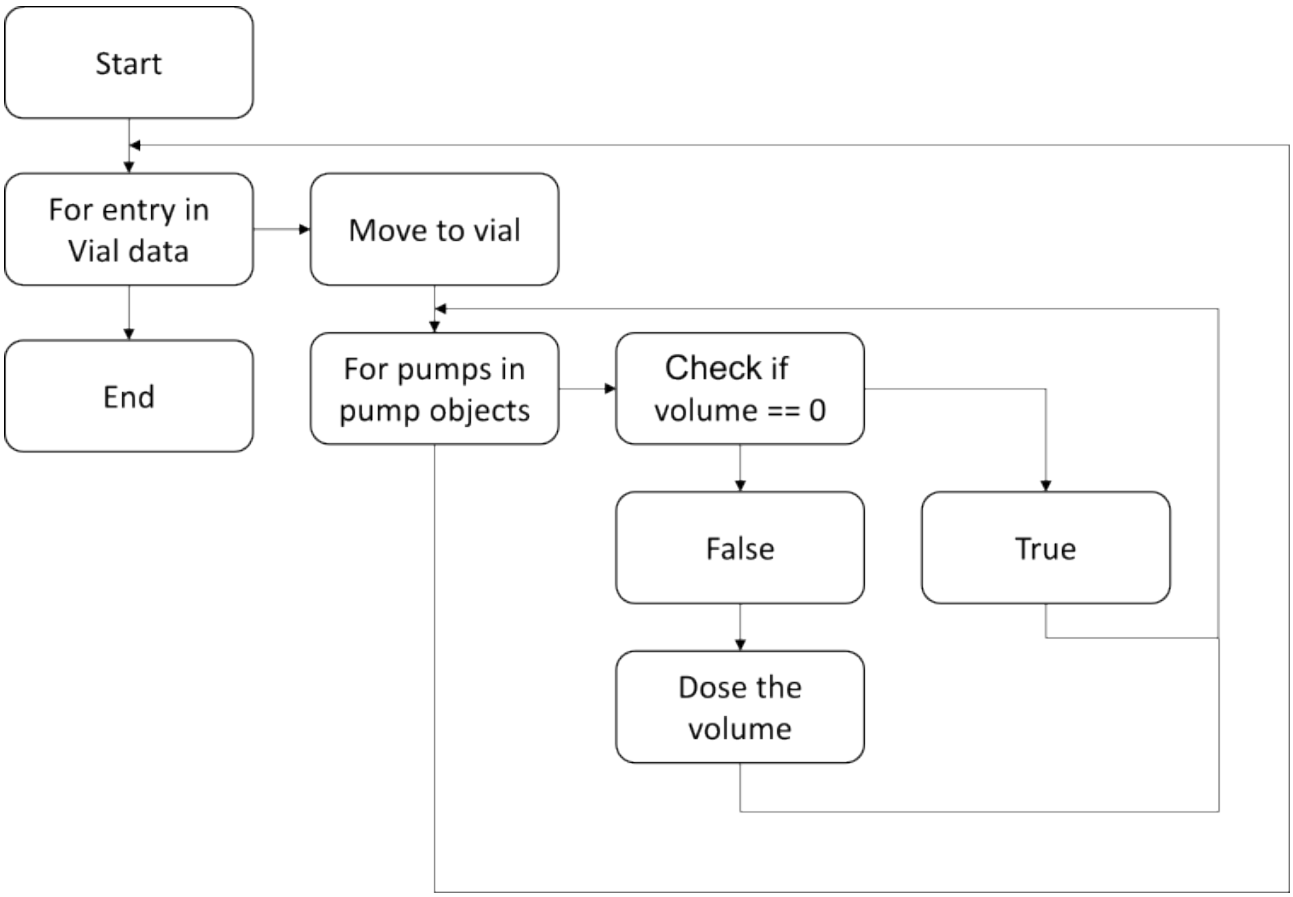



**Fig. 5.4:** Flowchart of the standard method

The first for-loop is similar to C where each experiment is iterated over, however, whereas C only doses from one pump, S needs to additionally iterate over each pump, ie. from the experiment-data it iterates over the solutions. If the solution has a volume of 0 it will skip the solution by continuing the for-loop.

### 5.4.3 Custom timed

The CT is the most complex of the three methods. The purpose of the method is to be able to dose 2-5 solutions in any order the user wants independently between each experiment, ie. experiment 1 goes A-B-C, experiment 2 goes A-C-B, etc. Additionally, the user has the possibility to introduce a time variable between each dosage starting from the second dosage, ie. solution B needs to be dosed 5 min. after solution A, and solution C 10 min after B. The CT was constructed on the basis of this usage. Therefore, in addition to the usual information found in experiment-data, there are additional columns containing the dosing priority of each solution and a timer between each priority. CT works in three major phases. Firstly, there is the data pre processing followed by main-loop 1, and lastly main-loop 2. The data pre processing is necessary to be able to arrange a nested dictionary, which is much easier to iterate efficiently through than dataframe. Based on the experiment-data the following dictionary is constructed

as seen in figure 5.5.

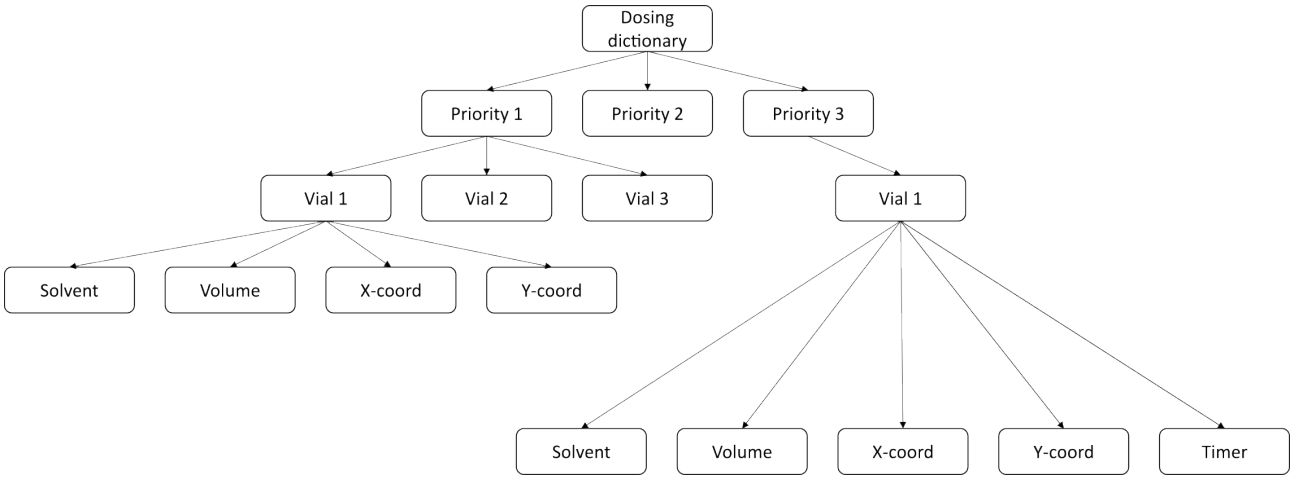



**Fig. 5.5:** Structure in the dosing dictionary. The experiment data is sorted according to the priorities

The dosage dictionary (DD) consists of nested dictionaries based on the priorities. Each priority then further has nested dictionaries for each experiment. These experiment dictionaries contain information about the pump it should be dosed from, the volume of the solution as well as the X and Y coordinate for the experiment. For priorities beyond priority 1, there is additionally a timer key which contains how much time should elapse before that solution should be dosed into that experiment after the prior solution has been dosed. The important function of this dictionary is the ease of iteration and when to stop for a certain experiment. Checking to see if the experiment is done getting solutions is simply to see if there is an experiment dictionary in the next priority dictionary.

After the dosing dictionary has been created the code goes through the two main loop phases. Phase 1 consists of a for loop which iterates over every key in priority 1 and can be seen in figure 5.6

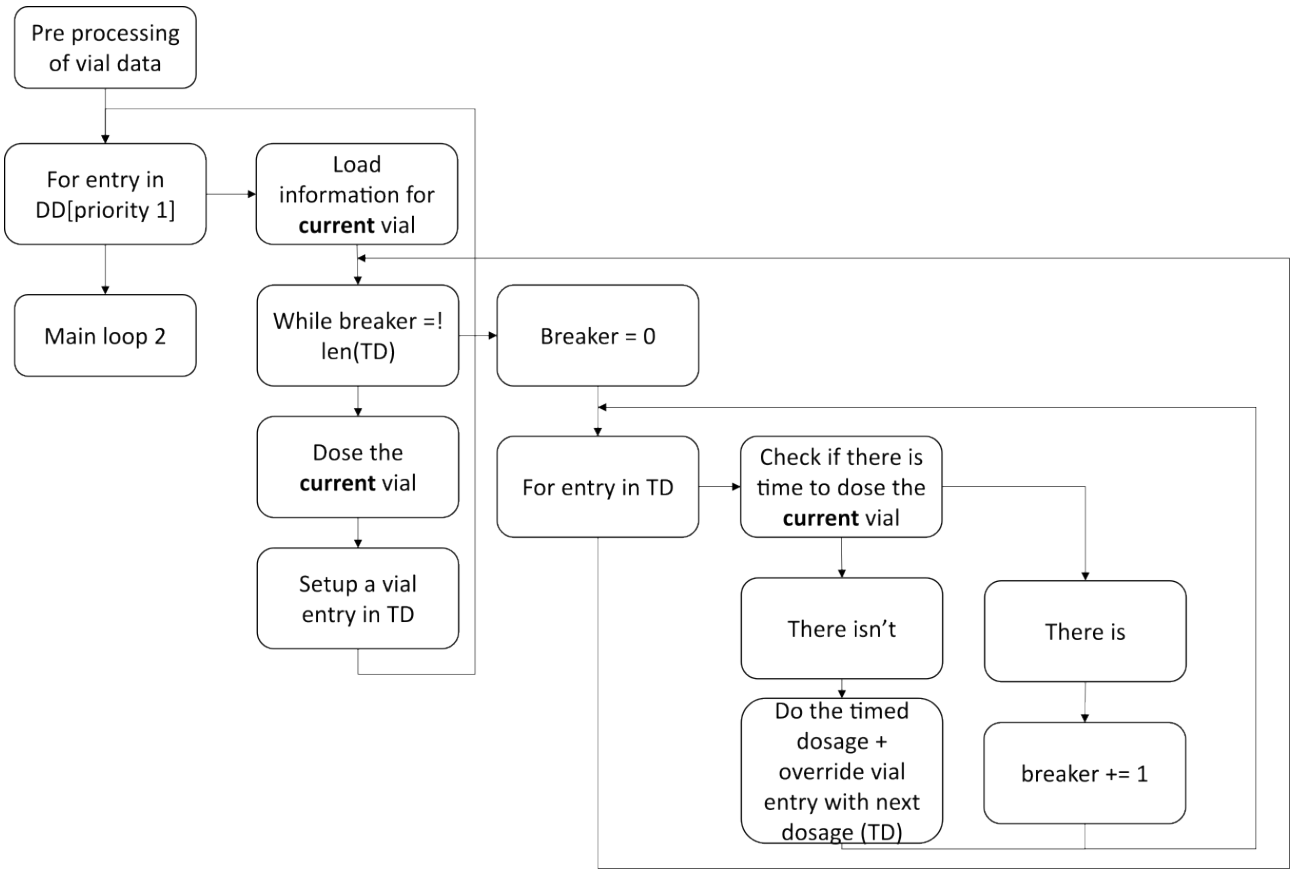



**Fig. 5.6:** Flowchart of the first loop in CT

As seen in the figure, experiment information is first loaded into variables. For the first experiment, it does not go into the while loop since the timed dictionary is empty. Using the loaded variables, the first is experiment is thus dosed without delay. The last step of the for-loop is to setup an entry in the timed dictionary, which can be seen in figure 5.7

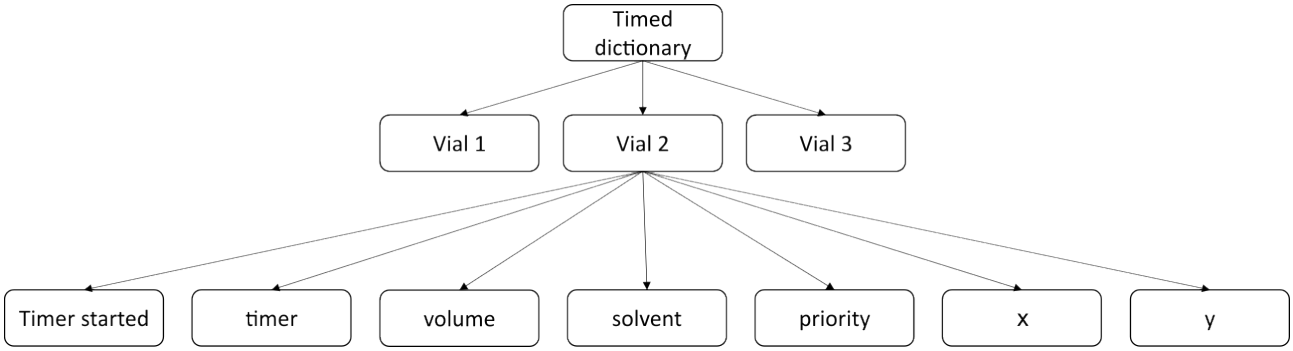



**Fig. 5.7:** Structure in the timed dictionary

By using the dosage dictionary, the next dosage is prepared by writing an entry into the timed dictionary. The timed dictionary has a nested dictionary for each experiment. These are built partly from the dosage dictionary and therefor contain which pump should be used, volume, x, y, priority and timer as keys. Additionally, they also contain when the prior dosage was dosed to keep track of the time elapsed and when this dosage should be done. The way the timed dictionary is utilized is that these nested experiment dictionaries are continuously overwritten

when that dosage occurs. Then, when the last dosage for an experiment occurs, the experiment dictionary is removed in the timed dictionary. This is what happens at the end of the while-loop in main-loop 1. When there is entries in the timed dictionary, after loading up a new experiment in the variables the code will go through the while loop. In the while loop there is a for-loop going over every entry in the timed dictionary. First it calculates the amount of time the current experiment would take to dose, then it compares that value to the timer for each entry in the timed dictionary. It takes the first found and not the one closest to reach its timer. The timer is calculated and if it is less than the time it would take to dose the current experiment it will delay the current dosage and instead dose the timed dosage instead. If a timed dosage has been dosed the breaker will not match the length of the timed dosage dictionary and thus not break out of the while loop and start over from the for-loop for the timed dictionary. The main-loop 1 will continue until each priority 1 has been dosed going to main-loop 2, which can be seen in figure 5.8

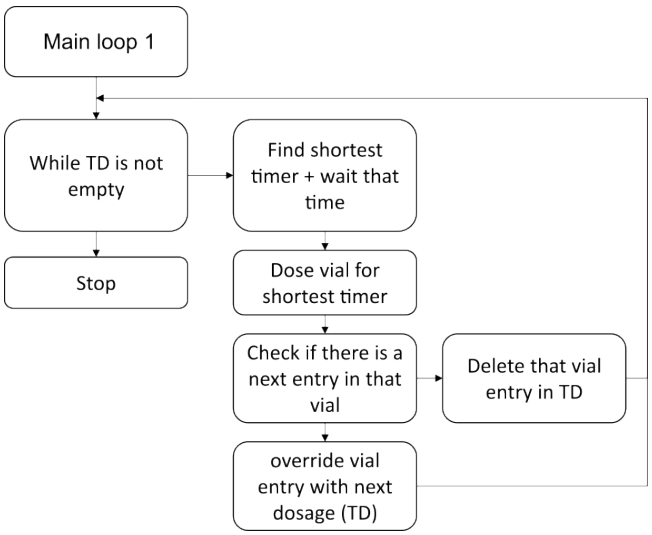



**Fig. 5.8:** Flowchart of the second loop in CT

This code consists of a while loop, which will only break when the timed dosages dictionary is empty of entries. It starts of sorting the timed dictionary for the lowest timer and then wait until that time has elapsed to dose for that experiment. Then, it will see if there is more priorities for that experiment, if there is, it override the current with a new entry, and if not it will delete that entry. Thus, after everything has been dosed there is no more entries left in the timed dictionary, which is also the condition for the while loop to stop, ending the code. The entire code can be seen as a flowchart in figure 5.9

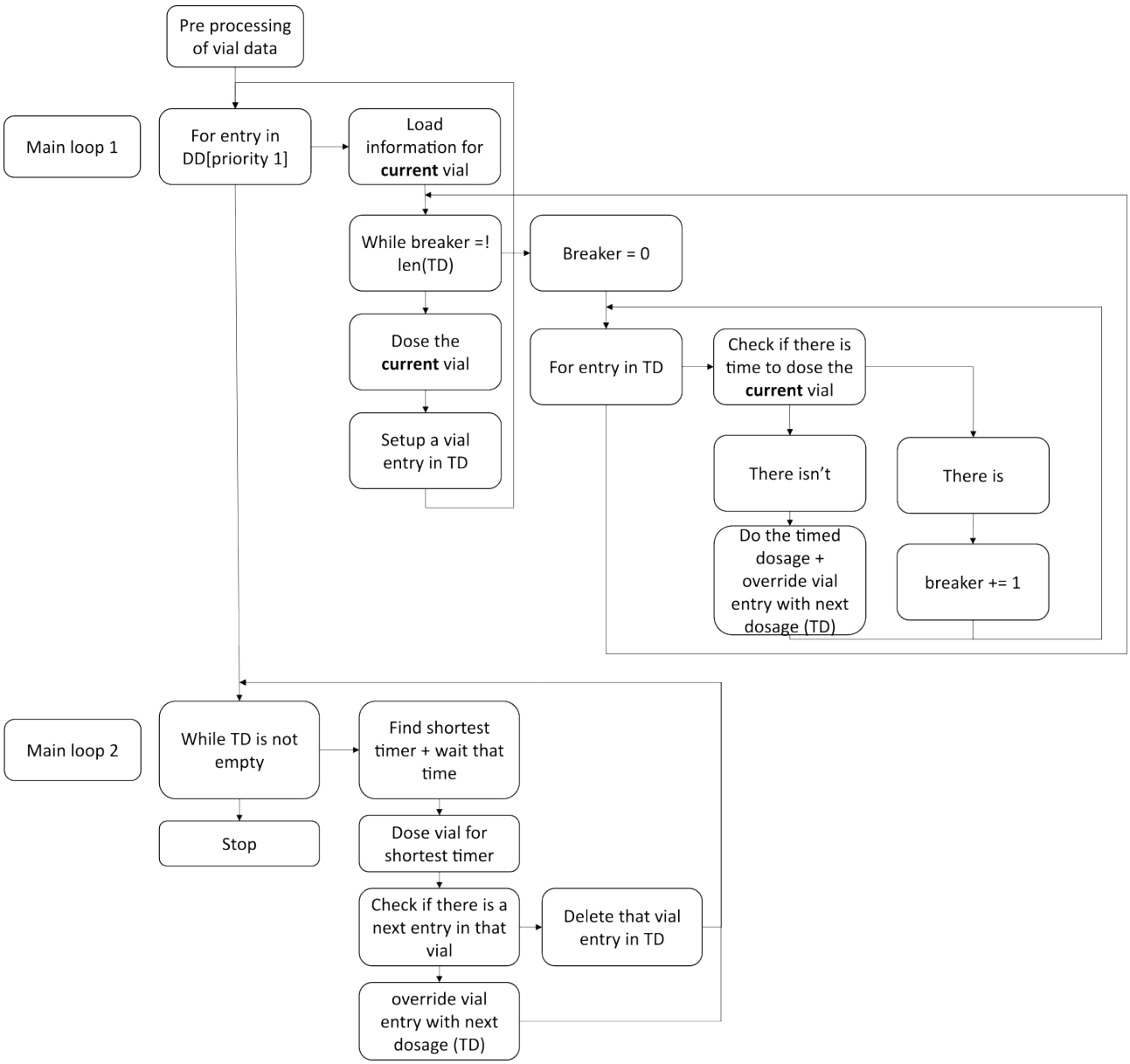



**Fig. 5.9:** Flowchart of custom/timed method