Developing an Automated System for Yeast Culture Cultivation and Control using Flow Cytometry

Sara Magnusson
Division of Applied Microbiology
Lund University
25th of September 2023

Supervisors
Magnus Carlquist
Raquel Perruca Foncillas
Marie Gorwa Grauslund
Ola Wallberg

Abstract

Flow cytometry is a versatile tool for monitoring a microbial population at a single-cell level. Multiple parameters can be monitored based on the users' requirements through the utilization of different dyes, biosensors etc., and the use of FCM for on-line measurements during fed-batch cultivations has the potential to be used for precise process control.

This project consisted of the development and testing of a Python program for on-line bioprocess regulation based on flow cytometry data. Specifically, the program was built to process FCS files resulting from at-line measurement of pentose-fermenting *S. cerevisiae* using the OnCyt autosampler connected to a BD Accuri C6+ flow cytometer. The program was designed to automatically start and stop a peristaltic pump through serial communication, by issuing commands based on changes within the microbial population.

The yeast strain used was the TMBRP011 strain, which was previously engineered for bioethanol production from pentose sugars and carries a previously developed redox biosensor. The biosensor reacts to cellular redox imbalance caused by inhibitory substances released during lignocellulose pretreatments through an increase in fluorescence. The program was designed to achieve higher volumes of inhibitory substances as the cells acclimatized to a set rate, indicated with a decrease in fluorescence. However, since these inhibitory substances are toxic in nature, the pump should stop pumping if the percentage of PI-stained cells within a sample increased, indicating cell membrane damage.

The program was successful in regulating a fed-batch process based on the prementioned parameters, and it managed to reinduce the cells by injecting higher volumes of inhibitory compounds per hour. The program was developed in a way that fulfilled the initial scope in terms of functionality and provides a roadmap for implementing on-line FCM as a basis for process control.

Foreword

This master thesis was performed from the 8th of May until the 25th of September, and thereby lasted for 20 weeks. Any practical experiments were performed at Kemicentrum, Sölvegatan 39, 223 62 Lund, and equipment was provided by the Division of Applied Microbiology, as well as the Department of Chemical Engineering.

Thank you to all the acting supervisors for this project; Magnus Carlquist, Raquel Perruca Foncillas, Marie Gorwa Grauslund and Ola Wallberg, and a special thank-you to Basel Al-Rudainy for the construction and installation of the Arduino, along with pump troubleshooting. I would also like to thank Ed Van Niel for acting as my examiner for this project. I am very grateful for all the advice and expertise you all have shared with me, and I am further grateful that I was entrusted with this project.

I am especially thankful to Raquel Perruca Foncillas for unfailingly assisting me and being a huge support for laboratory work and system setup deliberations. Thank you for spending endless hours musing over a malfunctioning flow cytometer with me while I could not even be of much help, even though you had much better things to do.

Another special thank you goes to Magnus Carlquist for always being available for input from the formative days of this project until the end, and for lending your expertise for experiment planning along the road. Although I could not always decipher your handwriting, I was always grateful for your input and support.

Finally, I would like to thank everyone at the Division of Applied Microbiology for their kindness and willingness to help with everything from leaking autoclaves to taking plates out of heating cabinets. Thank you also for the randomly occurring discussions that have helped this project be completed in the way that it has.

Table of Contents

Abstract 1
Foreword 2
Table of Contents 3
Background 5
1.1 Flow cytometry and Fed-batch fermentation 5
1.2 Bioethanol Production 5
1.3 Automation and Python 6
1.4 Aims of this project 7
2 Materials and Method 7
2.1 Coding platform and Outline 7
2.2 Python packages 8
2.3 Yeast strains 8
2.4 Pre-cultures 9
2.5 Analytical methods 9
2.6 Evaluation of base program functionality 9
2.7 Batch Cultivation 10
2.8 Fed-Batch Continuous Cultivation and Final set-up. 11
2.9 Recipes, and Equations 11
3 Result 14
3.1 FCM data processing 14
3.2 Pump control program 16
3.3 Results of the evaluation of base program functionality 22
3.4 Batch cultivations for determination of glucose consumption rate 22
3.5 Evaluation of sampling procedures 24
3.6 Comparisons between fed-batches and dynamically controlled fed-batches 24
4 Discussion 25
5 Conclusion 27
6 Future outlook 27
7 Popular Scientific Summary 29
References 30
Appendix 1 32
i. FCS Trial 32

ii. FCS Handling 35

iii. Pump Handling 37

iv. Pump Mock 39

v. Main Executor 40

Background

1.1 Flow cytometry and Fed-batch fermentation

Flow cytometry (FCM) is a versatile tool for single-cell analysis of a microbial population. There are several ways to utilize it to measure cell number and activity within a population. FCM data enables many separates culture statistics to be monitored, while the analysis is kept growth independent.

Cells suspended in liquid pass through one or several lasers, where the resulting light scatters give information on the cells' size and granularity, but also levels of any relevant stains or fluorescent biosensors, dyes, or antibody conjugates. The number of parameters one could monitor has only increased in recent years as development and implementation of new fluorescent dyes have enabled the measuring of strain-specific parameters (Drescher et al., 2021). A popular example of this is propidium iodide, or PI staining which is used to indicate cell membrane integrity by binding to DNA while not being able to permeate intact cell membranes (Riccardi and Nicoletti, 2006).

A fed-batch fermentation process is defined by continuously feeding the cell culture with substrates and nutrients such as sugars that the cells can utilize for either product generation or cell duplication. Typically, the feeding phase is preceded by a batch phase with a set concentration of nutrients present in the medium. The feed is then used to achieve optimal conditions within the cultivation vessel. This in turn will yield a faster process with higher product and biomass yield than a simpler batch fermentation. This and the ability to have the culture producing for a much longer time while limiting potential inhibition from product or substrates, makes fed-batch processes ideal for many processes within the industry (Poontawee and Limtong, 2020).

On-line flow cytometry with its continuous measuring capabilities is a potential option for fed-batch process control. Where many alternatives more commonly used today such as CO_2 sensors base their data on the entire culture (Nilsson et al., 2001), single-cell analysis and automatic gating based on different parameters presents an opportunity to reduce workload by making certain plating or HPLC steps redundant, depending on the implemented biosensors. Flow cytometry also is a useful tool for optimizing cultivation processes to reach optimal robustness or biomass production by analyzing a microbial population at a single-cell level. (Fernandes et al., 2013)

1.2 Bioethanol Production

Bioethanol is an example of a biofuel that has become widely used as an alternative to fossil fuels, due to its low greenhouse gas emissions, biomass availability and cost effectiveness (Farrell et al., 2006), (Hahn-Hägerdal et al., 2006).

The conversion of lignocellulose to bioethanol can be achieved in several ways, however there are some main staples that persist. The cellulose and hemi-cellulose contained in the biomass must first be depolymerized through enzymatic hydrolysis into sugars, primarily pentoses and hexoses. In order to achieve this efficiently, a pretreatment of the raw biomass is performed where the sugars are made more accessible for the enzymes (Akshay et al., 2021). During this pretreatment, inhibitory compounds can be released

which may hinder the hydrolyzation process. These substances include less potent acids such as acetic acid, phenolic compounds, and furaldehydes such as 5-hydroxymethyl furfural and furfural itself (Akshay et al., 2021). Hence, the yeast strain used for the cultivation must have a high tolerance for these substances, while also having the phenotype to metabolize both the pentoses and hexoses present in the hydrolysate. Industrial engineered strains of *Saccharomyces cerevisiae* has been used for this purpose previously (Perruca Foncillas et al., 2023).

In order to monitor the fitness of the cells during a fermentation process, redox imbalance has been used as a marker in previous studies (Almeida et al., 2008), (Ask et al., 2013). Reducing toxic aldehydes to their less toxic alcoholic forms has been shown to cause a redox imbalance in the cells as the NAD(P)H cofactors are utilized for the reduction (Perruca Foncillas et al., 2023). A biosensor has been engineered for the purpose of monitoring the extent of this imbalance, by utilizing the Yap1p gene and its promotor TRX2p to detect changes in the fluorescence through a tool such as a fluorometer or flow cytometry for at-line sampling (Zhang et al., 2016).

A common inhibitory compound in lignocellulose hydrolysate is furfural, which has been shown to trigger a response from this *TRX2p-yEGFP* biosensor, inducing the yeast cells and increasing the average green fluorescence protein (GFP) fluorescence of the culture. It was shown that there was a correlation between this induction and the production rate of bioethanol. With a constant feeding rate of furfural, the induction levels would increase indicating a redox imbalance, however further in the cultivation these levels would start decreasing, indicating that the same rate of added furfural no longer caused the oxidative stress needed for measurable induction. Thus, the cells have adapted to the injected furfurals toxicity and are able to handle higher levels (Perruca Foncillas et al., 2023). The measurement of this phenomenon with FCM may be used for process control.

1.3 Automation and Python

Automation within the scientific space and more particularly in the life science space has rapidly grown in importance on a lab scale as the industry continues to advance. While automation has the advantages of more robust and reproduceable methods with less researcher involvement, the ever-changing protocols and limited funding of projects performed in research labs has created an ‘automation gap’ when compared to the industry. When considering automation options, one should consider its implementation options in both large and small scale, and developers will require skills in both biology and technology to create something that is adequate and implementable in many scenarios and setups. As such, automation has high potential when it comes to making laboratory work easier and less strenuous on a researcher, which in turn enables them to create more elaborate experiments (Holland and Davies, 2020).

Programming is a skill that could become essential as technology continues to advance. Python has become a prevalent coding language used within the biotechnology space and is widely used for sequencing and bioinformatics. The ability to build specialized and efficient software that is easily maintained and further tailored to new datasets has become a valuable skill within the space for good reason (Shajii et al., 2021). There are many

possible applications of the language, and with more and more prebuilt modules used for biological applications being created the possibilities are many. Python can be used as a viable tool for automation strategies for in-house laboratory procedures, which if successful can be scaled up to commercial processes should the need arise (Holland and Davies, 2020).

1.4 Aims of this project

The aim of this project is to develop an automated system using Python to manage and control a microbial population using FCM data. The program was to be created to be reactive to changes within a population on a single cell level and react accordingly to said changes, enabling more precise control over more parameters than what is widely employed in the industry currently. The program should also serve as a general proof of concept for bioprocess control using flow cytometry data.

Parts of the program were to be developed with a specific yeast strain in mind. This *S. cerevisiae* strain which was engineered for lignocellulose bioethanol production, was the base for gating strategies and execution on a laboratory scale. Post-program development, this yeast strain was to be cultivated anew while controlled by the program to investigate whether it was possible to re-induce the cells as they adapt to furfural toxicity by injecting higher volumes based on changes in the induction level, which in turn could have a positive effect on bioethanol production based on previous studies (Perruca Foncillas et al., 2023).

2 Materials and Method

2.1 Coding platform and Outline

The code was primarily written using Python 3.9.13 on a Windows 11 computer, using the IDE PyCharm Community Edition 2023.1.1. This setup was primarily for testing and was not used on the BD Accuri C6+ computer, which runs Windows 7 and hence required a downgrade to Python 3.7.9 and PyCharm Community Anaconda Edition 2019.3.5. The downgrade brought with it some differences in code syntax, along with some reduced base functionality of the PyCharm user interface.

When developing the code, previous cultivation flow cytometry standard (FCS) files for the relevant yeast strain were used. The files were generated by the fed-batch processes done by Perruca Foncillas et al., 2023 and were used to simulate a fed-batch process where new files were created in a set folder. Program output results were compared to the same FCS files analyzed by the FlowJo software (BD Biosciences, Franklin Lakes, NJ, USA).

The original outline was for the program to be able to execute the following:

I. Retrieve data from created FCS files continuously and automatically.
II. Gate and plot the data on a single-cell level through a scatterplot. This should separate the induced intact cells, expressing GFP from non-induced intact cells and cells that are damaged or dead. The gates should also be able to remove any potential debris present in the samples.

III. Perform calculations on the gated samples to calculate the percentage of PI-stained cells, the mean GFP fluorescence of the intact cells and total number of events recorded.
IV. The program should be modular and have the ability to pass on and utilize any parameters that flow cytometry data can provide by changing the calculation parameters.
V. Pass these parameters to a function that can determine whether the slopes of the main parameters, in this case PI percentage and mean fluorescence intensity (MFI) GFP, is increasing or decreasing.
VI. Utilize the slope values to stop and start a pump with the help of an Arduino, Raspberry Pi, or other communication module. The pump should be started as the GFP fluorescence is decreasing indicating a drop in induction level, and stop if the PI percentage is increasing, indicating cell membrane damage.

2.2 Python packages

When building Python programs, it is usually encouraged to call on Python packages to simplify code structure and ensure adequate documentation for future developers, while avoiding potential program conflicts and keeping the code modular (Foundation, 2023c). For this project, several of these were employed. First, to extract data from generated FCS files the package 'FlowCytometryTools' (Yurtsev and Friedman, 2018) was used. To create scatterplots of these FCS files as well as graphs of program output, the package 'Matplotlib' (Matplotlib, 2023) was used. This provides a baseline graphical interphase able to create plots in new windows.

'pySerial' (Liechti, 2020) was used to establish serial communication between the computer and the Arduino used to control the pump. The base python package 'time' (Foundation, 2023d) was used primarily for the time.sleep function, which makes the program stall execution for a specified number of seconds. The similar package 'datetime' (Foundation, 2023b) was also used to initialize start times for the program in the format of a date and precise time.

The 'watchdog' (Mangalapilly, 2023) package was used to continuously monitor a specified folder and execute specified commands based on activity within said folder. 'pandas' (NumFOCUS, 2023) was used for its data frame functionality which enables calculations on data columns without long data transformation sequences. 'scipy' (community, 2023) was used for its built in linear regression function to calculate slopes. Finally, the 'atexit' (Foundation, 2023a) function was used to execute final commands as the program was terminated.

2.3 Yeast strains

The yeast strain used in both the trial files and experimental cultivations was the *S. cerevisiae* strain TMBRP011. The strain was previously obtained by engineering the industrial strain cv-110 to carry the *TRX2p-yEGFP* biosensor by utilizing a CRISPR/Cas9 system (Perruca Foncillas et al., 2023).

The *S. cerevisiae* strain used as a positive control in the initial artificial culture trials was TMBRP013 which carried the *TEF1*p-*yEGFP* construct. In this case, *TEF1*p is a constitutive promoter which means that the cells should be expressing GFP constantly, without the need for causing a redox imbalance (denoted TEF) (Perruca-Foncillas et al., 2022). The negative control was non-induced TMBRP011 cells (denoted TRX).

2.4 Pre-cultures

The relevant strains were taken from their respective -80°C stock and streaked on yeast extract-peptone-dextrose (YPD) plates which was incubated at 30°C for 48 hours. A two-step pre-cultivation was then performed where the cells were inoculated in a 50 mL Falcon tube with 5mL YPD medium. The over-night culture was subsequently placed in a shaking incubator at 30 °C and 180 rpm for 17 hours. The cultures were then reinoculated to an OD_{620nm} of 0.6 in new falcon tubes containing 5 mL of YPD, which was incubated for 4.5 hours under the same conditions.

2.5 Analytical methods

A combination of the Accuri C6+ (BD Biosciences, Franklin Lakes, NJ, USA) flow cytometer and the OnCyt autosampler (OnCyt, Switzerland) was used to monitor fluoresence and the amount of stained cells, along with the number of recorded events in a taken on-line sample. The Accuri C6+ was equipped with two excitation lasers, one at 488 nm and the other at 640 nm, along with detection filters of 533/30 nm in FL1, 585/40 nm in FL2, 670 LP in FL3 and 675/25 nm in FL4. Samples were diluted using phosphate buffered saline (PBS) to a total dilution factor of 100x, stained with 10 µg/mL propidium iodide (PI) and incubated for 5 minutes. The Accuri C6+ was set to run fixed-volume samples of 35 µL at a speed of 35 µL/min for cultivation samples, and to run MQ water for four minutes at 35 µL/min in between as washing step.

The OnCyt communicates with the Accuri C6+ through remote interfacing and allows the user to customize the sample cycles through the OnCyt software, and repeat these cycles over a long period of time. Samples were taken directly from the cultivation shake flasks to monitor PI and GFP continuously. The analysis data of every sample were stored as a FCS file in a specified output folder and denoted A01 to H12.

Cell concentrations of manual samples were estimated off-line through sample optical density using an Ultrospec 2100 pro UV/Visible spectrophotometer (Amersham Biosciences, Buckinghamshire, UK) at 620 nm (OD_{620}). Medium composition and metabolite concentrations were subsequently analyzed after centrifugation at 13,000 rpm for 5 minutes. The recovered supernatant was analyzed using a Waters HPLC system (Milford, CT, USA) coupled with an Aminex HPX-87H column (Bio-Rad, Richmond, VA, USA). The mobile phase used was 5 mM sulfuric acid, with a constant flowrate of 0.6 mL/min.

2.6 Evaluation of base program functionality

Initial trials were done to evaluate the functionality of the program during a live sample cycle. This was done by preparing samples of TRX cells, used as a GFP negative control, and TEF cells used as a GFP positive control in separate 1.5 mL Eppendorf tubes. One

sample of TRX cells was additionally placed in a 70°C-heating cabinet for 3 hours in order to damage them for use as a positive PI-stained control.

Samples were created by utilizing different volume compositions of the three controls to simulate a sequence of events within a culture which would cause the program to react. The exact volume compositions can be seen in table 2.1. Samples were run in an order that would initially simulate an increase in MFI GFP, followed by a subsequent decrease which was meant to cause the program to start the pump. Then, an amount of the PI-positive control was to be inserted to simulate culture damage, which the program should react to by stopping the pump.

Samples were manually prepared with OD_{620nm} measurements of 0.1 They were analyzed through the OnCyt autosampler and BD Accuri C6+ flow cytometer, and controls for the pure control-samples were run and displayed by the written FCS Trial program with specialized gates before startup. These new gates were made to accommodate the fluorescence differences between the TEF and the induced TMBRP011 strain.

Table 2.1. Depicting sample composition volumes for the primary evaluation of program functionality. The TRX strain was used as a negative MFI GFP control, while the TEF strain was used as a positive control. Damaged cells were inserted into the last sample to trial the reaction of the program when the percentage of PI-stained cells increases.

Sample	TRX (μL)	TEF (μL)	Damaged
0	900	100	-
1	800	200	-
2	500	500	-
3	100	900	-
4	500	500	-
5	800	200	-
6	900	100	+

2.7 Batch Cultivation

A 500 mL baffled shake flask equipped with a handcrafted cotton stopper with four inserted lines was used for the cultivation experiments. The shake flask was reinoculated to an OD_{620nm} 0.1 in 50 mL of YP medium, 40 g/L glucose and 40g/L xylose using the secondary pre-culture. The cultivation was then incubated aerobically in a shaking waterbath at 30°C. Manual 1 mL samples were taken hourly during start and finish, where OD_{620nm} was recorded, and the medium composition was analyzed using an HPLC. The manual sampling line had a length of approximately 20 cm made up of a Masterflex® 96400-16 line. The batch cultivations were run for 21-23.5 hours depending on when glucose was deemed as depleted through HPLC peak analysis.

2.8 Fed-Batch Continuous Cultivation and Final set-up.

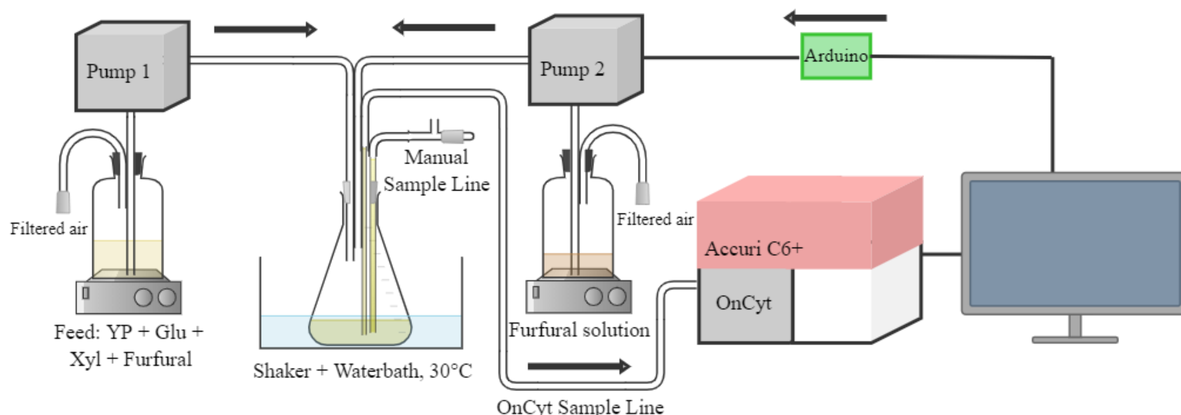

Figure 2.1. A schematic depicting the overall setup for a fed-batch cultivation regulated by the Python script. Pump 1 is used for achieving a constant feed rate, while pump 2 is controlled by the python script through an Arduino, pumping a solution of water and furfural. The OnCyt automatically takes samples through its sample line and analyses it in the BD Accuri C6+, which in turn informs the program. The manual sampling line was used to take samples for media analysis using HPLC and OD_{620} measurements. Schematic created on Chemix.org (Codelite Ltd, 2023).

Figure 2.1 depicts a schematic figure of the final experimental setup for the fed-batches. The shake flask with the batch cultivation was connected to 3 new lines along the manual sampling line after depleted glucose.

Line 1 held the feed, containing YP medium, 20 g/L glucose, 20 g/L xylose, and 6.9g/L furfural, with a constant feed rate of 104 µL/min using a peristaltic pump, denoted 'Pump 1' in figure 2.1. This line had a length of approximately 1 meter and was made up primarily of Masterflex® 96400-16 lines with fitted connectors.

Line 2 held the furfural-water solution with a furfural concentration of 50 g/L, and the rate of entry was controlled by an arduino connected to the PC and subsequently controlled by the program. This peristaltic pump, denoted ' Pump 2' in figure 2.1 was set to an rpm of 43 which corresponded to to a pump capacity of 259 µL/min or 15.6 mL/h. This line was approximately 2 meters long made up primarily of Masterflex® 06402-14 lines with fitted connectors.

The third line is the OnCyt sample line, which automatically took a sample from the cultivation at set times and analyzed them using the BD Accuri C6+ flow cytometer. The fed-batch cultivations were run for a total of approximately 44 hours, with 22 of those consisting of the feeding phase. The line was approximately 1.5 meters in lengths made up primarily of the OnCyt line itself.

2.9 Recipes, and Equations

Table 2.2 contains all utilized mediums and solutions used throughout the experimental phase of the project. The primary medium used was YP, with a sugar content of either solely glucose for the precultures, or glucose and xylose for the cultivations. Cells were also plated on YPD plates from a stock of the TMBRP011 strain every other week.

PI staining was used as an indicator of cell-membrane integrity during the cultivations and was used automatically by the OnCyt autosampler, diluting the samples 2x. Furthermore, cultivation samples were automatically diluted an additional 50x with PBS, which was used as a buffer. Similarly, Sodium Hypochlorite 0.5% and Sodium Thiosulphate 10 mM were used as recommended for sterilizing and quenching respectively after every taken sample (Besmer et al., 2016).

The recipes for the standard solutions needed to run the BD Accuri C6+ flow cytometer can also be seen in table 2.2. These were made according to instruction with the existing stock solutions provided by BD Biosciences.

Table 2.2. Depicting solutions and media as well as the recipes used during conducted experiments. All final volumes were measured in measuring cylinders while considering dissolved solids and their eventual water-contents The pH of PBS was checked with dissolved solids at 800 ml. '(AC)' denotes solutions that were autoclaved after mixing.

Solution	Final Volume	Recipe
YP medium 90% (AC)	360 mL	• 8 g peptone from casein • 4 g yeast extract • ~360 mL MQ water
YP medium 80% (AC)	320 mL	• 8 g peptone from casein • 4 g yeast extract • ~320 mL MQ water
YPD plates (AC)	400 mL	• 8 g peptone from casein • 4 g yeast extract • 6 g Agar-agar • ~360 mL MQ water • 40 mL 200g/L glucose
Glucose 200g/l (AC)	100 mL	• 8g Glucose • ~100 mL MQ water
Xylose 200g/l (AC)	100 mL	• 8g Xylose • ~100mL MQ water
Propidium Iodide 10 µg/mL	10 mL	• 200 µL, 500 µg/mL PI • 9.8 mL MQ water
Sodium Hypochlorite 0.5%	400 mL	• 20 mL NaOCl stock, 10% • ~380 mL MQ water
Sodium Thiosulphate 10 mM	800 mL	• 1.98 g $Na_2S_2O_3 * 5\ H_2O$ • ~800 mL MQ water
PBS pH 7.4 (AC)	1 L	• 8 g NaCl • 0.2 g KCl • 2.68 g $Na_2HPO_4 * 7\ H_2O$ • 0.24 g KH_2PO_4 • ~1 L MQ water
Accuri C6+ Sheath Fluid	1 L	• 1 'BD™ Solution Bacteriostatic Concentrate' bottle • 1 L MQ water

Accuri C6+ Detergent	200 mL	• 3 mL ‘BD^{TM} Detergent Solution Concentrate’ • 197 mL MQ water

Table 2.3 depicts recipes for both pre-culture steps as well as the batch cultivation and feed bottle contents used during the final iteration of the experimental phase. Mediums and sugar solution recipes can be seen above in table 2.2. The over-night culture was reinoculated in the preculture to an initial OD_{620nm} of 0.6, while the batch-cultivation used the preculture to be further reinoculated to an OD_{620nm} of 0.1.

Table 2.3. Depicting the final iteration of recipes for cultures and feeds used during final batch- and fed-batch cultivations. Yeast strains were taken from -80℃ stocks before streaked on YPD plates.

Solution	Final Volume	Recipe
Over-night Culture (YPD)	5 mL	• 4.5 mL YP • 0.5 mL Glucose 200g/L • Yeast cells from fresh YPD plates
Pre-Culture (YPD)	5 mL	• 4.5 mL YP • 0.5 mL Glucose 200g/L • OD_{620nm} 0.6
Batch Cultivation	50 mL	• 30 mL YP 80% • 10 mL Glucose 200g/l • 10 mL Xylose 200 g/l • OD_{620nm} 0.1
Feed	150 mL	• 120 mL YP 80% • 15 mL Glucose 200g/l • 15 mL Xylose 200g/l • 892 µL Furfural
Furfural	100 mL	• 4.312 mL Furfural • 95.688 mL MQ, Sterile water

$$C_1 * V_1 = C_2 * V_2 \tag{a}$$

$$Dilution\ factor_n = \frac{V_0 + V_{feed_n} + V_{pump_n}}{V_0} \tag{b}$$

Equation ‘a’ depicts the equation used to calculate specific volumes when reinoculating the pre-cultures and batch-cultivation through the help of optical density measured by a spectrophotometer at 620 nm in wavelength. Equation ‘b’ depicts the equation used to calculate the dilution factor of ingoing medium and furfural solution during the fed-batch

phase, which is then used to calculate an approximate number of events per sample, should it be undiluted.

3 Result

3.1 FCM data processing

The development of the code started as mentioned in section 2.1 with flowcytometry files from previous cultivations using the TMBRP011 strain (Perruca Foncillas et al., 2023). These files became the base for the fixed gating strategies used throughout the experimental phase.

To ensure that the static theoretical gates were sufficient, program returned results were compared with the same FCS file run in the software FlowJo. At first, the MFI GFP values seemed to differ greatly, the reason for which was subsequently discovered to be different transformation strategies for the raw data. In order to read and review FCS data in Python, the package 'FlowCytometryTools' (Yurtsev and Friedman, 2018) was used. Through giving it the path to an existing FCS file, it can call the data from their columns and transform it directly for easier viewing, along with simplifying further calculations by making said columns easy to access. While transforming FCS data is done in FlowJo as well, 'FlowCytometryTools' transforms the data directly, while FlowJo transforms its axes instead. To fix this discrepancy gates were created with the logarithmic axes in mind, which also removed any heatmapping capabilities from the program due to lack of support when using log axes from the used packages, heatmapping trials resulted instead in large, inaccurate blocks. The gates were then refined and changed by plotting all given datafiles through use of the package 'Matplotlib' (Matplotlib, 2023) and comparing the results with their FlowJo equivalent.

The program starts by removing debris by a static threshold gate for the forward scatter height (FSC-H) which indicate particle size. Particles bellow this threshold is considered debris and removed from further calculations. The remaining entities are then gated by plotting PI (FL3-H) against GFP (FL1-H) levels. A high PI value indicated a cell with a non-intact membrane, and where thereby denoted 'damaged', while the cells deemed intact were separated by fluorescence level into GFP negative and GFP positive. The resulting gates, as well as an example of program output plots and results compared to FlowJo can be seen in figure 3.1. As can be seen, the fixed gates sufficiently encompass the relevant cell populations and removes most of the debris. The MFI GFP was calculated using both GFP positive and GFP negative populations as the value of the culture as a whole was of interest when inducing in the experimental phase, it was also what was done in the study by Perruca Foncillas et.,al. An alternative approach based on the GFP subpopulations was also of interest and functional but was not utilized further in the experimental phase.

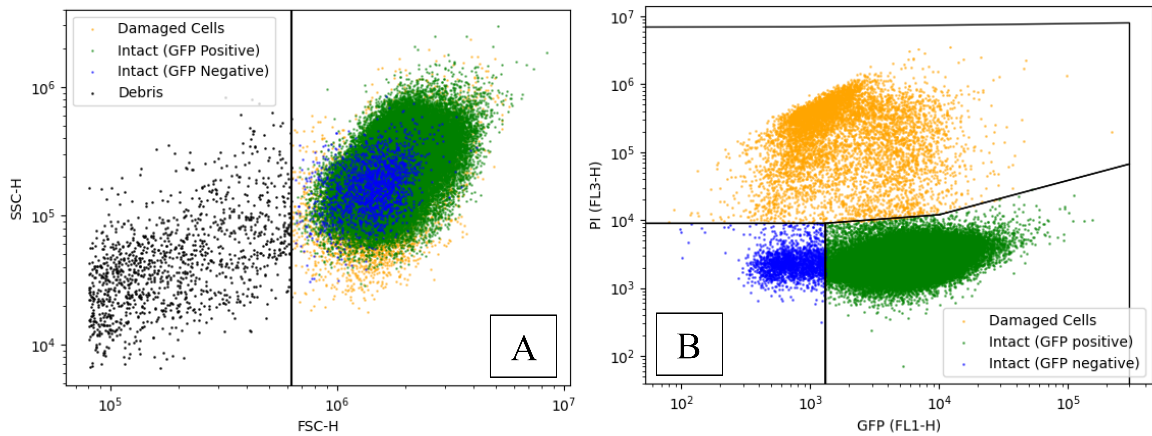

Figure 3.1. Depicting scatterplots of entities recorded in FCS file A01 in folder 2022-11-09_17-00-57 from the cultivations done by Perruca Foncillas et.al. Figure A depicts a forward scatter – side scatter plot indicating their size and granularity. Entities bellow a specified size was deemed debris, where the black line indicates a threshold gate Figure B takes the remaining cell population and gates them based on PI-level and fluorescence level. The black lines indicate an approximate illustration of the applied gates. Yellow scatter indicates cells with non-intact membranes, green scatter indicates GFP positive intact cells and blue scatter indicate GFP negative intact cells.

A statistical comparison of FlowJo and program output from the same FCS files can be seen in figure 3.2. Figures A and B depicts MFI GFP and the percentage of intact cells respectively, with FlowJo output on the x-axes and program output on the y-axes. Linear equations were fitted to the resulting datapoints and their R^2 values were calculated to indicate fit. Figure 3.2 A was fitted with the linear equation y = 1.0035x - 1.8443 with the R^2 value 0.9995, while figure 3.2 B was fitted with the equation y = 1.0481x - 4.9351 with an R^2 of 0.9943. This analysis indicated that the fixed gating strategy was sufficient due to the resulting slopes and R^2 values both being close to 1. An example of exact statistic comparisons for one FCS file can be seen in table 3.1, which in turn corresponds to the entities plotted in figure 3.1.

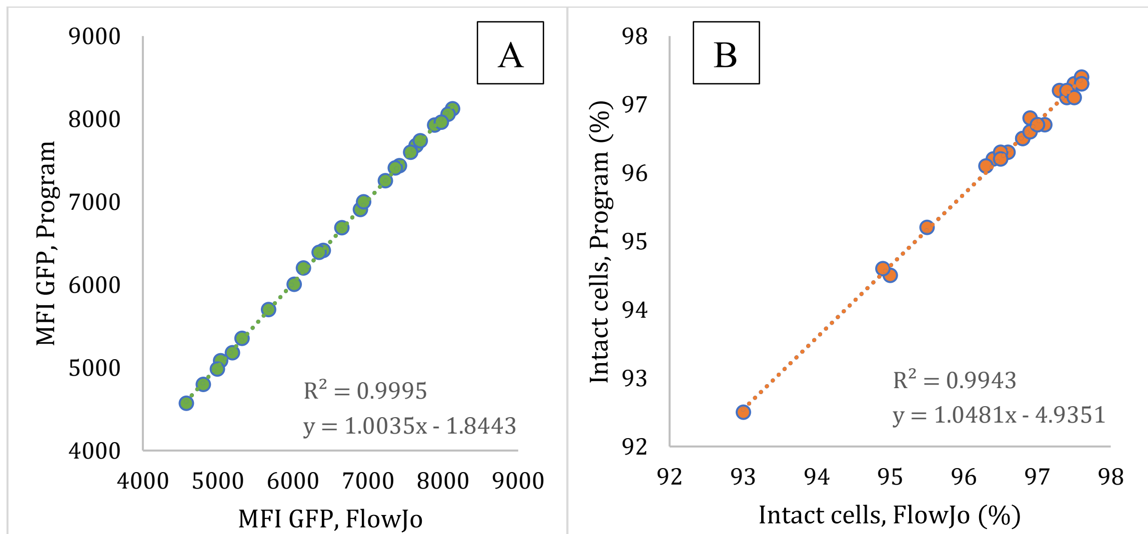

Figure 3.2. Depicting the result of a statistical analysis comparing program automatic output to FlowJo manual output. Figure A depicts MFI GFP, with FlowJo output on the x-axis and program output for the same file on the y-axis. A linear regression was fitted, with the equation y = 1.0035x - 1.8443 and R^2 value of 0.9995. Similarly, the percentage of intact cells were plotted in graph B, with a linear regression equation of y = 1.0481x - 4.9351 with R^2 = 0.9943.

Table 3.1. Depicting an example of data outputs from the program and the FlowJo software when used independently with the same FCS file. Corresponds to figure 3.1 for the program output graphs.

Statistic	Program Output	FlowJo Output
Number of events	97753	97753
Number of cells	96511	95409
Percentage of intact cells	92.6%	93.0%
MFI GFP (intact cells)	6412	6402

3.2 Pump control program

A plan for a final program structure was developed before further coding was done. The FCS Trial program was to be repurposed into a function to be called on by the Main Executor file to return numerical parameters from new FCS files. This program file would also be able to call on the pump handling file to control the pump further down the line.

Firstly, it was necessary that the program could access new FCS files as they were created rather than to be manually directed to said files. This was done with the 'Watchdog' package (Mangalapilly, 2023), specifically using the 'Observer' class. This enabled the code to send an instanced Observer to a specified folder, which can monitor changes within that folder. The 'on_created' method was used, which causes the Observer to react when a file is created in the specified folder. The Observer was also written to continuously monitor the folder until the program is terminated.

The program should also be able to handle different reactor setups according to the user's needs. This was done by initializing the 'file_counter' variable in the on_created method to keep track of the number of created FCS files. For a setup with one reactor and one washing step, the first and subsequently every other file would be of interest and denoted as sample files, which would continue program execution. Separate scripts were created for different reactor-water set-ups, however only the forementioned was employed experimentally.

Initial testing of pump control begun with the help of an Arduino and serial communication. It was demonstrated that manual input of commands into the Arduino terminal had the capability of stopping and starting the pump while connected by USB to a COM-port. To do this automatically with python, the 'Serial' package (Liechti, 2020) was used for COM communication. Serial is able to write commands and read responses from a port, which indicate if the commands are accepted and executed. For specific parameters and commands used, see Appendix I: III Pump Handling. The functions made to control the pump was only usable with the pump attached, hence why the Pump Mock file was created as a substitute for testing, which only prints messages in the console.

As starting and stopping the pump should be based on the slopes of real-time data, it was decided that the slope values should be based on the three latest datapoints, but the issue of possible outliers became apparent. Exponential Moving Average (EMA) was one option that was trialed. EMA is a weighted moving average commonly used when following stock market trends. In stock market applications, EMA "smooths" stock price

fluctuations and enables the user to easier distinguish between actual market trends and standard day-to-day fluctuations (5paisa, 2023). EMA differs from the method Standard Moving Average (SMA) by putting more weight on recent datapoints which makes it faster correcting when determining the current trend for a stock, which is also why it denoted as an exponential average. Historical datapoints have less and less relevance the further back they are. The formula for EMA calculation can be seen in equation 'c', where α is a constant, exponential "smoothing" factor, determining the weight of the newer datapoint, $Data_n$ is the raw data of measurement n.(5paisa, 2023).

$$EMA_n = EMA_{n-1} + \alpha * (Data_n - EMA_{n-1}) \quad \text{(c)}$$

There were several functions written in the Main Executor for further handling of sample files, in order of execution:

I. *Save_results*. This function extracts the data received from the FCS Handling file and transforms them into a singular 'Pandas' data frame with correct column headings and appends new datapoints to the 'Total Results' data frame. It also returns the sample time and calculates and appends the newest EMAs for PI percentage and GFP MFI.

II. *Plot_results*. This is used for plotting and is not strictly necessary for code function but is used as a visualization tool for the user. It plots PI percentage, PI EMA, GFP MFI, GFP MFI EMA, and recorded events in three separate subplots.

III. *Get_slope*. If this is at least the third sample file, this function calculates the EMA slopes for the last three datapoints of PI percentage and MFI GFP when plotted against time. It also appends these slopes to the 'Total Results' data frame.
If not, it appends "Premature" to those columns in the 'Total Results' data frame.

IV. *Use_pump*. This function is used to call functions from the Pump Handling file to control the pump, specifically pump 2 in figure 2.1. If it is at least the third sample file, this function utilizes several 'if' statements to decide if the pump should start, and how many mL/h to pump. The pump will stop if the PI slope is higher than a certain positive threshold or average PI percentage of the last three points is higher than a static number. It will start if it hasn't been told to stop and the MFI GFP slope is lower than a certain negative threshold, the first setting currently pumps 1 mL/h.

The volume per hour will subsequently increase if the PI slope does not increase and the MFI GFP slope is continuously negative. A higher mL/h in this case indicates a setting where the pump is left on for a longer period of time within a sample cycle, although always split into two stages. A higher pumping time will lead to more furfural entering the cultivation vial, which in turn should have a higher probability of reinducing the cells, but may also risk their integrity if done immediately, hence the tiered setting system based on the MFI GFP slope.

V. *Save_to_folder*. This is the function used by 'atexit' to save the output figure and data frame as a PNG and excel file respectively when the program is shut down. It also creates a new folder based on the start time of the program where it places the files.

The final program flowchart can be seen in figures 3.3.1 and 3.3.2. This illustrates the flow of events as an FCS file is created in de designated folder to when the program waits for the next file. This will then repeat until program termination, which is when the data will be saved by the *Save_to_folder* function. The incremental increase of furfural volume entering the culture vial is based on increased minutes pumping dispersed throughout a sample cycle, see section *2.3.2 Pump Handling* for specifics.

An example of a final program output graph can be seen in figure 3.4. This displays a test run done with data from the cultivation by Perruca Foncillas et al., with datapoint 0 being the file used to create figure 3.1 and table 3.1 with the FCS Trial program.

An example of program output data frame can be seen in table 3.2. This is the numerical representation of figure 3.4. As can be seen this simulated test-run was successful in controlling the pump based on flowcytometry data.

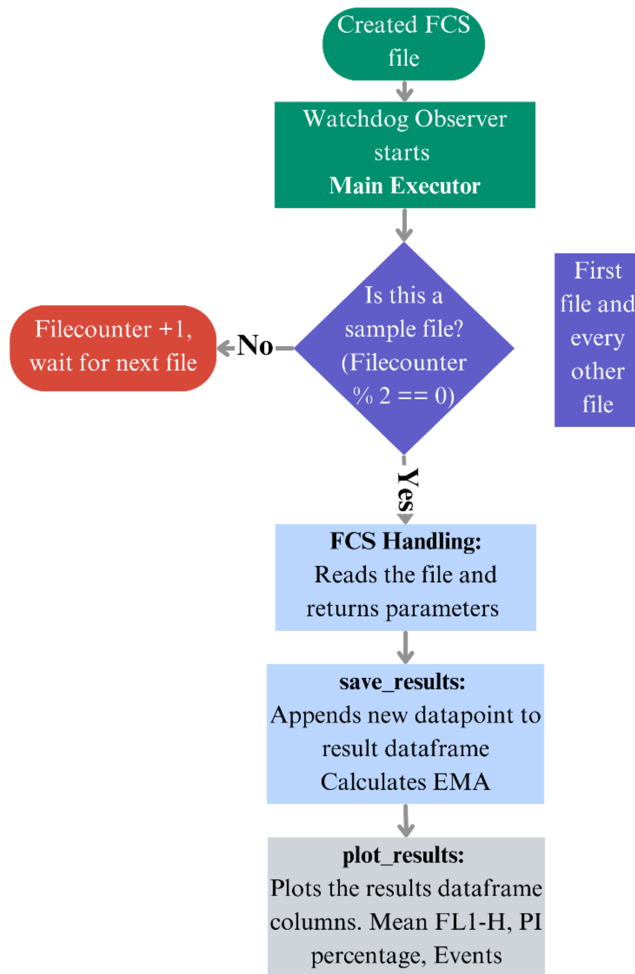

Figure 3.3.1. Depicting the first part of program execution as a new FCS file is created in a designated folder. The program checks whether the file is a sample or water file through the filecounter variable which is increased by 1 for each created file. The FCS Handling file is then called to extract data and calculate PI percentage and MFI GFP which it passes back to Main Executor. These results are then saved, and the EMA is calculated through the save_results function, these are subsequently plotted by the plot_results function.

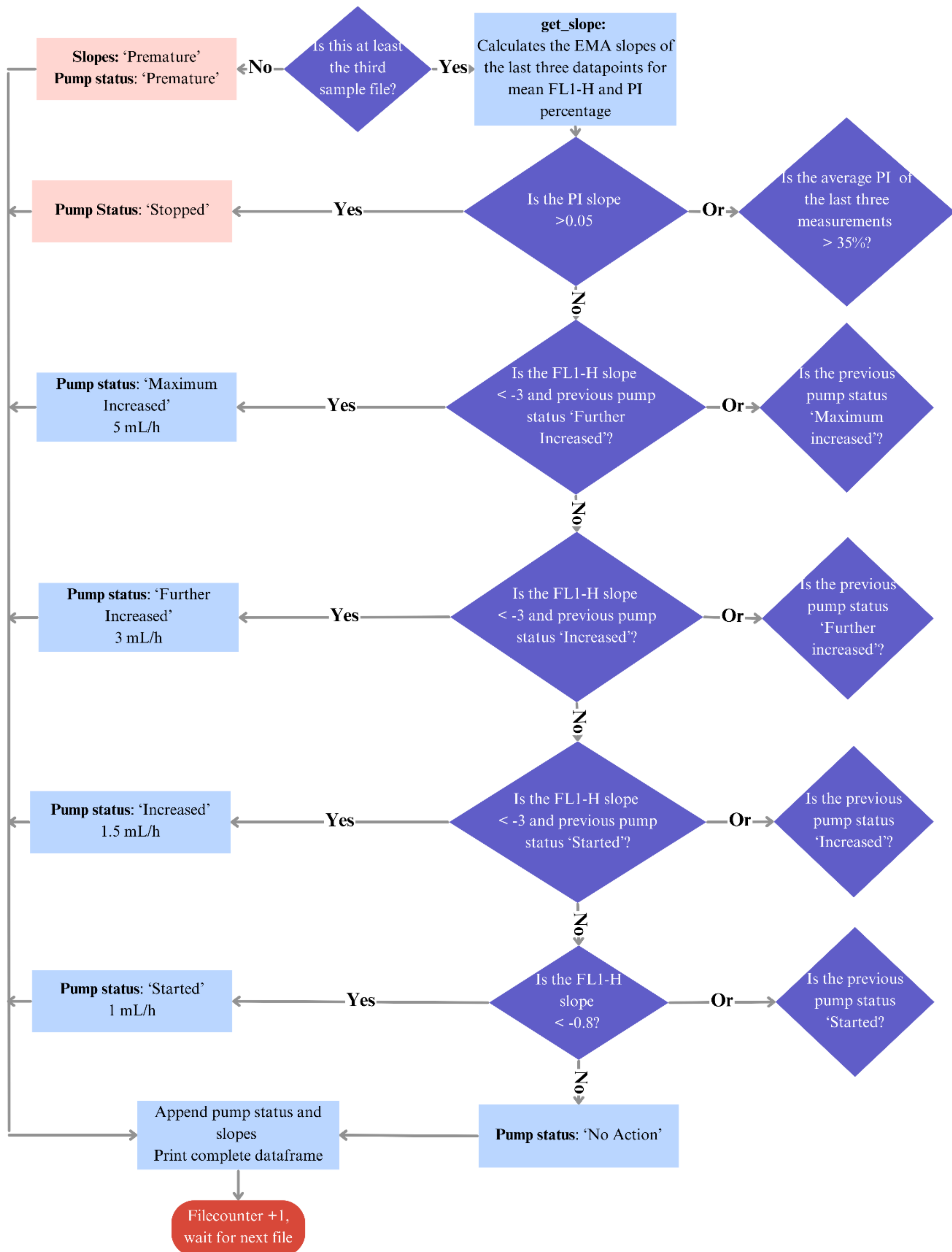

Figure 3.3.2. Depicting the second part of program execution after an FCS file has been created. If it is at least the third sample file, slopes are calculated which subsequently determines whether the pump should start, stop, or take no action. This represents actual program set up as the pumping volume is increased if the fluorescence continues to decrease after the pump is started. The different pumping rates is built on time that the pump is on in a sample cycle, so a higher rate means a longer pumping time per hour.

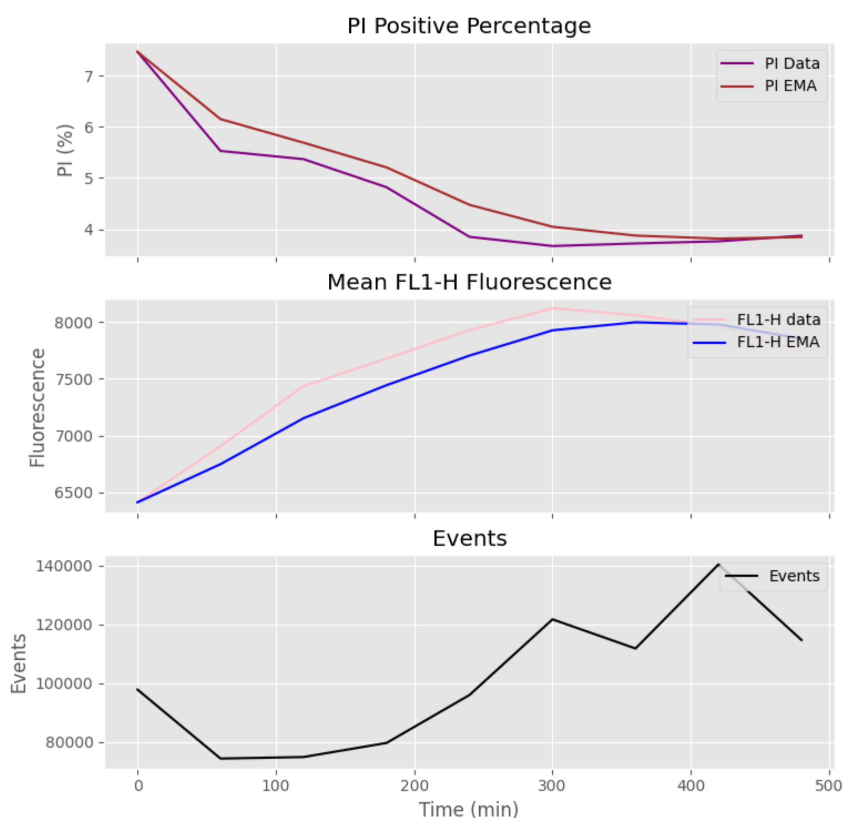

Figure 3.4. Depicting the final output graph for a theoretical test run done by moving existing FCS files to a designated folder. The three subplots represent number of events. Mean fluorescence and its EMA, and PI-stained percentage and its EMA. The figure is based on the data written in table 3.2.

Table 3.2. Depicting the final output data frame for a theoretical test run done by moving existing FCS files to a designated folder. The pump status column is based on the slopes, both PI percentage and MFI GFP is decreasing the pump is designed to start which can be seen in the 8th datapoint.

	PI percentage	MFI GFP	Time (min)	Events	PI slope (EMA)	GFP slope (EMA)	Pump Status
0	7.462053	6411.979	0	97 753	Premature	Premature	Premature
1	5.528853	6906.347	60	74 267	Premature	Premature	Premature
2	5.3683	7433.694	120	74 791	-0.01477	6.163345	No Action
3	4.820228	7675.728	180	79 583	-0.00785	5.78858	No Action
4	3.85195	7925.706	240	95 917	-0.01011	4.593783	No Action
5	3.673066	8119.962	300	121679	-0.00966	4.023424	No Action
6	3.722865	8058.241	360	111751	-0.005	2.438455	No Action
7	3.764887	7959.524	420	140391	-0.00193	0.430083	No Action
8	3.873295	7738.315	480	114649	-0.00025	-1.20386	Started

3.3 Results of the evaluation of base program functionality

The program was able to start and stop the pump based on manually created samples utilizing the TEF and TRX strains (table 3.3). At first the program was unable to recognize file creations in the OnCyt's own designated output folder, however this was solved by changing the output folder to one not directly managed by the OnCyt. Some waiting times were also introduced to the program (*time.sleep*) as it was accessing the files too quickly after creation, making them appear corrupted to the program while essentially incomplete. The time column was in this case artificial to generate comparable slope values to those the program was built on.

Table 3.3. Depicting program data output from the initial trials. As MFI GFP and PI percentage changed the program was able to recognize these changes and control the pump successfully. The Time column was in this case artificial.

	PI percentage	MFI GFP	Time (min)	Events	PI slope (EMA)	GFP slope (EMA)	Pump Status
0	3.401176	5974.791	0	11413	Premature	Premature	Premature
1	2.078148	6753.514	30	16212	Premature	Premature	Premature
2	2.765731	14472.23	60	14195	-0.01239	87.02383	No Action
3	0.424165	29220.53	90	15785	-0.01806	244.7067	No Action
4	2.264454	15621.46	120	16449	-0.01304	116.4891	No Action
5	5.070156	9984.896	150	11774	0.035935	-122.758	Started
6	23.58696	6756.726	180	1251	0.204825	-135.05	Stopped

3.4 Batch cultivations for determination of glucose consumption rate

The batch cultivations were run to determine how fast the cells would consume the glucose present in the medium, which would indicate when to start the feeding for the fed-batches. They were also run to evaluate the system setup with YP medium, and shake-flask as opposed to minimal medium and bioreactor which was used in the study by Perruca Foncillas et.al. While a triplicate of the batch cultivation was run, due to mechanical errors with the flow cytometer which stopped some measurements early from the end of the second run onwards, event count is non reliable from that point. However, MFI GFP and PI percentage remain consistent throughout and hence seemed reliable. As no duplicate was available for data after the 16h mark, the existing event count data will not be used for further calculations.

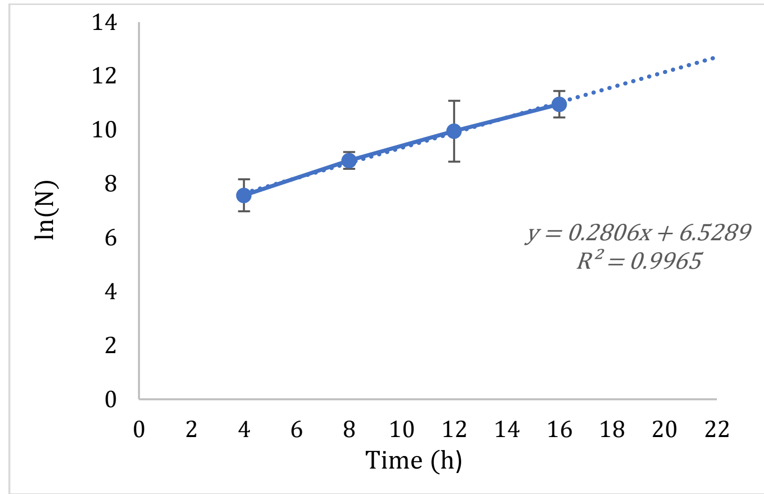

Figure 3.5. Depicting results from the primary batch cultivations. Displaying the natural logarithm of recorded events from 4 to 16 hours and its linear equation y = 0.2806x + 6.5289 with an R^2 value of 0.9965. The linear equation was used to calculate μ_{max} and T_b from equations 'd' and 'e'.

$$ln(N) = \mu_{max} * t + ln(N_0) \quad \text{(d)}$$

$$T_g = \frac{ln(2)}{\mu_{max}} \quad \text{(e)}$$

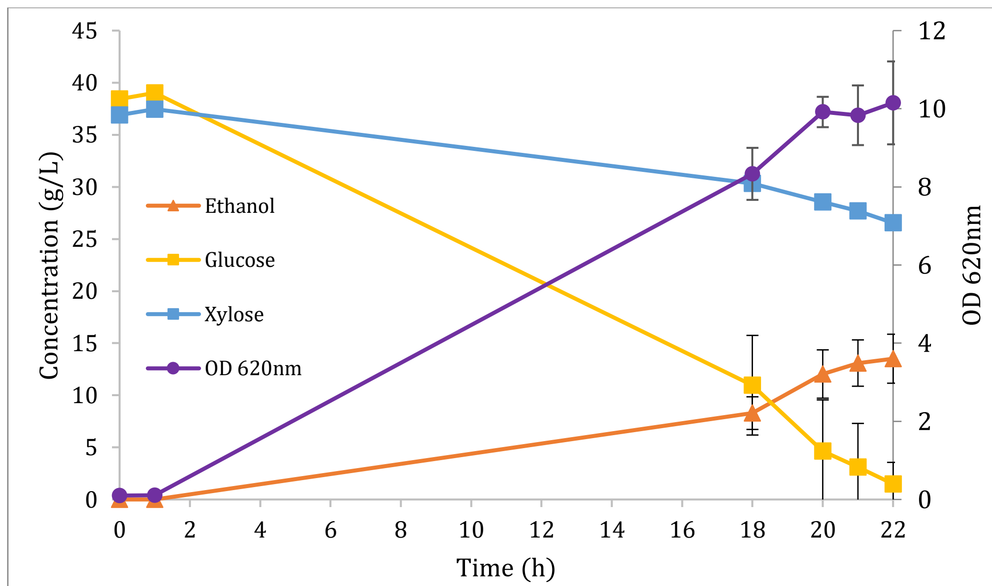

Figure 3.6. Depicting the HPLC analysis results for the batch cultivation mediums, as well as the recorded OD_{620nm}. Glucose continuously decreases and depletes from 21-22 hours on average with a maximum depletion time of 23.5 hours. The xylose consumption rate increases slightly as the glucose concentration decreases. There was some ethanol formation throughout the cultivation. OD_{620nm} increases gradually until the last three hours.

Figures 3.5 and 3.6 showcases the results from the batch cultivations. Figure 3.5 shows the natural logarithm of recorded events as functions of time. A linear equation was fitted to the recorded event graph from hour 4 to 16, which had the numerical values y = 0.2806x + 6.5289 with an R^2 value of 0.9965. This linear equation corresponds to equation 'd', where the maximum specific growth rate (μ_{max}) corresponds to the slope,

which equates to a value of 0.2806 h^{-1}. This was then used with equation 'e' to calculate the doubling time (T_g) to 2 hours and 28 minutes.

Figure 3.6 shows the HPLC results of the cultivations as well as their OD_{620nm} measurements. Glucose was observed to deplete on average at the 21–22-hour mark with a maximum depletion time of 23.5 hours, these times were used to plan for the fed batch-cultivations as the feeding phase was to be started after batch phase depletion. After glucose depletion there was a slight increase in the xylose consumption rate. Some formation of ethanol was observed. OD_{620nm} throughout the experiment, with a slight stagnation observed during the last three hours, corresponding to the lag-phase induced by the low sugar-content within the medium, along with the diauxic shift to xylose and ethanol.

3.5 Evaluation of sampling procedures

The batch cultivations were monitored by automatically sampling every 4 hours and running over 3 mL of dead volume before measuring to ensure fresh sample on the line, due to the length of the OnCyt sampling line being close to 1.5 meters. This was not ideal as a lot of volume was wasted of the 50 mL cultivation, thus making more frequent sampling impossible as well. For the pump control to work as intended later on, this four-hour window was deemed much too wide as letting the pump run unregulated for that long could result in killing the entire culture, and small cell clusters were noticeable in the line further into the cultivation which could potentially cause clogging in the SIP of the flow cytometer. A solution was developed where an air filter (0.22 µm) was put on the OnCyt's air intake, and the autosampler would use sterile-filtered air to push the still on the line back into the shake flask after a taken sample, resulting in a new dead volume of approximately 150 µL in the OnCyt. During the subsequent trials, the batch cultivations were closely monitored for contamination through the FCS Trial program, but none were noticed. This reduced the needed volume significantly and enabled sampling every 30 minutes during the fed-batch cultivations.

3.6 Comparisons between fed-batches and dynamically controlled fed-batches

Comparisons between the results from the regular fed-batch cultivations and the dynamically regulated fed-batch cultivations can be seen in figures 3.7 and 3.8.

Figure 3.7 features average PI percentage and MFI GFP plotted against time passed since the feeding phase was initiated. As can be seen in 3.7 A, PI percentage followed a similar profile for both the experimental setups, with a value of approximately ten percent towards the end of the cultivations. The impact of the dynamic control program can mainly be observed in figure 3.7 B, where the experimental setups follow a similar profile until the MFI GFP slope starts decreasing after the first peak at seven hours. The regular fed-batch cultivations approximately followed the profile shown in the study by Perruca Foncillas et.al., 2023, despite the differing medium and cultivation differences. The decrease in MFI GFP in the dynamically controlled fed-batches caused program activation and furfural addition, which caused the cells to be reinduced to a higher peak before decreasing again after 17 hours of feeding.

Up until the second peak, the program was kept in the 'Started' pump status, which means an addition of 0.5 mL furfural solution per half-hour sample cycle. Subsequently the volume per sample cycle was increased, but no further induction was observed with current pump settings and furfural concentrations. The regular fed-batch cultivations were administered 892 µL or 1.031 g of furfural in total from the feed at the end of the cultivations, while the dynamically regulated fed-batch cultivations had a final furfural addition of 1.97 mL ± 0.046 mL or 2.29 g ± 0.053 g. This equates to an increase of over 120%, while the percentage of PI-stained cells is steadily decreasing comparatively to the regular cultivations.

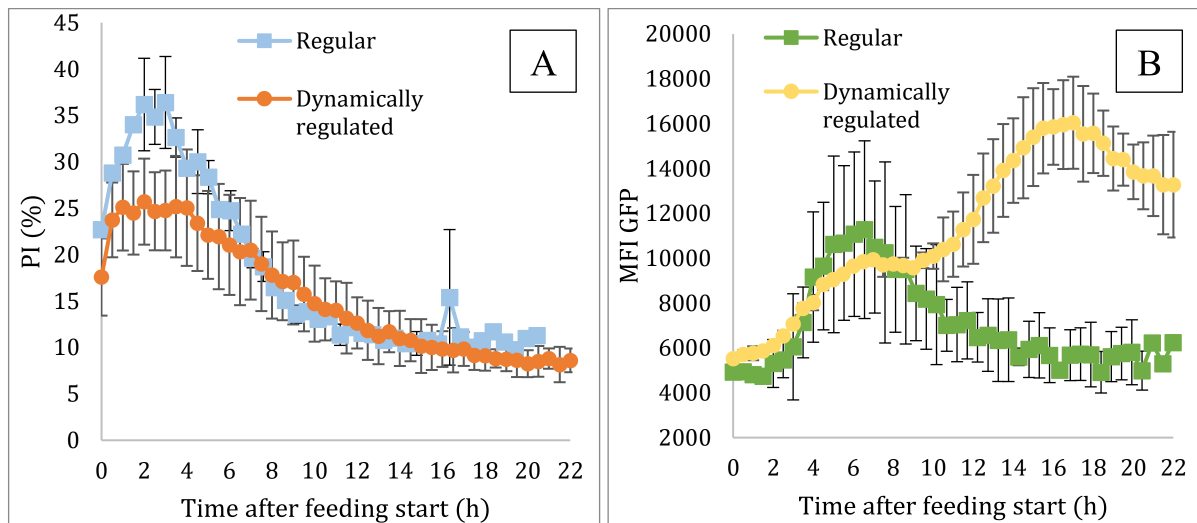

Figure 3.7. Illustrating data from the duplicates of the regular fed-batch cultivations and the dynamically controlled fed-batch cultivations. 3.7 A shows how the percentage of PI-stained cells changed throughout the feeding phase, with the regular fed-batch in blue and the dynamically regulated fed-batches in orange. 3.7 B shows changes in MFI GFP over time during the feeding phase with the regular fed-batches in green and the dynamically regulated fed-batches in yellow.

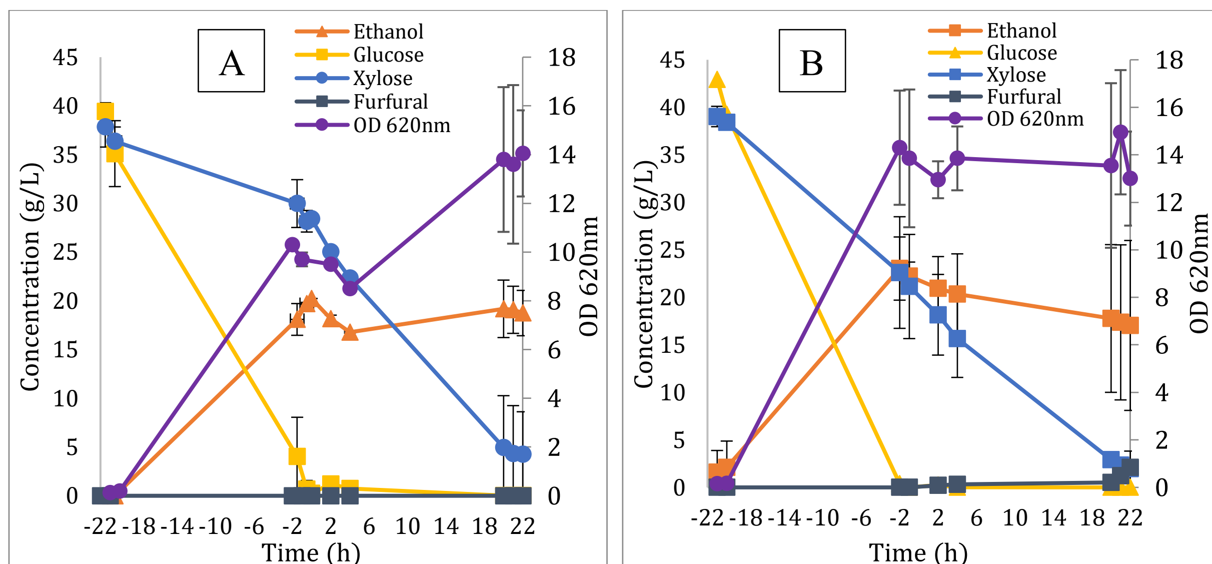

Figure 3.8. Showing the HPLC results for the duplicate regular fed-batches in 3.8.A, and the duplicate dynamically regulated fed-batches in 3.8 B, along with their calculated undiluted OD_{620} values. Hour 0 represents the transition into the feeding phase from the batch phase.

Figure 3.8 shows the medium HPLC analysis results of the fed-batch cultivations. As can be seen, the profiles for the regular cultivations (3.8 A) and the dynamically controlled cultivations (3.8 B) appear similar, the most notable difference during the feeding phase being some furfural accumulation within the medium. This is due to the increased administration rate of the dynamically regulated fed-batch cultivations, especially during the last three samples where the program was running at its 'Maximum increased' setting, administrating 5 mL/h furfural solution or 0.2 mL per sample cycle pure furfural into the cultivation vials. The measured OD_{620} values differ in profile, which can be explained by the extra dilution stemming from the added furfural solution.

4 Discussion

Here, a method for monitoring and regulating a yeast cultivation process was developed at the single cell-level. The program has been utilized successfully for cultivation process control and was able to reliably control a peristaltic pump based on specific input parameters, while interpreting FCS files in real-time as they were created. This single-cell level analysis and on-line process control has the advantages of being less labor-intensive while being able to regulate a microbial population based on several parameters that is otherwise difficult to analyze automatically in an on-line setup. The redox biosensor utilized in the TMBRP011 *S. cerevisiae* strain, in combination with PI staining was but one example of a potential control scheme one could use.

The dynamically regulated fed-batch cultivations were reinduced after the initial decrease in fluorescence when the program was employed. This resulted in final added furfural volumes over 120% more than the regular fed-batch cultivations, indicating acclimatation from the cells, as even the additional furfural injected eventually could not keep the cells induced. After the second peak at 17 hours of feeding, the program did increase its pumping time as instructed, but it seemed that neither 1.5, 3, or 5 mL/h were able to completely reinduce the culture. More experiments are necessary to determine whether further reinduction is possible, what can be said is that the PI percentages did not change with the higher volumes, indicating that the culture can take a higher volume in terms of survivability. Biomass production also seemed comparable to the regular fed-batch cultivations when accounting for dilution factor, which indicates that the additional furfural addition did not affect the growth of the cells.

As for accuracy, the final product and the gating strategy has been sufficient throughout and has not presented any issues that hindered project progression. If one wanted to utilize this program with a less static population a different strategy would be necessary, but as the static strategy was sufficient here this was not explored. One strategy that could be employed in this case is k-means clustering (Rao NS, 2023), which can automatically select subpopulations of FCM data and gate based on clustering. This method will be a useful alternative to the fixed gating strategy employed when building this program and could assist in making the program usable for more microbial populations without manually setting the gates beforehand.

The unregulated fed-batch cultivation showed a similar profile to those presented by Perruca Foncillas et., al. The initial increase in the percentage of cells that were PI stained could potentially be attributed to the cells entering a semi-stationary phase due to the diauxic shift, which could also be a reason for the apparent ethanol consumption before stabilization. Furthermore, the increase of MFI GFP seems to coincide with the negative PI percentage slope which coupled with the slight glucose accumulation during the first hour of feeding further gives credit to this theory. The cultivations were run aerobically, however as OD_{620nm} indicates such high cell concentrations and that there was no active air transfer, the oxygen contents towards the end of the cultivations may not have been sufficient, contributing to this phenomenon. Overall aeration may also have been limited in the shake flask setup. As no accumulated furfural was observed, it can be assumed that the cells were able to detoxify the entirety of it towards the end of the cultivation.

The batch cultivations were plagued by a mechanical error with the flow cytometer, which was later discovered to be a rusted waste outlet valve. This error made many event counts unusable, including those of the regular fed-batches. This makes biomass comparisons more difficult as relying on optical density measurements solely has a high level of variability. However, while not optimal, this was deemed as an error this particular setup could work with. As the most important statistics consists of a mean and a percentage, these can still be calculated even with smaller sample sizes. In the event of a crash, the cytometer would stop the measurement after approximately 16 seconds as opposed to the minute a full sample would take. Especially at higher cell concentrations, even 16 seconds yielded more than enough cells to make the extrapolation that the sample would follow a similar pattern.

5 Conclusion

The cultivations have proven reproducible, and the program design has fulfilled the initial scope, along with some additional features. The regulated fed-batch cultivations were successful in re-inducing the culture and outside of mechanical flow cytometer issues the project has delivered and proceeded as planned. Flow cytometry as a tool for on-line process control is in its early stages, but the result of this project proves that there are great possibilities for customization for specific processes, and that it is most certainly a viable possibility for the future. This project provides a roadmap for implementing on-line FCM monitoring as a basis for process control.

6 Future outlook

It was discovered late in the experimental phase that while the code is functional, the necessary downgrades from a windows 11 computer to a windows 7 computer presented some not-so-apparent issues. This is common when downgrading and brought with it difficulties when implementing new features such as an automatic save function on program exit. The downgrade was necessary due to the computer connected to the Accuri C6+ needed to be on windows 7 due to software license compatibility. Thus, for the program to work as it does on newer machines, some refactoring may be necessary. That is not to say that the program does not work, it just one less feature than what was desired

to be included further into the process. This could also be fixed by the use of a virtual machine or upgrading the windows version on the computer. There were also some threading issues with Matplotlib and Watchdog as well as atexit wanting to be executed by the main thread, which likely is the cause of some of these issues and might require a more thorough refactoring to fix on an older system. There is also a possibility of a Watchdog memory leak as the Observer function is running until the program is terminated.

While a warning may be given by Matplotlib on newer systems, no actual effect to code functionality has been observed, and it is able to execute atexit with exitcode 0, indicating a smooth exit without issues. This is as opposed to exitcode -1 which is given on the older system, indicating some sort of issue with termination, likely from a Watchdog issue as mentioned above. Fixing these would be the first step taken to improve on the project in the future.

This code is realistically only a back-end in its current state, as the user has to run the code directly from the IDE which is not optimal. The next step would be to develop a front-end and figure out a Backend – API – Front-end combination that would be functional and user friendly. This would likely take as long as the development of the backend itself if not longer.

Experimentally, there are several things one could test. What is the maximal furfural rate that the further induced cells can take? What is the maximum induction level possible, can it be reached in a multi-step increase of furfural insertion? Can the cells be kept at that fluorescence over a long period of time by further increasing furfural insertion, without an increase in PI percentage? Will this have a positive effect on biomass production? What are the optimal threshold values for slopes and percentages? What are other uses for this type of automation? The questions to be answered are many and may have significant effects on bioethanol production from lignocellulose in the future. The next step would be to move to a bioreactor with minimal medium to ensure similar reactivity from the cells.

7 Popular Scientific Summary

This project had the aim of developing a Python program capable of controlling a pump containing a toxic substrate based on on-line analysis of a yeast cultivation on a single-cell level. Yeast cultivation is used within a number of industries, including foods, pharmaceuticals, and fuels. The program was specifically developed with a yeast strain used for bioethanol production from lignocellulose in mind, which has a partial resistance to this toxic substrate.

The program was developed to be used for process control by monitoring the trends of two separate statistics which indicated cell viability, and induction level due to the stress caused by the toxic substrate respectively. The program was designed to start the pump when the induction level decreased, which for the specific yeast-strain used indicated an adaptation to the rate of the toxic substance present in a constant feed, the separate pump would then inject a higher concentration of the toxic substance to further induce the cells as they addapt. The reason for this was that a higher induction level had previously been linked to a higher production rate of bioethanol when using the same yeast strain.

Cell viability was monitored and a decrease in this statistic would indicate that the culture was not able to handle the amount of the toxic substrate injected without cell damage, which is why the increase of the volume of the toxic substrate injected into the culture was designed to be gradual. Should this statistic decrease, the pump was designed to stop and allow the cells to acclimatize further.

This project was furthermore done to automize a cultivation process through single cell monitoring by using flow cytometry data. This is a relatively novel concept that has not been applied previously. Basing process control on flow cytometry data enables many options for monitoring that is not available traditionally, as it is currently done on a culture-wide basis with less precise tools. This method could introduce more precise control systems and enable the use of specifically genetically engineered yeast strains for large-scale processes, without the need for additional analysis steps. Automation as a whole is also needed within the life-science space to enable researchers to develop more elaborate experiments that are less laborious, while also reducing sources of error.

The program was created as planned and was able to cause a reinduction of the yeast culture automatically without user input during the experimental phase, based on induction level and cell viability. The amount of toxic substance that could be inserted into the culture increased by over 120% with the program as opposed to without, and these results proved reproducible. The outcome of this thesis provides a roadmap for the implementation of single cell monitoring as a basis for bioprocess control.

References

5PAISA. 2023. *Learn How To Calculate The Exponential Moving Average (EMA)* [Online]. Available: https://www.5paisa.com/stock-market-guide/stock-share-market/exponential-moving-average-ema [Accessed 20-06 2023].

AKSHAY, R. M., ASHISH, P., ARINDAM, M. & PANT, K. K. 2021. Pretreatment of lignocellulosic biomass: A review on recent advances. *Bioresource Technology,* 334**,** 125235.

ALMEIDA, J. R., RÖDER, A., MODIG, T., LAADAN, B., LIDÉN, G. & GORWA-GRAUSLUND, M. F. 2008. NADH- vs NADPH-coupled reduction of 5-hydroxymethyl furfural (HMF) and its implications on product distribution in Saccharomyces cerevisiae. *Appl Microbiol Biotechnol,* 78**,** 939-45.

ASK, M., BETTIGA, M., MAPELLI, V. & OLSSON, L. 2013. The influence of HMF and furfural on redox-balance and energy-state of xylose-utilizing Saccharomyces cerevisiae. *Biotechnol Biofuels,* 6**,** 22.

BESMER, M. D., EPTING, J., PAGE, R. M., SIGRIST, J. A., HUGGENBERGER, P. & HAMMES, F. 2016. Online flow cytometry reveals microbial dynamics influenced by concurrent natural and operational events in groundwater used for drinking water treatment. *Scientific Reports,* 6**,** 38462.

CODELITE LTD, U. 2023. *Chemix* [Online]. Available: https://chemix.org [Accessed 2023].

COMMUNITY, T. S. 2023. *Statistical functions (scipy.stats)* [Online]. Available: https://docs.scipy.org/doc/scipy/reference/stats.html [Accessed 2023].

DRESCHER, H., WEISKIRCHEN, S. & WEISKIRCHEN, R. 2021. Flow Cytometry: A Blessing and a Curse. *Biomedicines,* 9**,** 1613.

FARRELL, A. E., PLEVIN, R. J., TURNER, B. T., JONES, A. D., O'HARE, M. & KAMMEN, D. M. 2006. Ethanol can contribute to energy and environmental goals. *Science,* 311**,** 506-8.

FERNANDES, R. L., CARLQUIST, M., LUNDIN, L., HEINS, A.-L., DUTTA, A., SØRENSEN, S. J., JENSEN, A. D., NOPENS, I., LANTZ, A. E. & GERNAEY, K. V. 2013. Cell mass and cell cycle dynamics of an asynchronous budding yeast population: Experimental observations, flow cytometry data analysis, and multi-scale modeling. *Biotechnology and Bioengineering,* 110**,** 812-826.

FOUNDATION, P. S. 2023a. *atexit — Exit handlers* [Online]. Available: https://docs.python.org/3/library/atexit.html [Accessed 2023].

FOUNDATION, P. S. 2023b. *datetime — Basic date and time types* [Online]. Available: https://docs.python.org/3/library/datetime.html#module-datetime [Accessed 2023].

FOUNDATION, P. S. 2023c. *An Overview of Packaging for Python* [Online]. Available: https://packaging.python.org/en/latest/overview/ [Accessed 2023].

FOUNDATION, P. S. 2023d. *time — Time access and conversions* [Online]. Available: https://docs.python.org/3/library/time.html [Accessed 2023].

HAHN-HÄGERDAL, B., GALBE, M., GORWA-GRAUSLUND, M. F., LIDÉN, G. & ZACCHI, G. 2006. Bio-ethanol – the fuel of tomorrow from the residues of today. *Trends in Biotechnology,* 24**,** 549-556.

HOLLAND, I. & DAVIES, J. A. 2020. Automation in the Life Science Research Laboratory. *Front Bioeng Biotechnol,* 8**,** 571777.

LIECHTI, C. 2020. *pySerial Overview* [Online]. Available: https://pyserial.readthedocs.io/en/latest/pyserial.html [Accessed 12-06 2023].

MANGALAPILLY, Y. 2023. *watchdog 3.0.0* [Online]. [Accessed 06-06 2023].

MATPLOTLIB. 2023. *Matplotlib 3.7.2 documentation* [Online]. Available: https://matplotlib.org/stable/index.html [Accessed 10-05 2023].

NILSSON, A., TAHERZADEH, M. J. & LIDÉN, G. 2001. Use of dynamic step response for control of fed-batch conversion of lignocellulosic hydrolyzates to ethanol. *J Biotechnol,* 89**,** 41-53.

NUMFOCUS, I. 2023. *pandas documentation* [Online]. Available: https://pandas.pydata.org/docs/ [Accessed 2023].

PERRUCA-FONCILLAS, R., DAVIDSSON, J., CARLQUIST, M. & GORWA-GRAUSLUND, M. F. 2022. Assessment of fluorescent protein candidates for multi-color flow cytometry analysis of Saccharomyces cerevisiae. *Biotechnol Rep (Amst),* 34**,** e00735.

PERRUCA FONCILLAS, R., SANCHIS SEBASTIÁ, M., WALLBERG, O., CARLQUIST, M. & GORWA-GRAUSLUND, M. F. 2023. Assessment of the TRX2p-yEGFP Biosensor to Monitor the Redox Response of an Industrial Xylose-Fermenting Saccharomyces cerevisiae Strain during Propagation and Fermentation. *Journal of Fungi,* 9**,** 630.

POONTAWEE & LIMTONG 2020. Feeding Strategies of Two-Stage Fed-Batch Cultivation Processes for Microbial Lipid Production from Sugarcane Top Hydrolysate and Crude Glycerol by the Oleaginous Red Yeast Rhodosporidiobolus fluvialis. *Microorganisms,* 8**,** 151.

RAO NS, E. L. L., TOMASSON J, TULLBERG C, BRINK DP, PALMKRON SB, VAN NIEL EWJ, HÅKANSSON S AND CARLQUIST M 2023. Non-inhibitory levels of oxygen during cultivation increase freeze-drying stress tolerance in Limosilactobacillus reuteri DSM 17938. *Frontiers in Microbiology*.

RICCARDI, C. & NICOLETTI, I. 2006. Analysis of apoptosis by propidium iodide staining and flow cytometry. *Nature Protocols,* 1**,** 1458-1461.

SHAJII, A., NUMANAGIĆ, I., LEIGHTON, A. T., GREENYER, H., AMARASINGHE, S. & BERGER, B. 2021. A Python-based programming language for high-performance computational genomics. *Nat Biotechnol,* 39**,** 1062-1064.

YURTSEV, E. & FRIEDMAN, J. 2018. *FlowCytometryTools* [Online]. Available: https://eyurtsev.github.io/FlowCytometryTools/ [Accessed 10-05 2023].

ZHANG, J., SONNENSCHEIN, N., PIHL, T. P. B., PEDERSEN, K. R., JENSEN, M. K. & KEASLING, J. D. 2016. Engineering an NADPH/NADP⁺ Redox Biosensor in Yeast. *ACS Synthetic Biology,* 5**,** 1546-1556.

Appendix 1

i. FCS Trial

```
import FlowCytometryTools as fct
import matplotlib.pyplot as plt
from pylab import *

# Initializes the folder and file used, attributes the data to
the 'sample' variable
file_path = r"C:\Users\ADMIN\Desktop\Saras
scripts\Data\Goal\2023-08-08_14-00-55\H01.fcs"
sample = fct.FCMeasurement(ID='Test 1', datafile=file_path)

# OBS! The flourochromes listed are factory settings and are
needed to call the data from their channels, they do
# not necessarily correspond with the flourochromes used in the
sample.
# Data can only be called from the first 'name' (e.g. FITC-A
not FL1-A for this setup)

# Event,            Number of events in measurement
# FSC-A,            Forward scatter area -> cell size
# SSC-A,            Side scatter area -> cell granularity
# FITC-A::FL1-A     Green flourochrome
# PE-A::FL2-A,      Orange-Red flourochrome
# PerCP-A::FL3-A,   Red flourochrome
# APC-A::FL4-A,     Far-red flourochrome
# FSC-H,            Forward scatter height -> additional size
info
# SSC-H,            Side scatter height -> additional granularity
info
# FITC-H::FL1-H,    Green intensity at its highest point
# PE-H::FL2-H,      Orange-red intensity at its highest point
# PerCP-H::FL3-H,   Red intensity at its highest point
# APC-H::FL4-H,     Far-red intensity at its highest point
# Width,            Pulse width
# Time              Time for recording of each event

# Prints channel names in the sample
# print("Channel names:")
# print(sample.channel_names)

# Transforms the relevant columns from the sample through log,
also attributes the number of entities to nmb_ent
# transformed_sample = sample.transform('hlog', channels=['FSC-
H', 'SSC-H', 'FITC-H', 'PerCP-H', 'FSC-A', 'SSC-A']
# , b=500.0)
nmb_ent = sample.data.shape[0]

# Initializes the gates based on graphical analysis. Values
correspond to corners in a polygon and should be changed
# to fit the particular dataset of interest. The cleanup gate
```

```
is a threshold for size inclusion.
cleanup_gate_x = fct.ThresholdGate(630000, 'FSC-H',
region='above')
cleaned_sample = sample.gate(cleanup_gate_x)
nmb_cells = cleaned_sample.data.shape[0]

debris_gate_x = fct.ThresholdGate(630000, 'FSC-H',
region='below')
debris = sample.gate(debris_gate_x)
nmb_debris = debris.data.shape[0]

intact_gate_pos = fct.PolyGate([(1300, 50), (1300, 9000),
(400000, 170000), (350000, 50)],
                               ('FITC-H', 'PerCP-H'),
region='in', name='Intact Cells OG')

intact_gate_neg = fct.PolyGate([(0, 50), (0, 9000), (1300,
9000), (1300, 50)],
                               ('FITC-H', 'PerCP-H'),
region='in', name='Intact Cells OG')

damaged_gate = fct.PolyGate([(0, 9000), (1000, 7e+6), (260000,
8e+6), (400000, 170000), (1300, 9000)],
                            ('FITC-H', 'PerCP-H'), region='in',
name='Damaged cells OG')


# - Gates data from the sample and counts number of cells
contained in each gate.
# - Calculates the percentage of PI stained cells in the sample
(FL3-A).
# - Calculates the mean FL1-H fluorescence for the non-stained
cells and GFP positive cells.
# - Prints total number of entities before and after gating,
ensures no double counting and inclusion of all entities.
# - Prints total number of cells without debris.
intact_cells_pos = cleaned_sample.gate(intact_gate_pos)
nmb_intact_pos = intact_cells_pos.data.shape[0]

intact_cells_neg = cleaned_sample.gate(intact_gate_neg)
nmb_intact_neg = intact_cells_neg.data.shape[0]

nmb_intact_tot = nmb_intact_neg + nmb_intact_pos

damaged_cells = cleaned_sample.gate(damaged_gate)
nmb_damaged = damaged_cells.data.shape[0]

damaged_percentage = nmb_damaged / nmb_cells * 100

avg_fl1_tot = sum(cleaned_sample['FITC-H']) / nmb_cells
avg_fl1_pos = sum(intact_cells_pos['FITC-H']) / nmb_intact_pos

print("\nNumber of entities total:")
print(nmb_ent)

nmb_ent_gate = nmb_intact_tot + nmb_damaged + nmb_debris
```

```
print("\nNumber of entities while gating:")
print(nmb_ent_gate)

nmb_cells_gate = nmb_intact_tot + nmb_damaged
print("\nNumber of cells after gating (no debris):")
print(nmb_cells_gate)

print("\nThe percentage of total cells that are PI positive:")
print(damaged_percentage)

print("\nAverage FL1-H florescence:")
print(avg_fl1_tot)
# print(avg_fl1_pos)

# Plots gated entities in different colors based on previous
gating using axes with logarithm scales.
# Also plots legends.
# The first graph plots the cells by GFP and PI, while the
second graph plots cells and debris based on
# size and granularity.
fig1 = plt.figure(1)
plt.scatter(cleaned_sample['FITC-H'], cleaned_sample['PerCP-
H'], s=0.8, alpha=0.5, label='All Cells')
plt.scatter(damaged_cells.data['FITC-H'],
damaged_cells.data['PerCP-H'], s=1.0, alpha=0.5, label='Damaged
Cells')
plt.scatter(intact_cells_pos.data['FITC-H'],
intact_cells_pos.data['PerCP-H'], s=1.0, alpha=0.5,
            label='Intact (GFP positive)')
plt.scatter(intact_cells_neg.data['FITC-H'],
intact_cells_neg.data['PerCP-H'], s=1.0, alpha=0.5,
            label='Intact (GFP negative)')
ax = matplotlib.pyplot.gca()
ax.set_xscale('log')
ax.set_yscale('log')
plt.xlabel('GFP (FL1-H)')
plt.ylabel('PI (FL3-H)')
plt.legend()

fig2 = plt.figure(2)
plt.scatter(sample['FSC-H'], sample['SSC-H'], s=0.8, alpha=0.5,
label='All Cells')
plt.scatter(damaged_cells.data['FSC-H'],
damaged_cells.data['SSC-H'], s=0.8, alpha=0.5, label='Damaged
Cells')
plt.scatter(intact_cells_pos.data['FSC-H'],
intact_cells_pos.data['SSC-H'], s=0.8, alpha=0.5,
            label='Intact (GFP Positive)')
plt.scatter(intact_cells_neg.data['FSC-H'],
intact_cells_neg.data['SSC-H'], s=0.8, alpha=0.5,
            label='Intact (GFP Negative)')
plt.scatter(debris.data['FSC-H'], debris.data['SSC-H'], s=0.8,
alpha=0.8, label='Debris')
ax = matplotlib.pyplot.gca()
ax.set_xscale('log')
ax.set_yscale('log')
```

```
plt.xlabel('FSC-H')
plt.ylabel('SSC-H')
plt.legend()

plt.show()
```

ii. *FCS Handling*

```
import FlowCytometryTools as fct
import matplotlib.pyplot as plt


def fcs_analysis(file_path):
    # Initializes the folder and file used, attributes the data to the 'sample' variable
    sample = fct.FCMeasurement(ID='Test 1', datafile=file_path)

    # OBS! The flourochromes listed are factory settings and are needed to call the data from their channels, they do
    # not necessarily correspond with the flourochromes used in the sample.
    # Data can only be called from the first 'name' (e.g. FITC-A not FL1-A for this setup)

    # Event,          Number of events in measurement
    # FSC-A,          Forward scatter area
    # SSC-A,        Side scatter area
    # FITC-A::FL1-A   Green flourochrome
    # PE-A::FL2-A,    Orange-Red flourochrome
    # PerCP-A::FL3-A, Red flourochrome
    # APC-A::FL4-A,   Far-red flourochrome
    # FSC-H, Forward scatter height

    # SSC-H, Side scatter height
    # FITC-H::FL1-H, Green intensity at its highest point
    # PE-H::FL2-H Orange-red intensity at its highest point
    # PerCP-H::FL3-H, Red intensity at its highest point
    # APC-H::FL4-H,Far-red intensity at its highest point
    # Width,          Pulse width
    # Time            Time for recording of each event

    # Prints channel names in the sample
    # print("Channel names:")
    # print(sample.channel_names)

    nmb_ent = sample.data.shape[0]

    # Initializes the gates based on graphical analysis. Values correspond to corners in a polygon and should be changed
    # to fit the particular dataset of interest. The cleanup gate is a threshold for size inclusion.
    cleanup_gate_x = fct.ThresholdGate(630000, 'FSC-H', region='above')
    cleaned_sample = sample.gate(cleanup_gate_x)
    nmb_cells = cleaned_sample.data.shape[0]
```

```
    debris_gate_x = fct.ThresholdGate(630000, 'FSC-H',
region='below')
    debris = sample.gate(debris_gate_x)
    nmb_debris = debris.data.shape[0]

    intact_gate_pos = fct.PolyGate([(1300, 50), (1300, 9000),
(400000, 170000), (350000, 50)],
('FITC-H', 'PerCP-H'), region='in', name='Intact Cells')
    intact_gate_neg = fct.PolyGate([(0, 50), (0, 9000), (1300,
9000), (1300, 50)], ('FITC-H', 'PerCP-H'), region='in',
name='Intact Cells')
    damaged_gate = fct.PolyGate([(0, 9000), (1000, 7e+6),
(260000, 8e+6), (400000, 170000), (1300, 9000)],   ('FITC-H',
'PerCP-H'), region='in', name='Damaged cells')

    # - Gates data from the sample and counts number of cells
contained in each gate.
    # - Calculates the percentage of PI stained cells in the
sample (FL3-A).
    # - Calculates the mean FL1-H fluorescence for the non-
stained cells and GFP positive cells.
    # - Prints total number of entities before and after
gating, ensures no double counting and inclusion of all
entities.
    # - Prints total number of cells without debris.
    intact_cells_pos = cleaned_sample.gate(intact_gate_pos)
    nmb_intact_pos = intact_cells_pos.data.shape[0]

    intact_cells_neg = cleaned_sample.gate(intact_gate_neg)
    nmb_intact_neg = intact_cells_neg.data.shape[0]

    nmb_intact_tot = nmb_intact_neg + nmb_intact_pos

    damaged_cells = cleaned_sample.gate(damaged_gate)
    nmb_damaged = damaged_cells.data.shape[0]

    damaged_percentage = nmb_damaged / nmb_cells * 100

    avg_fl1_tot = sum(cleaned_sample['FITC-H']) / nmb_cells
    avg_fl1_pos = sum(intact_cells_pos['FITC-H']) /
nmb_intact_pos

    print("\nNumber of entities total:")
    print(nmb_ent)

    nmb_ent_gate = nmb_intact_tot + nmb_damaged + nmb_debris
    print("\nNumber of entities while gating:")
    print(nmb_ent_gate)

    nmb_cells_gate = nmb_intact_tot + nmb_damaged
    print("\nNumber of cells after gating (no debris):")
    print(nmb_cells_gate)

    # print("\nThe percentage of total cells that är PI
positive:")
    # print(damaged_percentage)
```

```
    # print("\nAverage FL1-H florescence:")
    # print(avg_fl1_tot)
    # print(avg_fl1_pos)

    # Returns desired parameters, making them accessible for
the Main Executor.
    return damaged_percentage, avg_fl1_tot, nmb_ent  # ,
gfppos_percentage, meanfl1pos, cellconc
```

iii. Pump Handling

```
import serial
import time

port = "COM4"
baudrate = 115200
ser = serial.Serial(port, baudrate)
ser.setDTR(True)


# print(ser.name)  # Unhide this and run this script if you
want to find out the name of the USB port.

# This is used to control a specific pump from a specific
arduino. If another pump is to be used, the user has to find
# all its specifications and replace port, baudrate and
start/stop commands with the appropriate ones. it may also be
# necessary to encode/decode from a specific language (i.e.
ASCII). The user also has to change the "if response =="
# to the appropriate response phrase that is sent from the
pump/arduino. print(response) should let you know what the
# pump sends back, if you get b'x\00' the response is empty,
and something is wrong with the communication. Call the
# functions in an otherwise empty script to check
functionality.

def start_pump():
    command = "G0 S1\n"
    ser.write(command.encode())
    response = ser.readline()
    # print(response)
    if response == b'>\r\n':
        print("\nThe pump has been started.")
    else:
        print("\nThere was an issue with pump communication
(Incorrect response)")


def stop_pump():
    command = "G0 S0\n"
    ser.write(command.encode())
    response = ser.readline()
    # print(response)
    if response == b'>\r\n':
        print("\nThe pump has been stopped.")
```

```
    else:
        print("\nThere was an issue with pump communication
(Incorrect response)")


# Calibrated for pumping rpm 043, with 1 ml/h entering the
shakeflask (half-hour sample cycle).
def schedule_pump():
    for _ in range(1):
        start_pump()
        time.sleep(58)
        stop_pump()
        time.sleep(14 * 60)

    start_pump()
    time.sleep(58)
    stop_pump()


# Calibrated for pumping rpm 043, with 1.5 ml/h entering the
shakeflask (half-hour sample cycle).
def increase_pump():
    for _ in range(1):
        start_pump()
        time.sleep(1 * 60 + 27)
        stop_pump()
        time.sleep(13 * 60)

    start_pump()
    time.sleep(1 * 60 + 27)
    stop_pump()


# Calibrated for pumping rpm 043, with 3 ml/h entering the
shakeflask (half-hour sample cycle).
def fast_pump():
    for _ in range(1):
        start_pump()
        time.sleep(2 * 60 + 53)
        stop_pump()
        time.sleep(12)

    start_pump()
    time.sleep(2 * 60 + 53)
    stop_pump()


# Calibrated for pumping rpm 043, with 5 ml/h entering the
shakeflask (half-hour sample cycle).
def faster_pump():
    for _ in range(1):
        start_pump()
        time.sleep(4 * 60 + 49)
        stop_pump()
        time.sleep(10 * 60)
```

```
    start_pump()
    time.sleep(4 * 60 + 49)
    stop_pump()


# Calibrated for pumping rpm 043, with 10 ml/h entering the
shakeflask (half-hour sample cycle).
def constant_pump():
    for _ in range(1):
        start_pump()
        time.sleep(9 * 60 + 38)
        stop_pump()
        time.sleep(5 * 60)

    start_pump()
    time.sleep(9 * 60 + 38)
    stop_pump()
```

iv. *Pump Mock*

```
# This script is used for testing the functionality of the
other scripts while not having a pump connected.
# Essentially this is a mockup to be called in place of
Pump_Handling which will send phrases without starting
# anything else.
import time


def start_pump():
    print("The pump has been started (mock).")


def stop_pump():
    print("The pump has been stopped (mock).")


def schedule_pump():
    for _ in range(1):
        start_pump()
        time.sleep(1 * 60 + 17.5)
        stop_pump()
        time.sleep(2)

    start_pump()
    time.sleep(2)
    stop_pump()


def increase_pump():
    for _ in range(1):
        start_pump()
        time.sleep(1 * 60 + 3)
        stop_pump()
        time.sleep(4)

    start_pump()
```

```
        time.sleep(5)
        stop_pump()


    def fast_pump():
        for _ in range(1):
            start_pump()
            time.sleep(1 * 60 + 9)
            stop_pump()
            time.sleep(5)

        start_pump()
        time.sleep(2)
        stop_pump()


    def faster_pump():
        for _ in range(1):
            start_pump()
            time.sleep(1 * 60 + 28)
            time.sleep(4)

        start_pump()
        time.sleep(5)
        stop_pump()


    def constant_pump():
        for _ in range(1):
            start_pump()
            time.sleep(6)
            stop_pump()
            time.sleep(6)

        start_pump()
        time.sleep(6)
        stop_pump()
```

v. Main Executor

```
from Main import Pump_Handling as pmp
from Main import FCS_Handling as fch
# from Main import Pump_Mock as pm
import time
from datetime import datetime
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler
import matplotlib.pyplot as plt
from scipy.stats import linregress
import pandas as pd
import os
import warnings

# Some initial settings for data frame size and plot style
warnings.filterwarnings("ignore", category=UserWarning)
```

```
pd.set_option('display.max_columns', 10)
# backend = matplotlib.get_backend()
# print(backend)
# matplotlib.use('QtAgg')
plt.style.use('ggplot')

# Stops/starts the pump initially
# pmp.start_pump()
pmp.stop_pump()


class FileHandler(FileSystemEventHandler):
    # Initializes all the variables used in the FileHandler class
    def __init__(self):
        self.fig1, (self.ax1, self.ax2, self.ax3) =
plt.subplots(3, 1, sharex='col')
        self.file_counter = 0
        self.start_time = None
        self.analysis_results = []

        self.reactor_1_result = pd.DataFrame(columns=['PI
percentage', 'Mean FL1-H', 'Time(min)', 'Events'])
        self.new_data_1 = {}
        self.df_new_data_1 = None
        self.ema_reactor_1_PI = pd.DataFrame(columns=['PI
percentage (ema)'])
        self.ema_reactor_1_fl1 = pd.DataFrame(columns=['Mean FL1-
H (ema)'])
        self.dmg_percentage = []
        self.mean_fl1 = []
        self.events = []
        self.time_since = []
        #  self.time_since_seconds = None
        self.time_since_date = None

        self.fig1 = None
        self.ax1 = None
        self.ax2 = None
        self.ax3 = None

        self.reactor_1_last3_PI = []
        self.reactor_1_last3_fl1 = []
        self.reactor_1_last3_time = []

        self.pi_1_slope = 0
        self.fl1_1_slope = 0
        self.df_new_slopes = None
        self.new_slopes = None
        self.slopes_columns = pd.DataFrame(columns=['PI slope',
'FL1-H slope'])
        self.appended_result = None

        self.pi_last3 = None
        self.pi_last3_avg = None
        self.pump_status = None
        self.new_status = None
```

```
        self.df_new_status = None
        self.status_column = pd.DataFrame(columns=['Pump
Status'])
        self.full_result = None

    # - If first file, sets start time to current time.
    # - If first or every other file, set the file path to the
new file and call the FCS Handling script for
    #   extracting data, and subsequently saving, plotting,
finding the slopes and calling on the Pump Handling file
    #   to use the pump.
    # - Increases the file counter by one.
    def on_created(self, event):
        if self.file_counter == 0:
            self.start_time = datetime.now()

        if self.file_counter % 2 == 0:
            time.sleep(1)
            file_path = event.src_path
            self.analysis_results = fch.fcs_analysis(file_path)
            self.save_results()
            self.plot_results()
            self.get_slope()
            self.use_pump()

        self.file_counter += 1

    # - Saves the extracted results from the FCS file into Pandas
Dataframes which are easier to handle.
    # - Uses the file counter to calculate sample time
    # - Appends new results to the existing results dataframe in
their correct columns.
    # - Calculates EMAs for the new data (com = alpha = 0.9)
    def save_results(self):
        self.dmg_percentage = self.analysis_results[0]
        self.mean_fl1 = self.analysis_results[1]
        self.events = self.analysis_results[2]

        if self.file_counter >= 2:
            self.time_since = 15 * self.file_counter
        elif self.file_counter == 0:
            self.time_since = 0
        # self.time_since_date = datetime.now() - self.start_time
        # self.time_since_seconds =
self.time_since.total_seconds()
        # self.time_since = self.time_since_seconds/60

        self.new_data_1 = {'Time(min)': self.time_since, 'PI
percentage': self.dmg_percentage,
                           'Mean FL1-H': self.mean_fl1,
                           'Events': self.events}
        self.df_new_data_1 = pd.DataFrame([self.new_data_1])
        self.reactor_1_result = pd.concat([self.reactor_1_result,
self.df_new_data_1], ignore_index=True)
        self.ema_reactor_1_PI = self.reactor_1_result['PI
percentage'].ewm(com=0.9).mean()
```

```
        self.ema_reactor_1_fl1 = self.reactor_1_result['Mean FL1-
H'].ewm(com=0.9).mean()
        time.sleep(0.5)

    # - Plots MFI GFP, PI percentage, and events in a single
window as subplots, along with their legends.
    # - Plots the EMA for MFI GFP and PI percentage.
    def plot_results(self):
        if self.file_counter >= 1:
            plt.close(self.fig1)
        plt.ion()
        self.fig1, (self.ax1, self.ax2, self.ax3) =
plt.subplots(3, 1, sharex='col')
        self.fig1.set_figheight(8)
        self.fig1.set_figwidth(7)
        self.fig1.canvas.manager.window.geometry("+0+0")
        self.ax1.plot(self.reactor_1_result['Time(min)'],
self.reactor_1_result['PI percentage'], label='PI Data',
                      color='purple')
        self.ax1.plot(self.reactor_1_result['Time(min)'],
self.ema_reactor_1_PI, label='PI EMA', color='brown')
        self.ax1.set_ylabel('PI (%)')
        self.ax1.set_title('PI Positive Percentage')
        self.ax1.legend(loc='upper right')

        self.ax2.plot(self.reactor_1_result['Time(min)'],
self.reactor_1_result['Mean FL1-H'], label='FL1-H data',
                      color='pink')
        self.ax2.plot(self.reactor_1_result['Time(min)'],
self.ema_reactor_1_fl1, label='FL1-H EMA', color='blue')
        self.ax2.set_ylabel('Fluorescence')
        self.ax2.set_title('Mean FL1-H Fluorescence')
        self.ax2.legend(loc='lower right')

        self.ax3.plot(self.reactor_1_result['Time(min)'],
self.reactor_1_result['Events'], label='Events',
                      color='black')
        self.ax3.set_xlabel('Time (min)')
        self.ax3.set_ylabel('Events')
        self.ax3.set_title('Events')
        self.ax3.legend(loc='upper right')

        self.fig1.canvas.draw()
        self.fig1.canvas.flush_events()

    # Uses linear regression to get the slopes of the last three
datapoints if at least three sample files has been
    # run, appends new slopes to existing results dataframe and
prints the slopes.
    # Otherwise appends the text "Premature" to the results
dataframe.
    def get_slope(self):
        if self.file_counter >= 4:
            self.reactor_1_last3_PI = self.ema_reactor_1_PI[-3:]
            self.reactor_1_last3_fl1 = self.ema_reactor_1_fl1[-
3:]
```

```
            self.reactor_1_last3_time =
self.reactor_1_result['Time(min)'][-3:].values.tolist()

            pi_1_linreg = linregress(self.reactor_1_last3_time,
self.reactor_1_last3_PI)
            self.pi_1_slope = pi_1_linreg[0]
            fl1_1_linreg = linregress(self.reactor_1_last3_time,
self.reactor_1_last3_fl1)
            self.fl1_1_slope = fl1_1_linreg[0]

            self.new_slopes = {'PI slope': self.pi_1_slope, 'FL1-
H slope': self.fl1_1_slope}
            self.df_new_slopes = pd.DataFrame(self.new_slopes,
index=[0])
            self.slopes_columns = pd.concat([self.slopes_columns,
self.df_new_slopes],
                                            ignore_index=True)
            self.appended_result =
self.reactor_1_result.join(self.slopes_columns.set_index(self.rea
ctor_1_result.index)
                                                                    ,
rsuffix='_status')

            print("\nSlopes of the last three points:")
            print("PI:", self.pi_1_slope, "\nFL1-H:",
self.fl1_1_slope)
        else:
            self.pi_1_slope = "Premature"
            self.fl1_1_slope = "Premature"
            self.new_slopes = {'PI slope': self.pi_1_slope, 'FL1-
H slope': self.fl1_1_slope}
            self.df_new_slopes = pd.DataFrame(self.new_slopes,
index=[0])
            self.slopes_columns = pd.concat([self.slopes_columns,
self.df_new_slopes],
                                            ignore_index=True)
            self.appended_result = self.reactor_1_result.join(

self.slopes_columns.set_index(self.reactor_1_result.index)
                , rsuffix='_status')

    #  Uses previously calculated parameters to control the pump
if at least three sample files has been
    #  run, appends new slopes to existing results dataframe and
prints the slopes.
    #  Otherwise appends the text "Premature" to the results
dataframe.
    #  Prints the final dataframe.
    def use_pump(self):
        if self.file_counter >= 4:
            self.pi_last3 = self.reactor_1_result['PI
percentage'][-3:].values.tolist()
            self.pi_last3_avg = sum(self.pi_last3) /
len(self.pi_last3)

            # Stops the pump if the PI slope or the average value
```

```
of the last three exceedes certain values.
            # Appends "Stopped" to the Pump Status column.
            if self.pi_1_slope >= 0.05 or self.pi_last3_avg >=
40:
                pmp.stop_pump()
                self.pump_status = "Stopped"
                self.new_status = {'Pump Status':
self.pump_status}
                self.df_new_status =
pd.DataFrame([self.new_status])
                self.status_column =
pd.concat([self.status_column, self.df_new_status],
                                                ignore_index=True)
                self.full_result =
self.appended_result.join(self.status_column.set_index(self.appen
ded_result.index),

rsuffix='_status')
            # Maximum, 10ml/h.
            # Starts the pump at the maximum setting if the MFI
GFP slope is bellow a certain threshold and the last
            # pump status was the setting that precedes it, for
this setting "Maximum increased", or if the last
            # setting used was this one and the PI slope is not
increasing.
            # Appends "Maximum" to the Pump Status column.
            elif ((self.status_column.iloc[-1] == "Maximum
increased").item() and (self.fl1_1_slope <= -17.5) and
                    (self.pi_1_slope < 0.05)) or \
                      ((self.status_column.iloc[-1] ==
"Maximum").item() and (self.pi_1_slope < 0.05)):

                pmp.constant_pump()
                self.pump_status = "Maximum"
                self.new_status = {'Pump Status':
self.pump_status}
                self.df_new_status =
pd.DataFrame([self.new_status])
                self.status_column =
pd.concat([self.status_column, self.df_new_status],
                                                ignore_index=True)
                self.full_result =
self.appended_result.join(self.status_column.set_index(self.appen
ded_result.index),

rsuffix='_status')
            # Maximum increased, 5ml/h
            # Starts the pump at the maximum increased setting if
the MFI GFP slope is bellow a certain threshold
            # and the last pump status was the setting that
precedes it, for this setting "further increased",
            # or if the last setting used was this one and the PI
slope is not increasing.
            # Appends "Maximum increased" to the Pump Status
column.
            elif ((self.status_column.iloc[-1] == "Further
```

```
increased").item() and (self.fl1_1_slope <= -10) and
                  (self.pi_1_slope < 0.05)) or \
                    ((self.status_column.iloc[-1] == "Maximum
increased").item() and (self.pi_1_slope < 0.05)):

                pmp.faster_pump()
                self.pump_status = "Maximum increased"
                self.new_status = {'Pump Status':
self.pump_status}
                self.df_new_status =
pd.DataFrame([self.new_status])
                self.status_column =
pd.concat([self.status_column, self.df_new_status],
                                               ignore_index=True)
                self.full_result =
self.appended_result.join(self.status_column.set_index(self.appen
ded_result.index),

rsuffix='_status')
            # Further increased, 3ml/h
            # Starts the pump at the further increased setting if
the MFI GFP slope is bellow a certain threshold
            # and the last pump status was the setting that
precedes it, for this setting "Increased",
            # or if the last setting used was this one and the PI
slope is not increasing.
            # Appends "Further increased" to the Pump Status
column.
            elif ((self.status_column.iloc[-1] ==
"Increased").item() and (self.fl1_1_slope <= -10) and
                  (self.pi_1_slope < 0.05)) or \
                    ((self.status_column.iloc[-1] == "Further
increased").item() and (self.pi_1_slope < 0.05)):

                pmp.fast_pump()
                self.pump_status = "Further increased"
                self.new_status = {'Pump Status':
self.pump_status}
                self.df_new_status =
pd.DataFrame([self.new_status])
                self.status_column =
pd.concat([self.status_column, self.df_new_status],
                                               ignore_index=True)
                self.full_result =
self.appended_result.join(self.status_column.set_index(self.appen
ded_result.index),

rsuffix='_status')
            # Increased, 1.5ml/h
            # Starts the pump at the Increased setting if the MFI
GFP slope is bellow a certain threshold
            # and the last pump status was the setting that
precedes it, for this setting "Started",
            # or if the last setting used was this one and the PI
slope is not increasing.
            # Appends "Increased" to the Pump Status column.
```

```
            elif ((self.status_column.iloc[-1] ==
"Started").item() and (self.fl1_1_slope <= -10) and
                  (self.pi_1_slope < 0.05)) or \
                   ((self.status_column.iloc[-1] ==
"Increased").item() and (self.pi_1_slope < 0.05)):

                pmp.increase_pump()
                self.pump_status = "Increased"
                self.new_status = {'Pump Status':
self.pump_status}
                self.df_new_status =
pd.DataFrame([self.new_status])
                self.status_column =
pd.concat([self.status_column, self.df_new_status],
                                                ignore_index=True)
                self.full_result =
self.appended_result.join(self.status_column.set_index(self.appen
ded_result.index),

rsuffix='_status')
            # Start, 1ml/h
            # Starts the pump if the MFI GFP slope is bellow a
certain threshold,
            # or if the last setting used was this one and the PI
slope is not increasing.
            # Appends "Started" to the Pump Status column.
            elif (self.fl1_1_slope <= -0.8) or \
                   ((self.status_column.iloc[-1] ==
"Started").item() and (self.pi_1_slope < 0.05)):

                pmp.schedule_pump()
                self.pump_status = "Started"
                self.new_status = {'Pump Status':
self.pump_status}
                self.df_new_status =
pd.DataFrame([self.new_status])
                self.status_column =
pd.concat([self.status_column, self.df_new_status],
                                                ignore_index=True)
                self.full_result =
self.appended_result.join(self.status_column.set_index(self.appen
ded_result.index),

rsuffix='_status')
            # If no conditions apply (i.e turning/stationary
points)
            else:
                print("\nNo action taken")
                self.pump_status = "No Action"
                self.new_status = {'Pump Status':
self.pump_status}
                self.df_new_status =
pd.DataFrame([self.new_status])
                self.status_column =
pd.concat([self.status_column, self.df_new_status],
                                                ignore_index=True)
```

```
                self.full_result =
self.appended_result.join(self.status_column.set_index(self.appen
ded_result.index),

rsuffix='_status')

        # Before 3 data points, no slope available.
        else:
            print("\nNo action taken (premature)")
            self.pump_status = "Premature"
            self.new_status = {'Pump Status': self.pump_status}
            self.df_new_status = pd.DataFrame([self.new_status])
            self.status_column = pd.concat([self.status_column,
self.df_new_status],
                                           ignore_index=True)
            self.full_result =
self.appended_result.join(self.status_column.set_index(self.appen
ded_result.index),

rsuffix='_status')

        print("\n", self.full_result)

    # Can be called to save the current results dataframe and
figure. Will create a new folder named after the
    # start time and save the data to an excel file and graph as
a PNG.
    def save_to_folder(self):
        time_string = self.start_time.strftime("%Y-%m-%d_%H-%M-
%S")
        figure_name = f"Figure_{time_string}.png"
        data_name = f"Data_{time_string}.xlsx"
        save_to = r"C:\Users\ADMIN\Desktop\Saras
scripts\Data\Saved"

        folder_name = time_string
        new_path = os.path.join(save_to, folder_name)
        os.makedirs(new_path, exist_ok=False)
        plt.savefig(new_path + r'\ ' + figure_name)
        self.full_result.to_excel(new_path + r'\ ' + data_name)
        print("Data has been saved to the folder:" + time_string)

# Sets the folder path and sets up and starts the observer in the
designated folder.
folder_path = r"C:\Users\ADMIN\Desktop\Saras
scripts\Data\Goal\2023-08-17_11-30-22"
event_handler = FileHandler()
observer = Observer()
observer.schedule(event_handler, path=folder_path,
recursive=False)
observer.start()

try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
```

```
    observer.stop()
observer.join()
```