Taris Bioreactor User Manual

UCSC iGEM

2016

Contents

1 Introduction

The 2016 UCSC iGEM team proudly presents this manual with a purpose twofold: to guide those interested in assembling their own reactors for the advancement of synthetic biology at a do-it-yourself level, and to document the ten weeks of unquestionable effort that went into this project. In the following pages, we describe the bioreactor from the board up, including–but not limited to–assembling the peripheral circuit board using a predetermined list of components (found in the Appendix), installing the required software using `taris-controller`, our installable Python package, calibrating the bioreactor's sensors, and setting up the server software, `taris-server`, to view bioreactor conditions through an online interface.

2 Taris Controller

2.1 Bioreactor Hardware Setup

The Taris bioreactor can be segregated into several main components, each with their discrete sections. The hardware hierarchy is as follows:

1. **Physical Reactor**: houses culture and outputs desired metabolic byproducts
2. **Peripheral Circuit Board**: directs sensor signals and outputs to physical motors and heater
3. **Microcontroller**: governs feedback controls, safeguards, sensor data, and sending information to the online interface

2.1.1 Reactor Design

The physical reactor consists of several sections: a nutrient tank that contains growth media required to sustain the culture, the chemostatic chamber that houses the culture, the filter that removes impurities and biological mass, and a collection tank with which to store the desired metabolic byproducts. Figure 2.1.1 shows the overall physical setup of the Taris bioreactor.

It is important, first of all, to cover the processes that governs the Taris bioreactor. Namely, the concept of chemostasis, in which fresh growth media consisting of glucose, fructose, and LB, are pumped through a sterilizable vessel (in this case, a 1-gallon glass container). The rate at which media are provided to the reaction chamber is governed by the specific growth rate μ of the culture. This is determined through the doubling time t_d of the specific bacterial strain:

$$\mu_{max} = \frac{\ln(2)}{t_d}$$

The goal of chemostasis is to achieve a steady state in which the dilution rate D, or the rate at which the bioreactor culture is being diluted with fresh media,

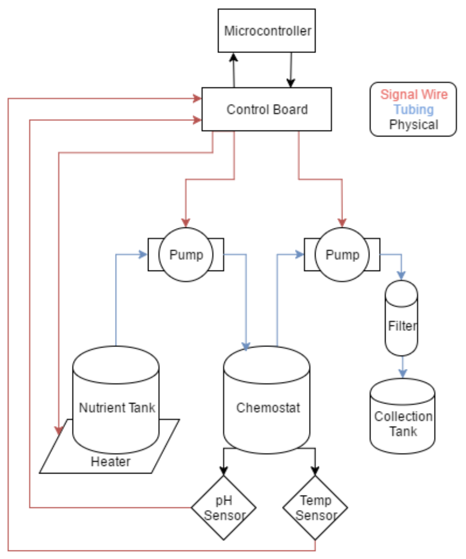


Figure 1: Reactor flow process diagram. The various connection types are indicated by the above key.

equals the rate at which the bacteria are growing. The dilution rate can be calculated from the medium flow rate F, which is defined by the rate which liquid is flowing into and out of the reaction vessel, and the culture volume V:

$$D = \frac{F}{V}$$

In the case of our bacterial strain, this $t_d \approx 100$ min. This means that

$$\mu_{max} = \frac{\ln(2)}{100\text{min}} \approx \boxed{0.007 \text{ min}^{-1}}$$

With a culture volume of 3.78L (1 US gallon), the medium flow rate can be determined:

$$F = DV = \mu V = 3.789\text{L} \cdot 0.007\text{min}^{-1} = \boxed{0.0256\text{L/min}}$$

This calculation may vary according to the specific conditions provided with the strain, so optimization is required to determine the ideal temperature, pH, and oxygen concentration–the latter of which is currently a rough estimate due to sensor cost–in order to provide the culture with conditions for maximum growth. The result of keeping the culture in a constant state of growth is the avoidance of bacterial stagnation due to intrinsic density limiters (quorum sensing).

The motors that drive the peristaltic pumps are 12V, 200mA DC. They are driven individually by nFETs controlled by the attached Raspberry Pi. The power delivered to the motors is controlled via a pulse width modulated (PWM)

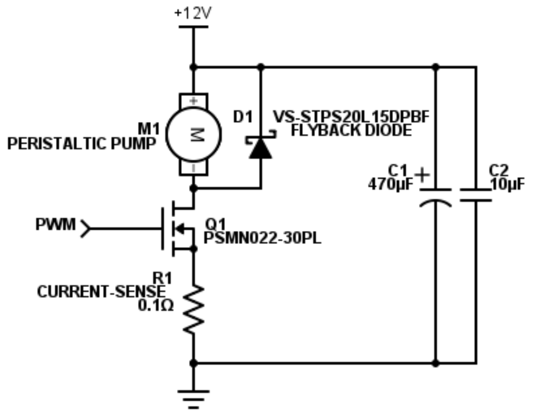


Figure 2: Motor driver for peristaltic pump control, complete with current-sense resistor and back EMF suppression.

signal, which is provided via the `PiBlaster` library (described in Section 2.3). Figure 2 shows a driver circuit for a single peristaltic pump.

Notice the additional capacitor bank in the figure above. This was necessary because of the inductive spikes due to the rotation of the DC motors that powered the peristaltic pumps. With each turn of a motor, back EMF (electromotive force) $\mathcal{E}$ is generated. This EMF appears as a voltage spike with a reverse polarity to the power supply, and can occur multiple times per rotation depending on the number of poles in the motor. Because the coils of wire constituting the motor windings have low resistance (the average off-state resistance of our peristaltic pumps was $\sim 1.2\Omega$), EMF also generates large current spikes–relative to steady state operation–to the ground plane. For instance, although the operating current of the peristaltic pump motors was normally 200mA, tests using 0.1Ω sense resistors yielded momentary spikes of up to 1.6A. By putting a bulk capacitor bank in parallel with each motor, noise due to EMF could be effectively attenuated. This bank was comprised of a 470μF, 16V aluminum polymer capacitor and a 10μF, 50V ceramic capacitor. This range was used to optimize noise suppression across a range of frequencies–covering the high-frequency spikes from EMF as well as the accompanying lower-frequency "plateaus" due to the motor charging with reverse polarity.

In addition to the capacitors, Vishay Semiconductors VS-STPS20L15DPBF[**?**] Schottky Rectifier diodes were used for flyback suppression. In the same line as EMF suppression, the diodes were used to form a conductive loop to starve the motor of residual charge. These diodes are placed with a reverse bias relative to the power supply so that they only conduct–becoming forward-biased–when the motor charges with reverse polarity to the power supply. Energy is then dissipated by the diode as heat, instead of being transferred to the rest of the circuit. These Schottky diodes were chosen because of their low forward voltage

(0.2V compared to 0.7V for standard diodes), their fast response (<500ns), and their high power tolerances–up to 1.5°C/W thermal resistance from junction to case and 125 °C T_{JMAX}, with transient spikes from the motors of 0.3W. These power ratings may be excessive, but they allow for scaling the motors without requiring PCB redesign.

2.1.2 Peripheral Circuit Board Design

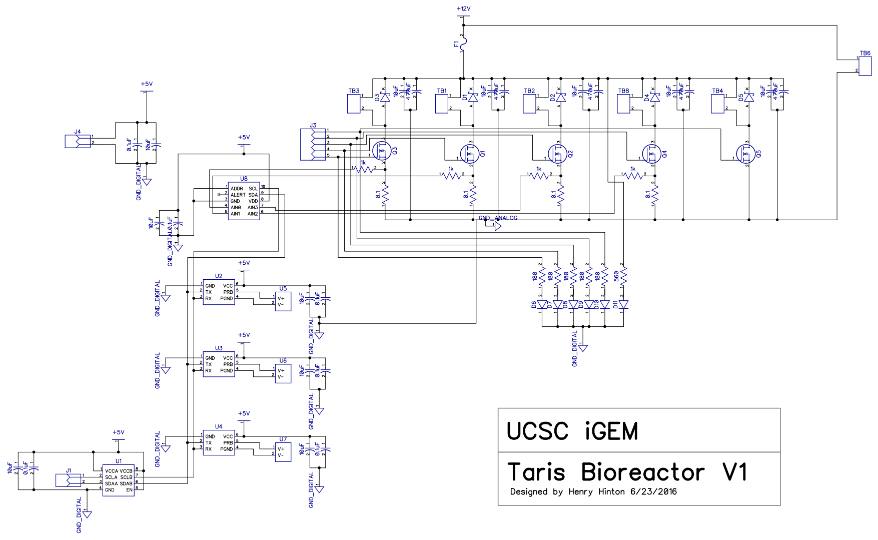
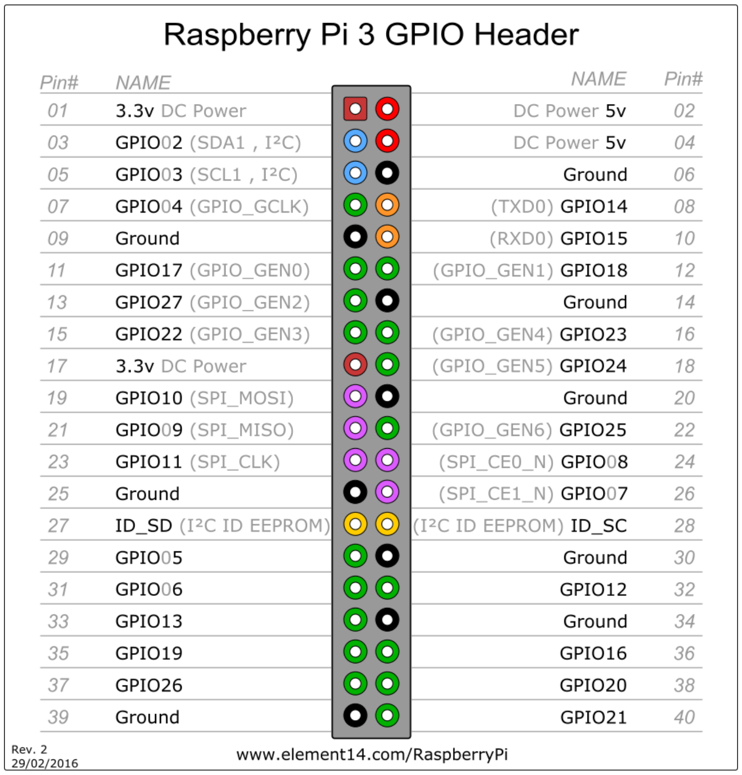


Figure 3: Schematic of the Taris circuit design. 12V and 0V connects to TB6. TB1, TB2, TB3, TB4, and TB8 connect to motors (or a relay to control a heating element).

The I2C address buses for the peripherals are defined as pH sensor: 0x63, temperature sensor: 0x66, and ADC (ADS1115): 0x48. The board is powered from two sources: 5V (typically the raspberry pi) powers the ADC, nMOSFETs, and LEDs; 12V powers the motors and any relays attached.

2.1.3 Microcontroller Setup

Power (5V) and ground pins need to be hooked up to the circuit from pins in Figure 3 (typically pins 1 and 39). Pins allocated for use in pi-blaster can be used for PWM control of motors or the heating element. Default pins for the motors are pins 21, 22, and 23. The default pin for the heating element is pin 17.



Raspberry Pi 3 GPIO Header

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I²C)	DC Power 5v	04
05	GPIO03 (SCL1 , I²C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)	(I²C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

Rev. 2
29/02/2016
www.element14.com/RaspberryPi

Figure 4: Pinout for the raspberry pi 3, courtesy of element14.com.

2.2 Software Installation (Raspberry Pi)

The software is currently run from taris_hw.py in its destination folder using python 2.7. The pi must be in I2C Mode. Matplotlib and Adafruit's ADS1X15 Library must also be installed prior.

2.3 Using the Bioreactor

2.3.1 Calibrating the Sensors

If the sensors have not yet been calibrated, this step is required, otherwise, it may be skipped, however, it is good practice to calibrate before the start of each new culture. In order to calibrate the sensors (both the Atlas EZO pH Sensor and Atlas EZO Temperature Sensor), simply follow the on-screen menu, and select "Calibrate Sensors". Both the temperature sensor and pH are recommended to be calibrated at the same time, as the effects of one can influence the bias of the other.

2.3.2 Running the Bioreactor

Once the sensors are calibrated, the bioreactor can be run with the option: "Run Bioreactor". Selecting this should immediately start displaying sensor data output from the system and begin warming the system to specified the parameter temperature. If the pH is lower than the parameter specified, it will add NaOH in slowly until the system reaches the desired pH. HCl addition was not implemented, because B. subtilis has been shown to only lower pH, and so we only expect to keep the pH from sinking. Because our pumps are not accurate, it only adds a drop or so at a time accounting for slow changes in pH, and it may take time to stabilize from an initial significant difference in pH. It is recommended to begin the culture with pH conditions as close as possible to the desired parameters.

Once running, the bioreactor will automatically begin sending JSON files of data to the server specified in the code (currently UCSC's SOE cloud server), and will check for new JSON files from the server telling it to change pH or temperature parameters.

2.4 Calibrating the PID

The control scheme we used for stabilizing the internal conditions of the bioreactor was PID, which stands for Proportional-Integral-Derivative. This is a common feedback mechanism that is widely applicable because of its simplicity and scalability. The PID control consists of three main transfer functions, each providing a particular gain effect on the system's output. Figure 5 shows a basic PID block diagram.

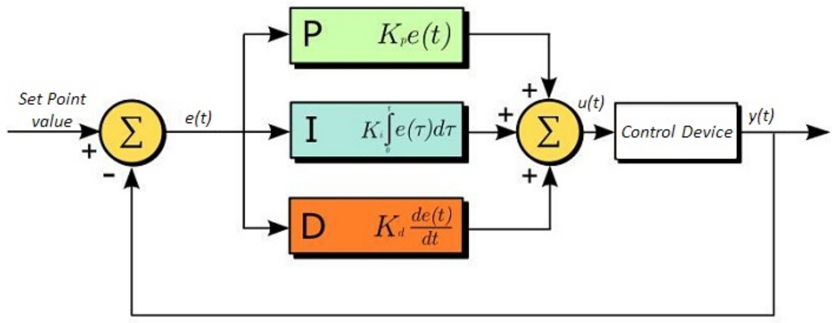


Figure 5: PID block diagram. In this case, the input to the system $x(t)$ is data from the EZO sensors, and the output $y(t)$ is the motor PWM frequency–the conversion from flow rate is done automatically.[?]

The first parameter in PID control is the Proportional signal, $P(t)$, where the error $e(t)$ is the difference between the last system output value and the system setpoint. In other words, $e(t) = x(t) - y_{sp}$. This proportional signal is completely based on the current error signal, scaled by some gain factor K_p:

$$P(t) = K_p e(t)$$

The second parameter is the Integral signal, which takes into account the speed at which an error has occurred. This means that the controller will limit the rate at which it responds to large error in order to decrease over-response to a brief oscillation that would otherwise keep the system average fairly constant. This means that, given the system error $e(t)$,

$$I(t) = K_i \int_0^t e(\tau) d\tau$$

where K_i is another scaling factor for the integral signal. The integral signal also has the opposite behavior for an error that has not been rectified for a long time. By integrating over the history of the error within a given time frame, the system's output continuously increases if the error is not rectified. It is evident that if $e(t) = 0$ (the system has reached its desired value), the integral signal will reset to 0 and will begin integrating again from that new time origin.

The addition of the integral signal $I(t)$, however, does not come without its consequences. As briefly noted, because any error that exists for a significant multiple of the sampling time t_s will quickly throw the output of the system to a very large value–and ruin the purpose of PID. This sharp rise in integral error then necessitates the additional Derivative signal, which analyzes the rate of change of the signal (and thus its future values) and accordingly dampens the behavior of the integral signal. This prevents the system output from oscillating due to high gain on the first two transfer functions, and allows it to narrow in on the desired final output value. As with the previous transfer functions, the derivative control also has a gain factor, K_d:

$$D(t) = K_d \frac{d}{dt} e(t)$$

Combining all of these transfer functions yields the full continuous form of the PID control:

$$y(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

Table 1 describes the effects of modifying the gain factors K_p, K_i, and K_d for a PID controller:

The basic tuning for a PID control is as follows:

- Set all gain coefficients to zero.

Parameter Increase	Rise Time	Overshoot	Settling Time	Steady-State Error
K_p	Decrease	Increase	$\pm\delta$	Decrease
K_i	Decrease	Increase	Increase	Greatly Reduce
K_d	$\pm\delta$	Decrease	Decrease	$\pm\delta$

Table 1: Effects of increasing parameter gain for each of the transfer function coefficients of a standard PID controller.[?]

- Increase K_p until system oscillates
- Increase K_d until system is critically damped
- Repeat last two steps until K_d does not damp oscillations, then increase K_i until system stabilizes to the desired oscillation time T_i.

The `taris-controller` package comes with an integrated PID control calibration tool to assist with determining the correct gain coefficients.

3 Taris Server

3.1 Server Setup

First, a server must be set up. For our purposes, we used a cloud-hosted server, first on Amazon's AWS, and then shifting over to one of our engineering department's SOE cloud servers. This was loaded with Ubuntu 14.04, and an Apache layer was installed on top of this. Anaconda (V4.2.0) was then installed. This provides for the dependencies that the Taris's python programs need (all are python version 2.7). Port access was limited to block traffic for all ports except port 22 SSH (Secure Shell), port 80 HTTP (HyperText Transfer Protocol), port 443 HTTPS (HyperText Transfer Protocol Secure), and port 22 (ssh) was restricted to only accept traffic from within UCSC.
The raspberry pi had to be setup to use eduroam through the school system in order to transfer data via this ssh. This involves editing the raspberry pi's wpa_supplicant.conf file.

First, navigate to the file and edit it with nano:

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

Edit in the following wifi network:

```
ctrl_interface=/var/run/wpa_supplicant
network={
    scan_ssid=1
    ssid="eduroam"
    key_mgmt=WPA-EAP
    eap=PEAP
    identity=" xxx@ucsc.edu"
    password="XXXXXX"
    ca_cert="/etc/ssl/certs/ucsc_noc_ca_der.crt"
    phase1="peaplabel=0"
    phase2="auth=MSCHAPV2"
}
```

After setting up the dependencies, the python, html, js, and css code is hosted on the server and run via TarisV1.py.

3.2 Resetting the Server

Just in case:

```
sudo -i #become root
ps aux | grep flask
sudo kill [task IDs to be killed]
cd ~directory #go into the proper directory
cd [path to TarisV1.py]
export FLASK_APP=TarisV1.py
nohup flask run --host=0.0.0.0 --port=80 &
```

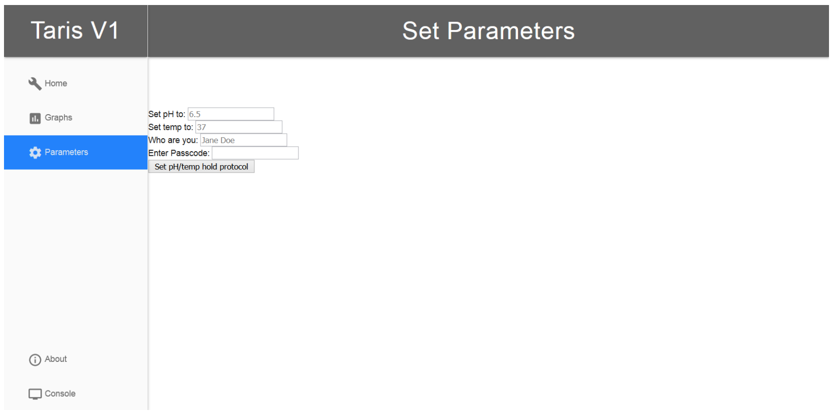
3.3 Interacting with the Website

The website is mainly a visual method of checking on the bioreactor, as well as allowing a look at past history to see if the heater ever boiled the culture briefly or overflowed or the pH went crazy and ticked up some morse code... or anything really. The real functionality is in the parameters page (accessed from the sidebar) which takes the following inputs:

- Set pH to:
- Set temp to:
- Who are you:
- Enter Passcode:

Each parameter must be entered and a username and password given to authenticate the changes to the bioreactor's functions.

3.4 Website Screenshots



Set pH to: 6.5
Set temp to: 37
Who are you: Jane Doe
Enter Passcode:
Set pH/temp hold protocol

Figure 6: Parameters webpage allowing access and control of the bioreactor remotely (smartphone or computer).

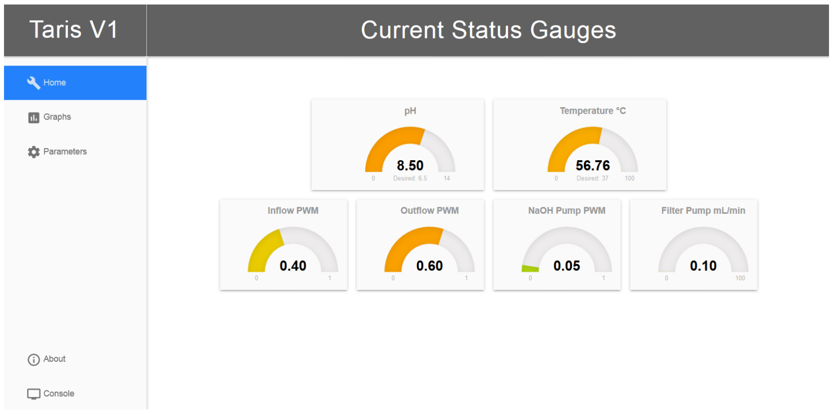


Figure 7: Main webpage with gauges indicating current bioreactor conditions.

3.5 Sample Data

Here is some data collected from *Bacillus subtilis* (strain 3NA) over several hours.

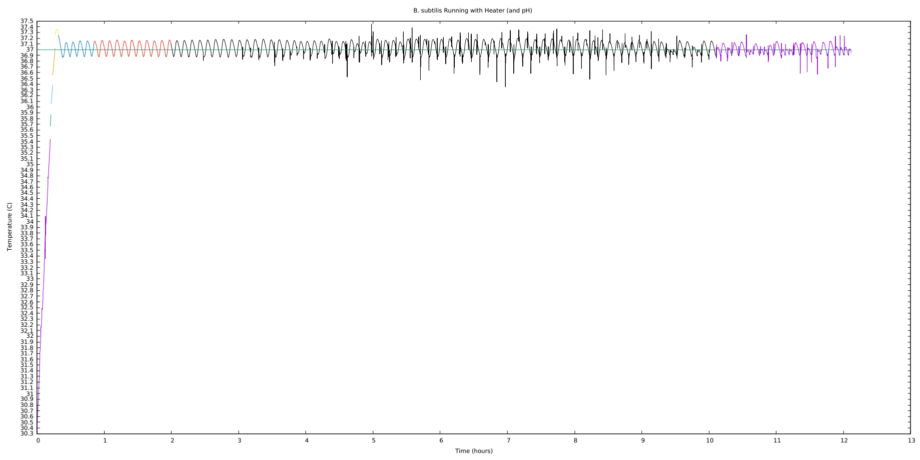


Figure 8: Temperature measurements recorded using the Atlas Scientific RTD sensor. Slight oscillations are normal due to the nature of PID, and are on an inconsequential scale.

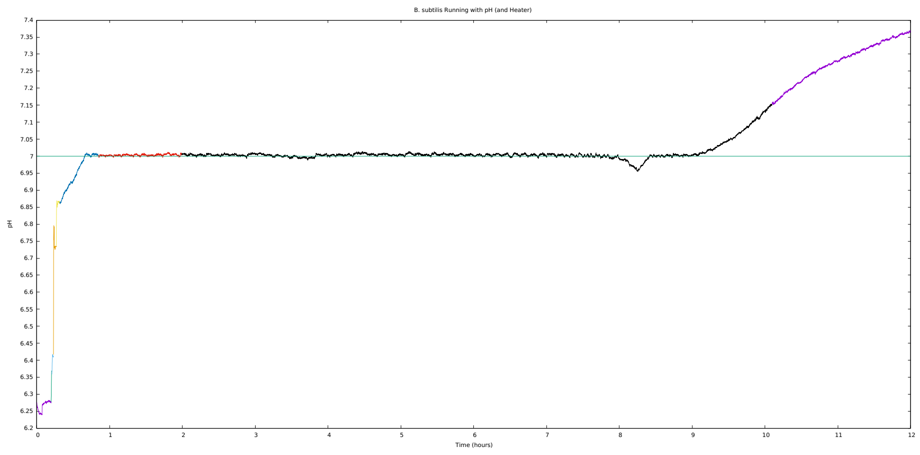


Figure 9: pH recording using the Atlas Scientific EZO sensor indicates cultural contamination or a mishap with the pH control circuit. It is unclear which occured, but it is nonetheless important to recognize the general stability of the system.

4 Appendix

4.1 Parts List

Table 2: PCB Components

Component Model	Quantity	Component Description
NXP PSMN022-30PL,127	5	nFET for motor driver
STPS20L15DPBF-ND	5	Flyback diode
296-38849-1-ND	1	ADS 1115 ADC i2C
P16299-ND	5	Alu-Poly Cap 470uF
43FR10E-ND	4	3W 0.1Ω sense resistor
CRT0603-BY-1001EASCT-ND	8	1K SMD resistor
WM5514-ND	2	BNC Connector
445-2887-ND	10	10uF Ceramic Cap
478-5741-ND	5	0.1uF Ceramic Cap
ED1609-ND	6	2 Screw Terminal
WK6245-ND	1	Fuse Holder 5x20mm
283-2836-ND	1	Fuse 15A
296-35972-1-ND	1	TCA9517 I2C Buffer
RMCF0805JT180RCT-ND	5	180Ω CL resistor
311-560ERCT-ND	1	560Ω CL resistor
SML-D12U1WT86CT-ND	1	Red LED
SML-D12Y1WT86CT-ND	5	Yellow LED
S7000-ND	2	2 Pin female Header
S6103-ND	1	5 Pin female Header
Atlas Scientific EZO pH and RTD	1	Sensors for bioreactor

Table 3: Reactor Parts

Component Description	Quantity
1 Gallon Fermentation Vessel	3
Peristaltic Pump 100mL/min	4
150W Heater Pad (60W min)	1
Air Pump + Air Stone (8L/min)	1
Check Valve	8
Inline Air Filter (ideally HEPA)	4
12V, 10A power supply	1
3/16” barb to 1/4” NPT Male	6
3/16” barb to 1/4” NPT Female	6
3/16” OD silicone tubing	15’
Mounting Brackets for Vessels + Pumps	6
Rubber stopper for holding probe leads	2

4.2 Schematics

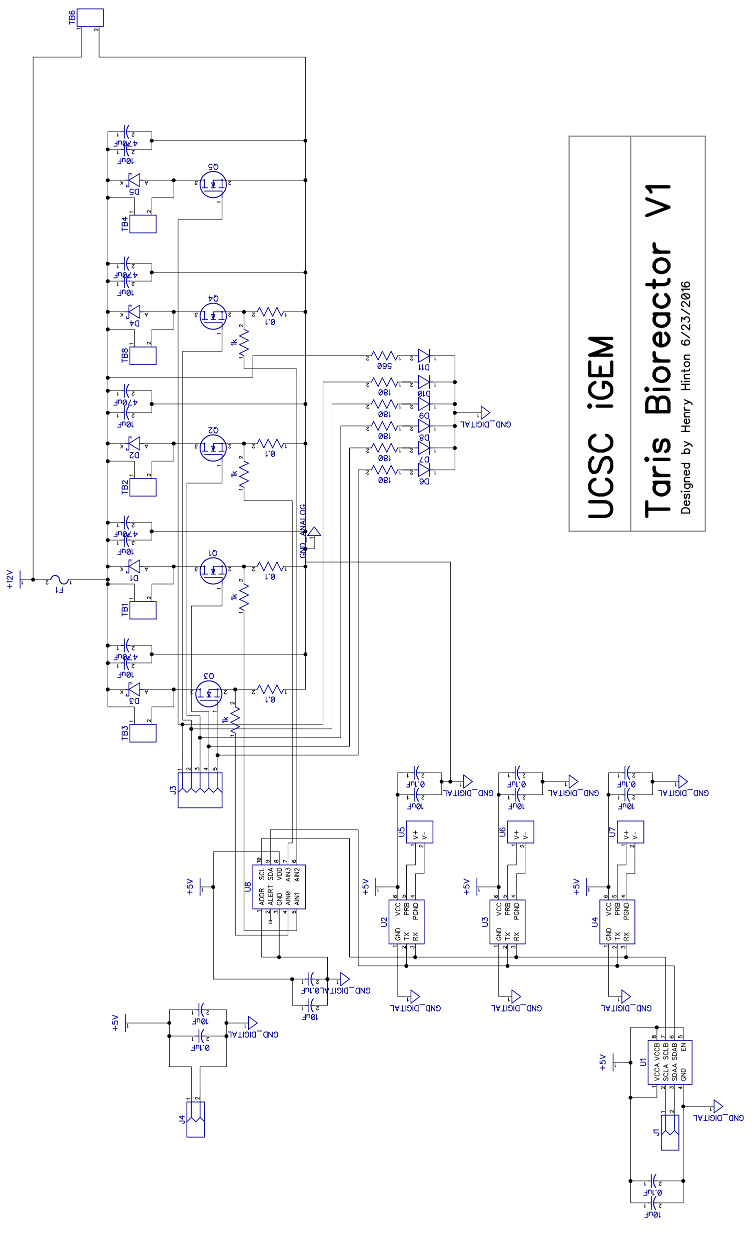


Figure 10: Schematic for bioreactor control board