

Principles of Bioinstrument Design: Syringe Pump/Microscope

Sina Booeshaghi

Jase Gehring

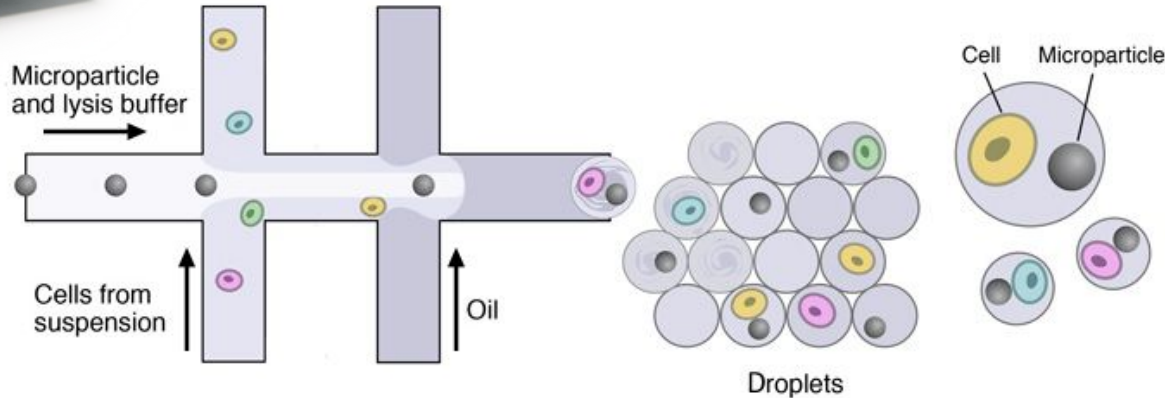
(We work under Professor Lior Pachter in BBE/CMS)

Single-cell RNA Seq profiles thousands of cells

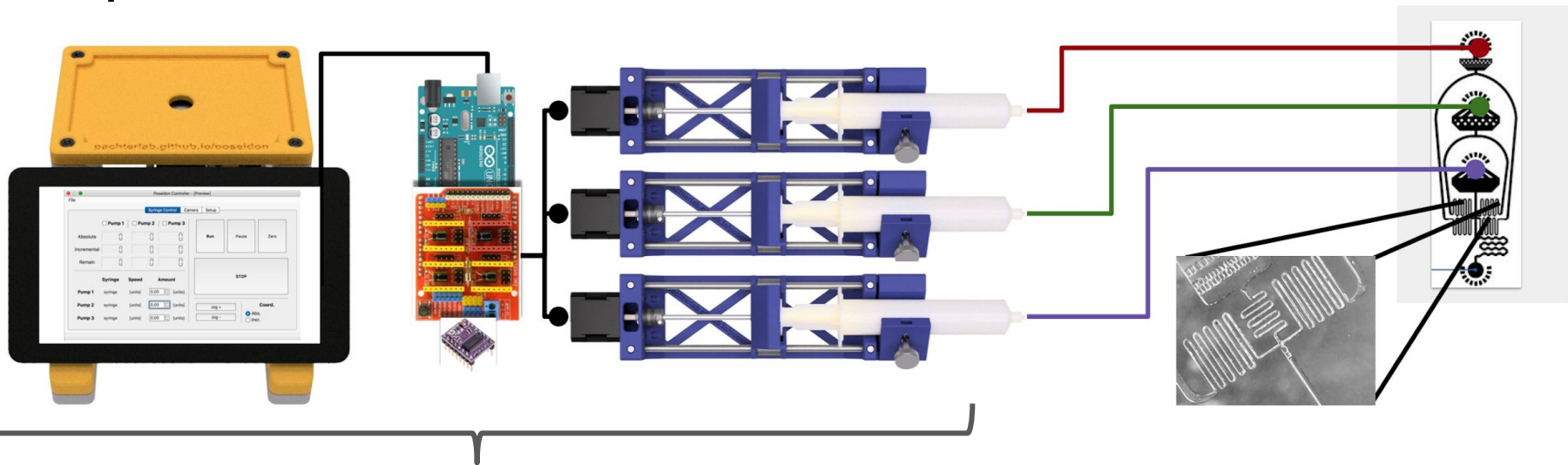


← Cost for a single 10x
Genomics Machine \$75k
Cost per run: \$1500

Cost for a DropSeq rig
and Pumps \$10K
Cost per run: \$600



The problem: syringe pump/microscopes expensive, not hackable



Poseidon Cost: ~\$310

Harvard Pumps Cost: ~\$5-10K



We followed 6 Principles of Bioinstrumentation to tackle the problem

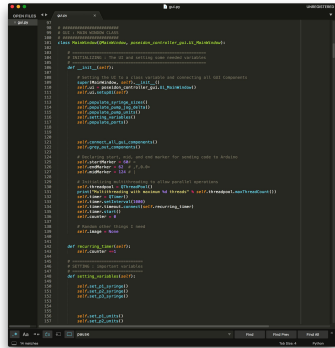
1. Functionality (Follow functional requirements)
2. Simplicity (Avoid complicated solutions)
3. Modularity (Use standard components)
4. Robustness (The “idiot user” approach)
5. Benchmarking (Test and retest and retest)
6. Documentation (Videos, pictures, text)

Recommendation: print out a list like this and post it to your wall. It can help serve as a template for making design decisions.

Functionality: Always start with a set of functional requirements

Specification	Description	Associated Value
Pump Size	Can be printed in one shot	Build Volume 8 x 8 x 10 in
Syringe Sizes	Adaptable to BD syringe	[1, 3, 5, 10, 20, 30, 60] mL
Desired Flow Rate	from DropSeq Protocol*	1,000-15,000 μ L/hr
Stepper Motor Driven	Run off of Arduino 12VDC	200 steps/rev w/ 32 μ step
Microscope	Magnification	Image microfluidic device
Cost	Total cost of the system + parts	<\$500

With these requirements in mind, we selected the appropriate design tools



```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
    3D printing process control script
    """
import sys
import os
import subprocess
import time
import argparse
import logging
import signal
import threading
import queue
import random
import math
import re
import json
import glob
import shutil
import datetime
import platform
import psutil
import subprocess

# Configuration
CONFIG_PATH = os.path.join(os.path.dirname(__file__), 'config.json')
LOG_PATH = os.path.join(os.path.dirname(__file__), 'logs')
PRINT_PATH = os.path.join(os.path.dirname(__file__), 'prints')
GCODE_PATH = os.path.join(os.path.dirname(__file__), 'gcode')
TEMP_PATH = os.path.join(os.path.dirname(__file__), 'temp')

# Logging
logging.basicConfig(level=logging.INFO, filename=os.path.join(LOG_PATH, 'process.log'))
logger = logging.getLogger(__name__)

# Arguments
parser = argparse.ArgumentParser()
parser.add_argument('--config', type=str, default=CONFIG_PATH, help='Path to configuration file')
parser.add_argument('--log', type=str, default=LOG_PATH, help='Path to log file')
parser.add_argument('--print', type=str, default=PRINT_PATH, help='Path to print files')
parser.add_argument('--gcode', type=str, default=GCODE_PATH, help='Path to gcode files')
parser.add_argument('--temp', type=str, default=TEMP_PATH, help='Path to temp files')
parser.add_argument('--verbose', action='store_true', help='Enable verbose logging')
args = parser.parse_args()

# Global variables
print_name = None
print_path = None
gcode_path = None
temp_path = None
verbose = False

def signal_handler(sig, frame):
    """Signal handler to gracefully exit the program"""
    logger.info("Received signal %s, exiting gracefully.", sig)
    sys.exit(0)

signal.signal(signal.SIGINT, signal_handler)

def load_config(config_path):
    """Load configuration from JSON file"""
    config = {}
    if os.path.exists(config_path):
        with open(config_path, 'r') as f:
            config = json.load(f)
    return config

def create_log(log_path):
    """Create log directory if it doesn't exist"""
    os.makedirs(log_path, exist_ok=True)

def create_print_dir(print_path):
    """Create print directory if it doesn't exist"""
    os.makedirs(print_path, exist_ok=True)

def create_gcode_dir(gcode_path):
    """Create gcode directory if it doesn't exist"""
    os.makedirs(gcode_path, exist_ok=True)

def create_temp_dir(temp_path):
    """Create temp directory if it doesn't exist"""
    os.makedirs(temp_path, exist_ok=True)

def get_print_files():
    """Get list of print files"""
    files = []
    for file in os.listdir(print_path):
        if file.endswith('.gcode'):
            files.append(os.path.join(print_path, file))
    return files

def get_gcode_files():
    """Get list of gcode files"""
    files = []
    for file in os.listdir(gcode_path):
        if file.endswith('.gcode'):
            files.append(os.path.join(gcode_path, file))
    return files

def get_temp_files():
    """Get list of temp files"""
    files = []
    for file in os.listdir(temp_path):
        if file.endswith('.gcode'):
            files.append(os.path.join(temp_path, file))
    return files

def print_file(print_path, gcode_path, temp_path):
    """Print a file"""
    print_name = os.path.splitext(os.path.basename(print_path))[0]
    gcode_file = os.path.join(gcode_path, print_name + '.gcode')
    temp_file = os.path.join(temp_path, print_name + '.gcode')

    # Copy gcode to temp
    shutil.copy(gcode_file, temp_file)

    # Print
    subprocess.run(['python3', 'printer.py', temp_file])

    # Move back to gcode
    shutil.move(temp_file, gcode_file)

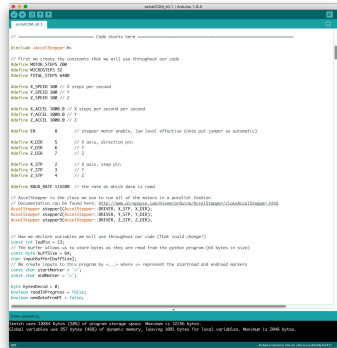
def main():
    """Main function"""
    config = load_config(CONFIG_PATH)
    logger.info("Configuration loaded: %s", config)

    create_log(LOG_PATH)
    create_print_dir(PRINT_PATH)
    create_gcode_dir(GCODE_PATH)
    create_temp_dir(TEMP_PATH)

    files = get_print_files()
    if not files:
        logger.error("No print files found in %s", PRINT_PATH)
        sys.exit(1)

    for file in files:
        print_file(file, GCODE_PATH, TEMP_PATH)

if __name__ == '__main__':
    main()
```



```
#!/bin/bash
# 3D printing process control script
set -e

# Configuration
CONFIG_PATH="config.json"
LOG_PATH="logs"
PRINT_PATH="prints"
GCODE_PATH="gcode"
TEMP_PATH="temp"

# Logging
LOG_FILE="${LOG_PATH}/process.log"
touch "$LOG_FILE"
logger="logger -t 3D-Printing -s"

# Arguments
VERBOSE=""
while getopts "v" opt; do
    case $opt in
        v) VERBOSE="--verbose";;
    esac
done

# Global variables
PRINT_NAME=""
PRINT_PATH=""
GCODE_PATH=""
TEMP_PATH=""

# Functions
load_config() {
    local config_path=$1
    local config=""
    if [ -f "$config_path" ]; then
        config=$(cat "$config_path")
    fi
}

create_log() {
    local log_path=$1
    mkdir -p "$log_path"
}

create_print_dir() {
    local print_path=$1
    mkdir -p "$print_path"
}

create_gcode_dir() {
    local gcode_path=$1
    mkdir -p "$gcode_path"
}

create_temp_dir() {
    local temp_path=$1
    mkdir -p "$temp_path"
}

get_print_files() {
    local print_path=$1
    local files=""
    for file in $(ls "$print_path"); do
        if [ "${file##*.}" = "gcode" ]; then
            files="$files $print_path/$file"
        fi
    done
}

get_gcode_files() {
    local gcode_path=$1
    local files=""
    for file in $(ls "$gcode_path"); do
        if [ "${file##*.}" = "gcode" ]; then
            files="$files $gcode_path/$file"
        fi
    done
}

get_temp_files() {
    local temp_path=$1
    local files=""
    for file in $(ls "$temp_path"); do
        if [ "${file##*.}" = "gcode" ]; then
            files="$files $temp_path/$file"
        fi
    done
}

print_file() {
    local print_path=$1
    local gcode_path=$2
    local temp_path=$3

    PRINT_NAME=$(basename "$print_path" .gcode)
    GCODE_FILE="$gcode_path/$PRINT_NAME.gcode"
    TEMP_FILE="$temp_path/$PRINT_NAME.gcode"

    cp "$GCODE_FILE" "$TEMP_FILE"

    python3 printer.py "$TEMP_FILE"

    mv "$TEMP_FILE" "$GCODE_FILE"
}

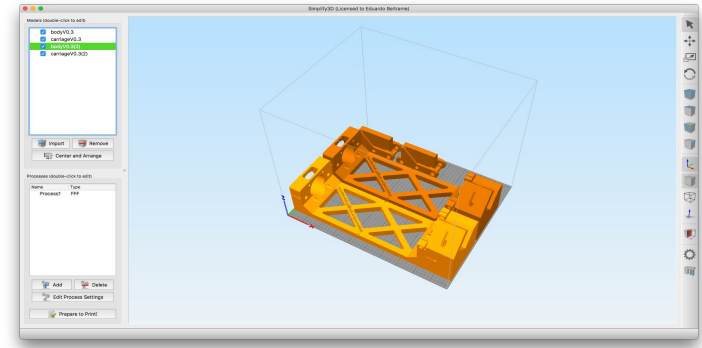
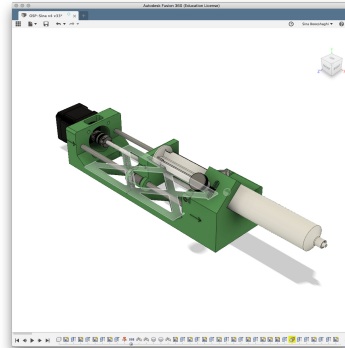
main() {
    load_config "$CONFIG_PATH"
    logger "$LOG_FILE" "Configuration loaded: $CONFIG_PATH"

    create_log "$LOG_PATH"
    create_print_dir "$PRINT_PATH"
    create_gcode_dir "$GCODE_PATH"
    create_temp_dir "$TEMP_PATH"

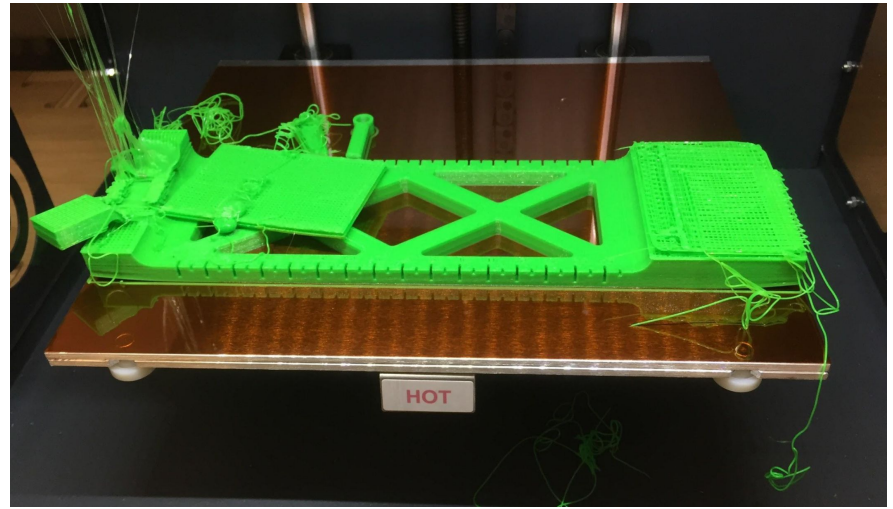
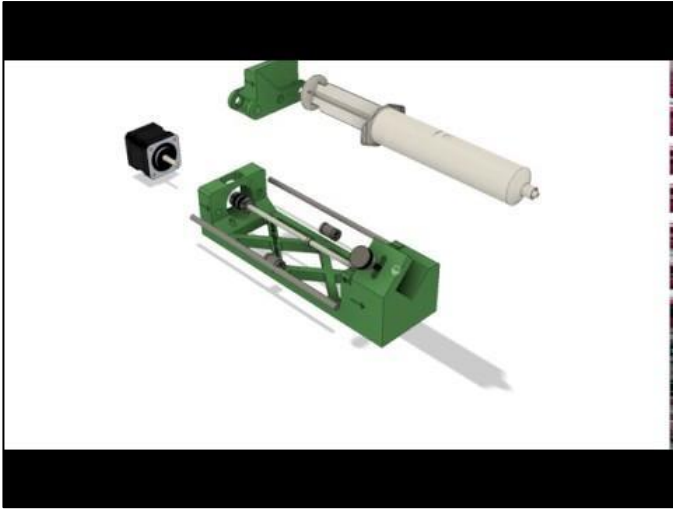
    files=$(get_print_files "$PRINT_PATH")
    if [ -z "$files" ]; then
        logger "$LOG_FILE" "No print files found in $PRINT_PATH"
        exit 1
    fi

    for file in $files; do
        print_file "$file" "$GCODE_PATH" "$TEMP_PATH"
    done
}

main
```



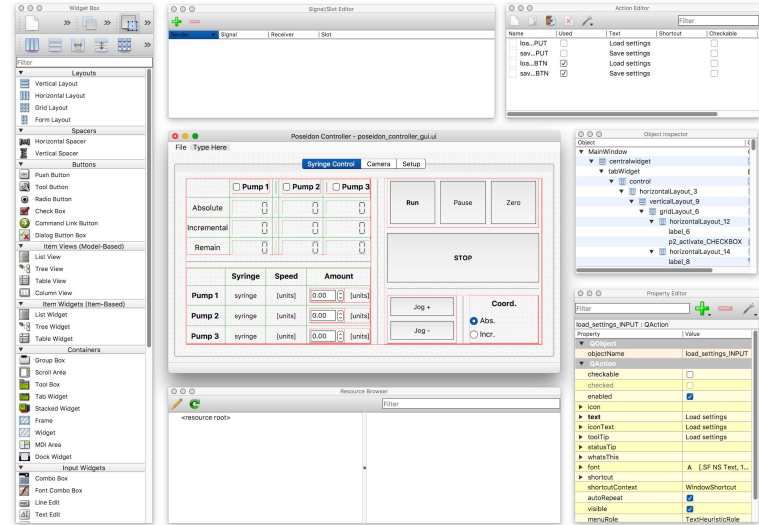
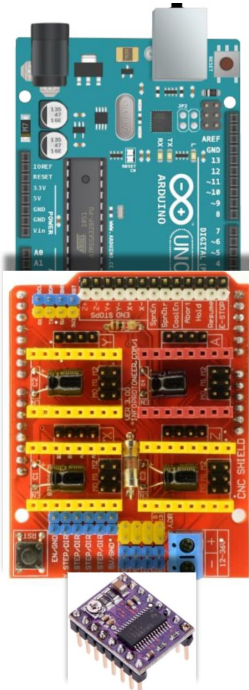
Then we began designing and iterating..



Simplicity: We wanted the system to be simple enough to use but flexible enough to hack

Hardware:

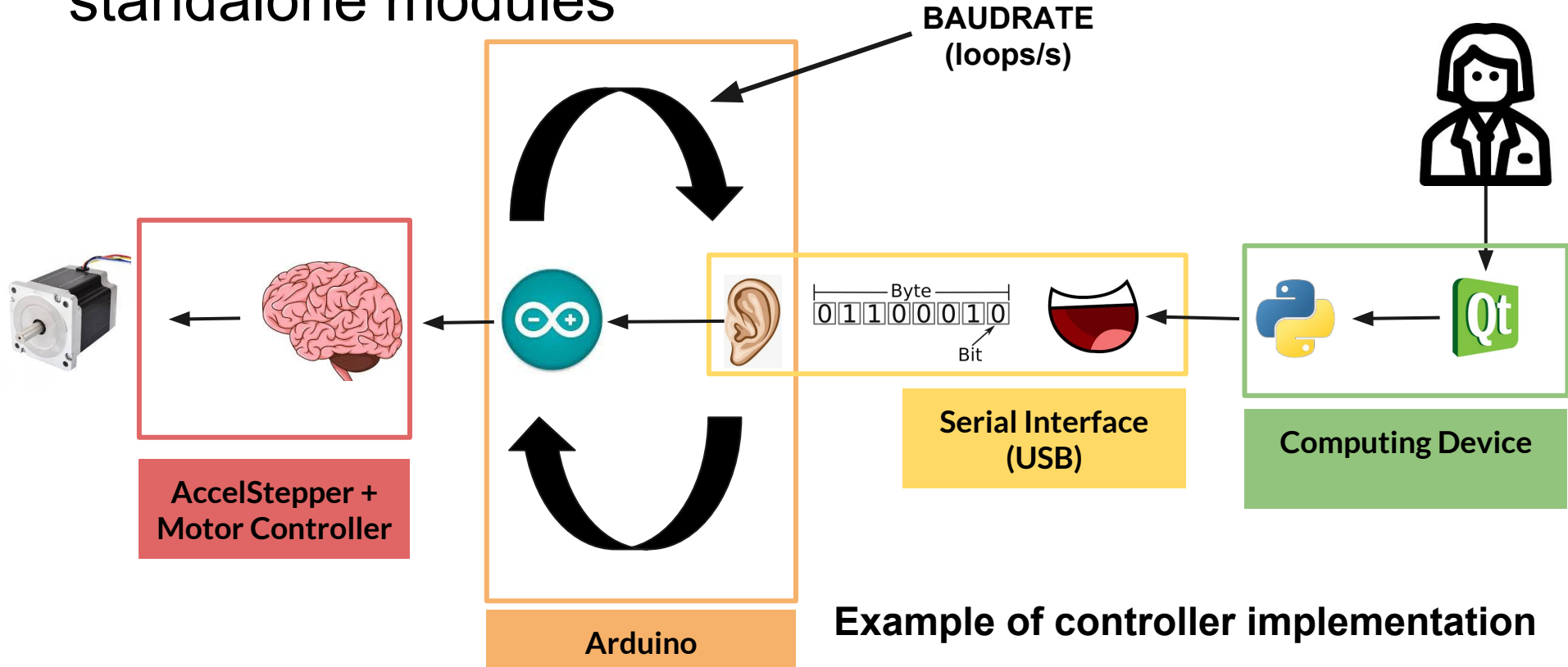
Plug and play parts, no soldering required



Software:

Drag and drop GUI development, controls written in python

Modularity: System can be broken down into standalone modules



Modularity: We used standard components

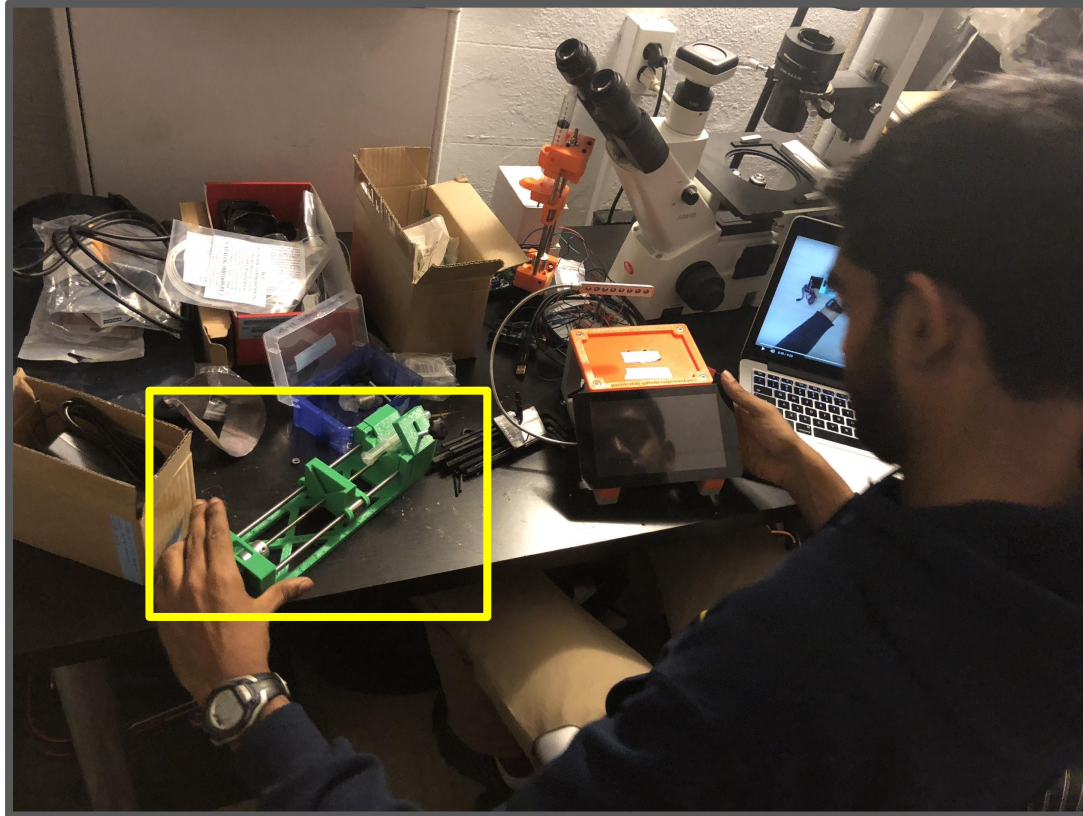
Syringe Pump Array		Cost per pump	31.445							
Total for 3 pumps		\$141.92								
Item Number	Item Description	Items per package	Items per Pump	Items per 3 pumps	Packages per 3 pumps	Cost per Package	Price per item	Cost per 3 pumps	Cost per pump	Supplier
1	Nema 17 Stepper Motor (Bipolar, 40mm, 59Ncm)	3	1	3	1	31.99	10.66333333	31.99	10.66333333	Amazon
2	5mm to 5mm Motor Shaft Coupling	10	1	3	1	22	2.2	22	2.2	Amazon
3	6mm Steel Rod (length 200mm, pack of 2)	2	2	6	3	6.24	3.12	18.72	6.24	Amazon
4	6mm Linear Bearing (pack of 12)	12	2	6	1	10.66	0.8883333333	10.66	1.776666667	Amazon
5	M5x0.8 Threaded Rod (length 170mm)	5	1	3	1	7.98	1.596	7.98	1.596	Amazon
6	M5x0.8 nut	50	2	6	1	6.23	0.1246	6.23	0.2492	Amazon
7	M3x0.5 socket head screws to mount motor (length 20	100	4	12	1	8.47	0.0847	8.47	0.3388	Amazon
8	M5 knob (hold syringe in place)	10	1	3	1	11.9	1.19	11.9	1.19	Amazon
9	12V power unit (end stripped to fit CNC shield power ir	1	0.3	1	1	9.89	9.89	9.89	2.967	Amazon
10	Arduino + CNC Shield Pack + DRV8825 (4)	1	0.3	1	1	14.08	14.08	14.08	4.224	Amazon
Microscope		Per Microscope	160.2768							
Microscope Total		\$169.01								
Item Number	Item Description	Items per package	Items per microscope	Packages per microscope	Cost per package	Price per item	Miscroscope cost			Supplier
1	Raspberry Pi Motherboard	1	1	1	34.99	34.99	34.99			Amazon
2	Raspberry Pi 7" touchscreen display	1	1	1	68.7	68.7	68.7			Amazon
3	Raspberry Pi Power Suppy (5v 1.5A DC)	1	1	1	9.99	9.99	9.99			Amazon
4	16gb MicroSD card (comes with adapter)	1	1	1	7.17	7.17	7.17			Amazon
5	Keyboard + Mouse Bundle (wired)	1	1	1	14.44	14.44	14.44			Amazon
6	M5x0.8 Socket Head Screw (length 14mm)	15	8	1	7.5	0.5	4			Amazon
7	M5x0.8 nuts	50	8	1	6.23	0.1246	0.9968			Amazon
8	USB Camera	1	1	1	19.99	19.99	19.99			Amazon
Project Total		\$310.93								

Robustness: Employing the idiot user approach



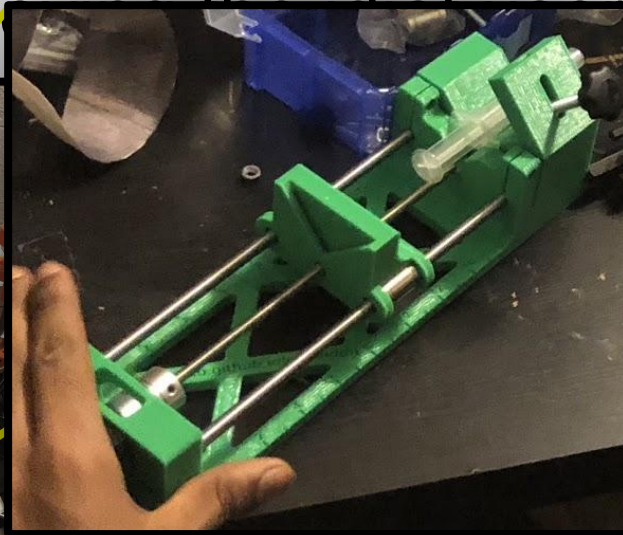
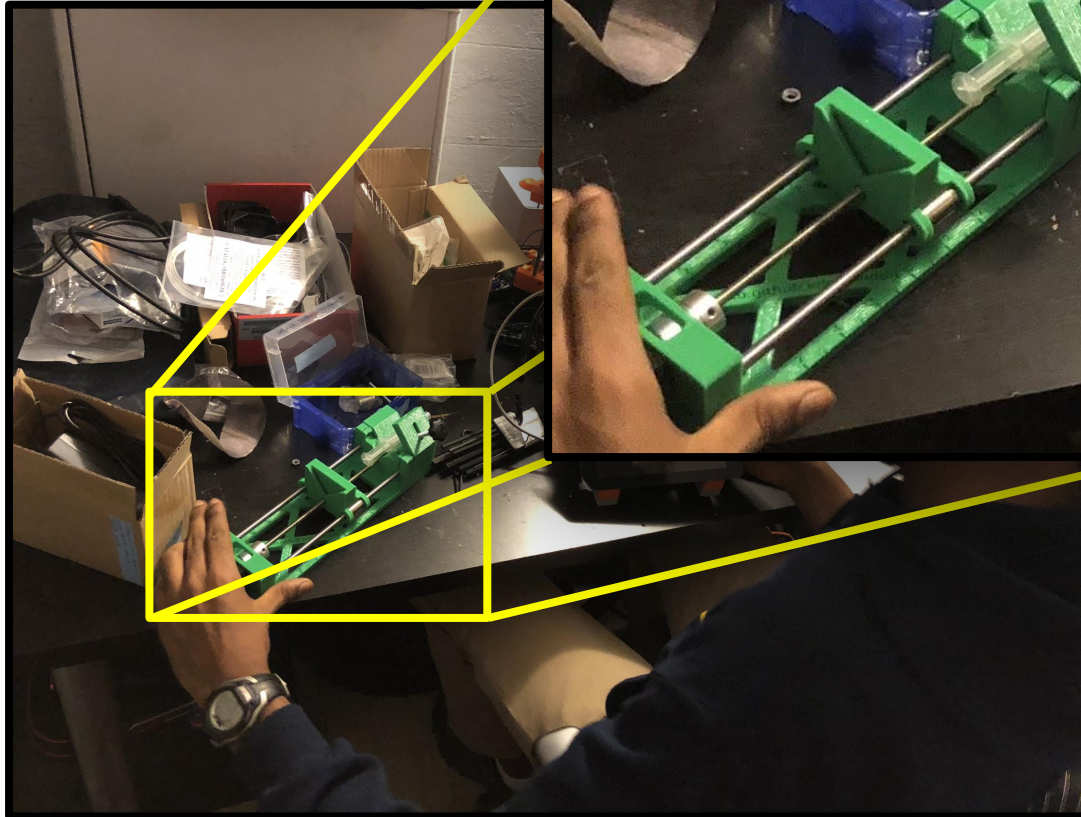
Notice something wrong with the device?

Robustness: Employing the idiot user approach



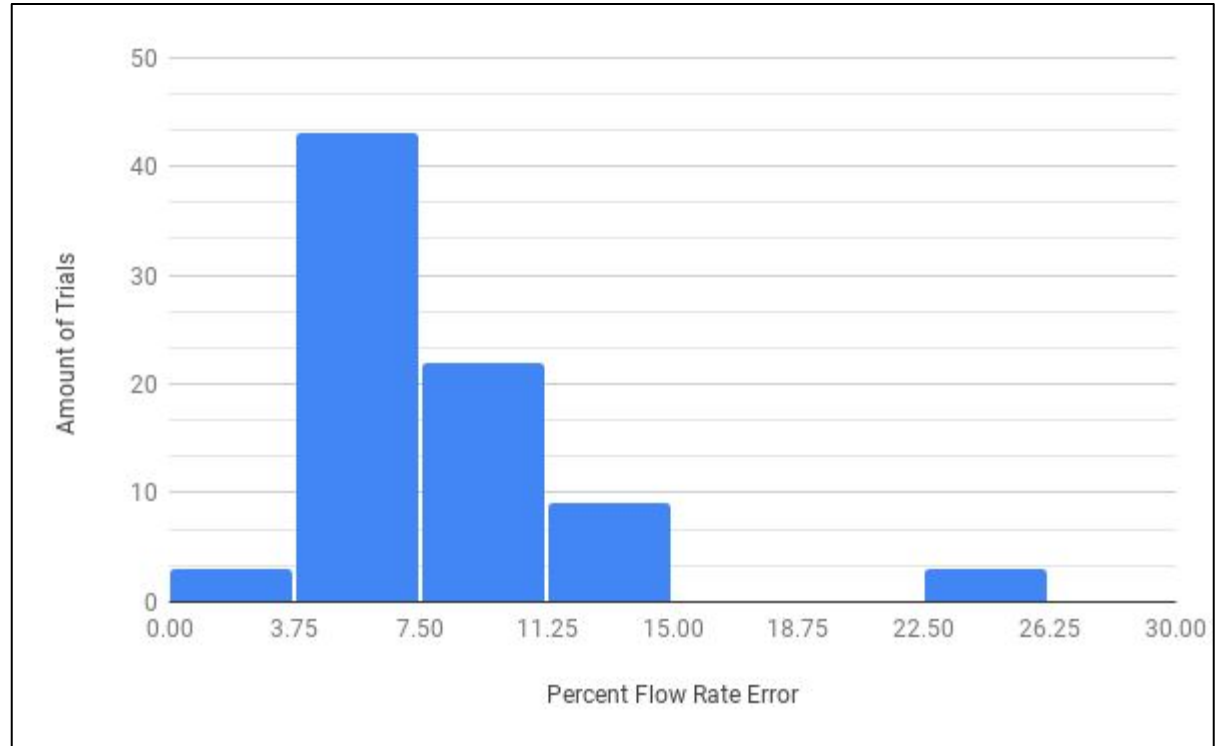
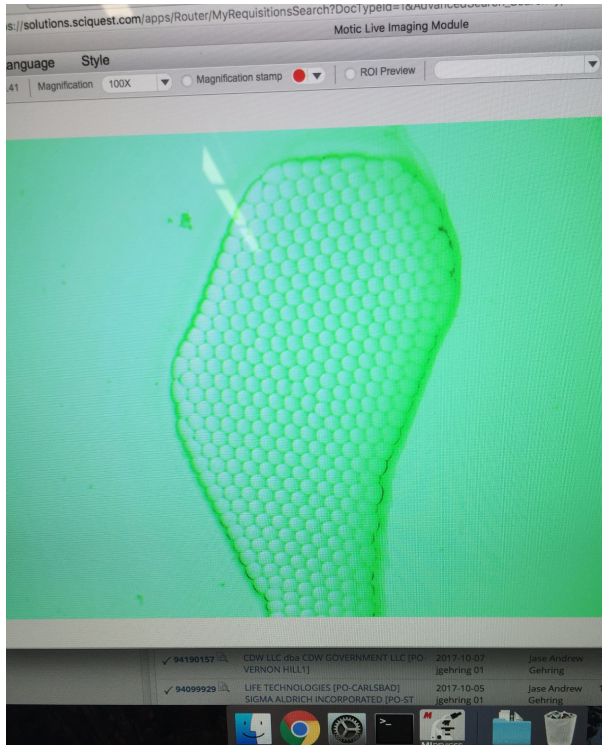
Notice something wrong with the device?

Robustness: Emphasizing the initial approach

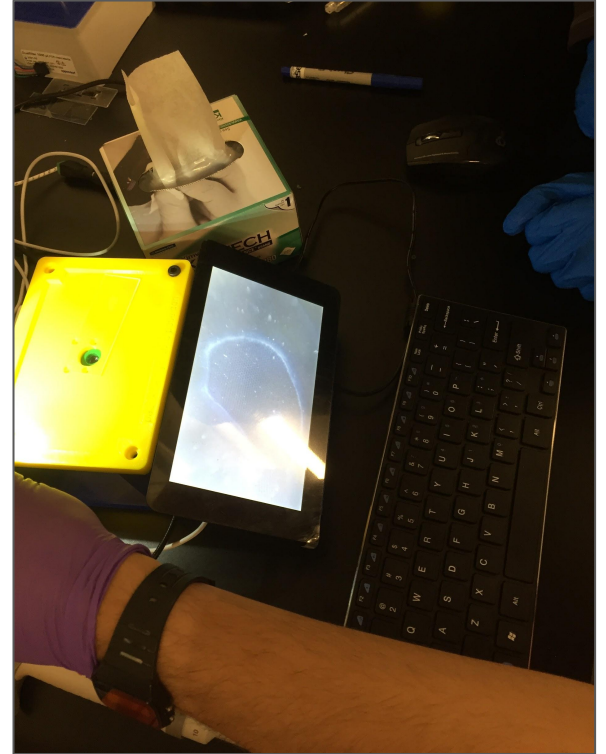
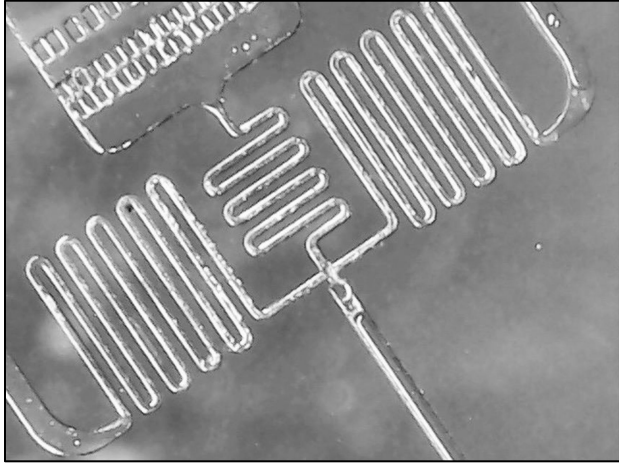


Notice something wrong with the device?

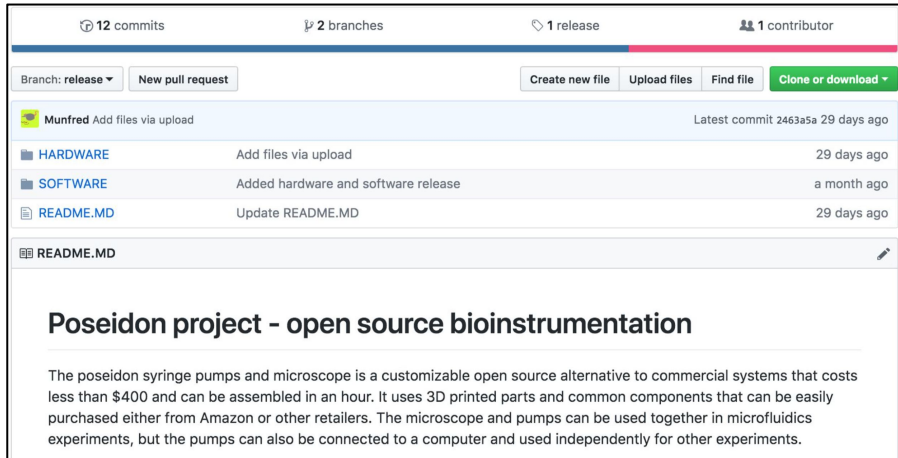
Benchmarking: Test and retest and retest



Benchmarking: Test and retest and retest



Documentation: The most important (and hardest) part of designing

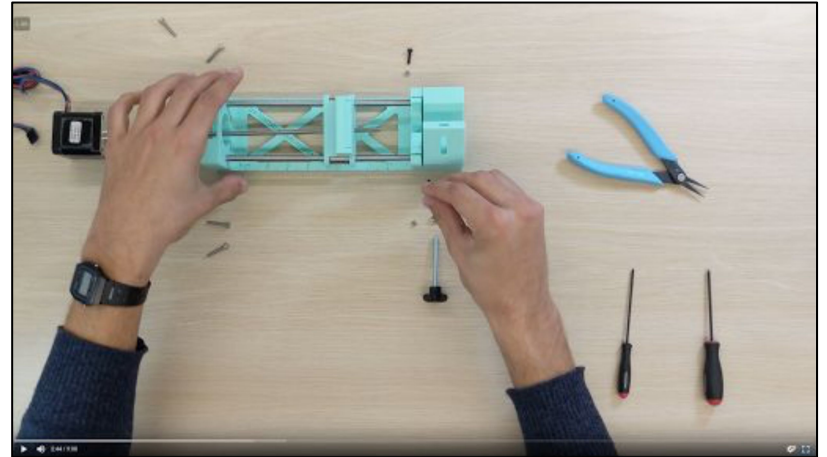


The screenshot shows the GitHub interface for the 'Poseidon project - open source bioinstrumentation'. At the top, it displays '12 commits', '2 branches', '1 release', and '1 contributor'. Below this, there are buttons for 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. A commit history table is visible, listing files like 'HARDWARE', 'SOFTWARE', and 'README.MD' with their respective commit dates. The main content area features the project title and a descriptive paragraph about the open-source bioinstrumentation.

File	Commit Message	Time Ago
Munfred	Add files via upload	Latest commit 2463a5a 29 days ago
HARDWARE	Add files via upload	29 days ago
SOFTWARE	Added hardware and software release	a month ago
README.MD	Update README.MD	29 days ago

Poseidon project - open source bioinstrumentation

The poseidon syringe pumps and microscope is a customizable open source alternative to commercial systems that costs less than \$400 and can be assembled in an hour. It uses 3D printed parts and common components that can be easily purchased either from Amazon or other retailers. The microscope and pumps can be used together in microfluidics experiments, but the pumps can also be connected to a computer and used independently for other experiments.



“Any code written by oneself six or more months ago should be considered someone else’s code”

Recap: Principles are good, only if you follow them

1. Functionality (Follow functional requirements)
2. Simplicity (Avoid complicated solutions)
3. Modularity (Use standard components)
4. Robustness (The “idiot user” approach)
5. Benchmarking (Test and retest)
6. Documentation (Videos, pictures, text)

Recommendation: print out a list like this and post it to your wall. It can help serve as a template for making design decisions.

A many thanks to those who helped on the project



Professor Lior
Pachter



Jase Gehring



Eduardo Da Veiga
Beltrame



Dylan Bannon

Project website: <https://pachterlab.github.io/poseidon/hardware>

If you like these kinds projects then reach out to us!

With the skills you will learn in Justin's class, we can work together to develop all sorts of novel bioinstruments. Our goal is to produce open, reliable, and modifiable bioinstruments for academic, medical, and research applications.

Examples of possible projects:

1. Fast Pressure Liquid Chromatography (Protein purification, liquid handler)
2. Automated fraction collector
3. Centrifuge microfluidics

Or just stop by our offices in the basement of Kerckhoff to check out our lab.

Contact: Sina Booeshaghi (abooesha@caltech.edu)