Supplementary Information for "An Open-Source Platform Enabling Continuous, Low-Flow Control for Long-Term Operation in Biotechnology Systems"

Hamed Rastaghi[1], Kara A. Walp[2,3], Bukola A. Akindipe[1], Tolulope Oyeniran[1], Yash H. Patel[2,3], Samuel M. D. Oliveira[1,3,*]

[1]Joint School of Nanoscience and Nanoengineering, North Carolina A&T State University, Greensboro, NC 27401, USA

[2]Department of Biomedical Engineering, Boston University, Boston, MA 02215, USA

[3]Department of Computer and Electrical Engineering, Boston University, Boston, MA 02215, USA

[*]Corresponding Author: smdoliveira@ncat.edu

Table of Contents

Supplementary Methods

S1. Minimum Flow Rate Calculation

S1.1 Positional Mode

In positional actuation mode, fluid delivery is generated in discrete angular steps of the actuator, with each increment corresponding to a fixed volume displacement determined by the syringe geometry and actuator resolution.

The linear displacement per degree of motor rotation is determined by the actuator's mechanical design. For the current system, the actuator provides approximately 0.256 mm of linear motion per degree of rotation. The minimum controllable angular increment of the servo motor is approximately 1.5 degrees, corresponding to a minimum linear displacement per step of:

$$\Delta x = (0.256\, mm/degree) \times (1.5\, degrees) \quad (1)$$

The volume displaced per step is then given by:

$$\Delta V = \pi \left(\frac{ID}{2}\right)^2 \times \Delta x \quad (2)$$

where ID is the inner diameter of the syringe.

This defines the minimum volume increment per actuation step, which represents the resolution of fluid delivery in positional mode.

Table S1: Reachable Flow Rates by the DIY pump with various market-available syringe sizes. * Each inner diameter corresponds with the standard dimensions for each syringe size.

Syringe Volume (µL)	Syringe ID (mm)*	Volume per Minimum Increment (µL)
10	0.485	71E-3
50	1.03	310.4E-3
100	1.46	630E-3
250	2.30	1.564E-2
1000	4.61	6.284E-2

S1.2 Continuous Mode

In continuous actuation mode, fluid delivery is generated by sustained rotation of the servo motor, and the resulting flow rate is determined by the motor's angular velocity and the cross-sectional area of the syringe. The volumetric flow rate can be expressed as:

$$Q = A \cdot v \quad (3)$$

where $A = \pi(ID/2)^2$ is the cross-sectional area of the syringe and v is the linear velocity of the plunger, which is proportional to the motor's angular velocity through the actuator conversion factor. The minimum achievable flow rate in this regime is limited by the motor's minimum stable rotational velocity. The servo is driven by pulse-width modulation (PWM) signals centered around a neutral value corresponding to zero velocity. In this system, stable continuous rotation is only achieved above a minimum control offset of approximately 57 (relative to the neutral signal of

1492 µs), below which the motor exhibits unstable or intermittent motion. This threshold defines the lower bound of achievable flow rates in continuous operation. Following calibration (Section S4), the relationship between the PWM input and motor velocity was determined experimentally by measuring the actuator displacement at different control signals and interpolating between them. Using this experimentally derived mapping, the plunger velocity corresponding to the minimum stable continuous input was estimated to be approximately 102 mm/min. For the 50 µL Hamilton syringe used in this work (inner diameter = 1.03 mm), this velocity yields a flow rate of approximately 85 µL/min. This value establishes the lower bound of continuous-mode operation in the present system.

S2. Hardware Design and Mechanical Assembly

This section describes the step-by-step mechanical assembly of the DIY dual syringe pump used in this work. The pump consists of a 3D-printed chassis housing a rack-and-pinion drive mechanism powered by a single servo motor, together with 2 solenoid pinch valves (three-way valves) and 1 two-way pinch valve for flow control. All structural components were fabricated by fused deposition modeling (FDM) from polylactic acid (PLA) filament. Assembly requires only basic hand tools and can be completed in approximately 30–45 minutes. An exploded-view diagram of the pump is provided in Supplementary Figure S1. A step-by-step video demonstration of the mechanical assembly process is available at: https://samoliveiralab.github.io/DIY_DSCPM/.

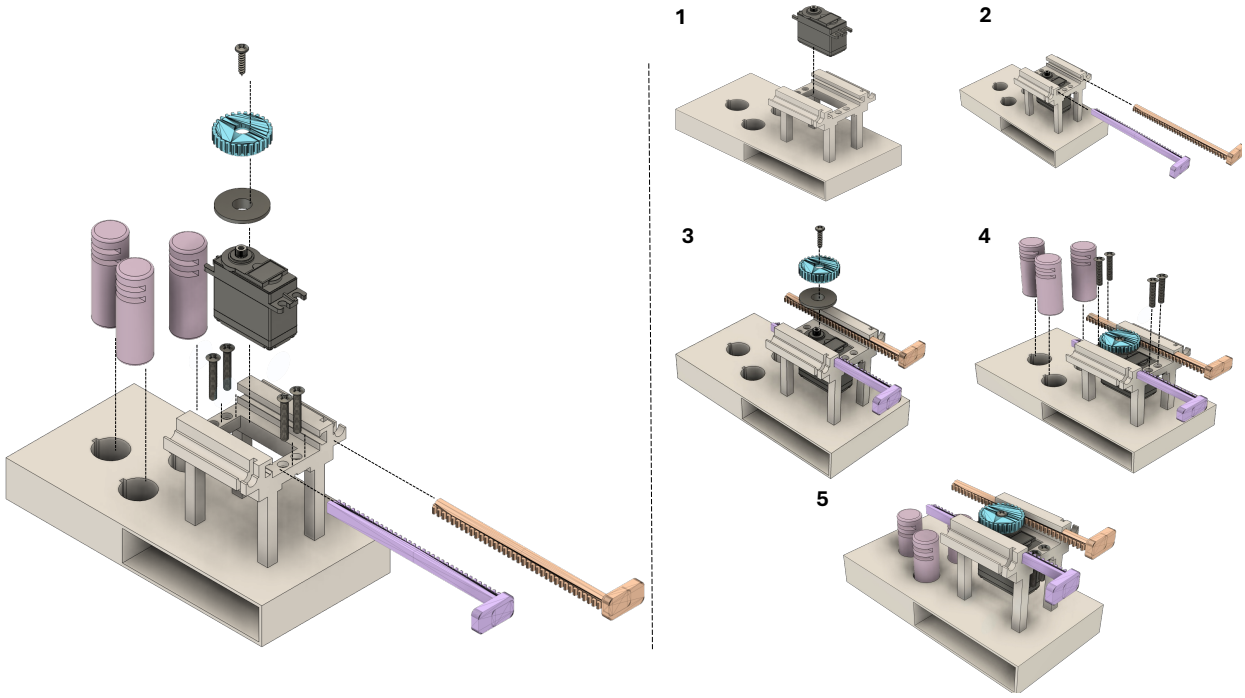

Figure S1. Exploded view of the DIY dual-syringe pump. The pump components, including the 3D-printed chassis, syringe pushers, rack-and-pinion mechanism, servo motor with disc and pinion gear, and mounting hardware. The exploded representation illustrates the spatial relationships between structural, actuation, and fastening elements prior to assembly.

S2.1 Components

S2.1.1 3D-printed parts

Three distinct parts were printed from PLA filament (layer height: 0.2 mm; infill: 30–50%). Design files (.STL) are available in the supplementary repository at https://github.com/SamOliveiraLab/DIY_DSCPM. Table S2 lists all printed components.

Table S2. 3D-printed components for the DIY dual syringe pump.

Qty.	Part	Material	Description
1	Chassis (base)	PLA	Base plate with rail slots, integrated rack gear teeth, and pinch valve holders
2	Syringe pushers	PLA	Linear pusher blocks; slide along chassis rails to advance syringe plungers
1	Pinion gear	PLA	Drives the pushers via meshing with rack gear on chassis

S2.1.2 Off-the-shelf components

Table S3 lists the commercially sourced components. Several fasteners are included with the servo motor package and do not need to be purchased separately.

Table S3. Off-the-shelf components and fasteners.

Qty.	Component	Specification	Source
1	Servo motor	goBILDA 2000-0025-0003; dual mode; 25-tooth spline output; 135:1 gear ratio; stall torque 9.3 kg·cm at 6.0 V	goBILDA
1	Servo disc (horn)	Included with servo motor	Servo package
3	Solenoid pinch valves	2 three way valves and 1 two way valve	Precigenome
4	M4 screws and nuts	M4 socket head cap screws with matching hex nuts	commercially available
1	Horn screw	Self-tapping, flanged Phillips head	Servo package
1	Shaft bolt	Socket head cap screw	Servo package

S2.1.3 Tools

Assembly requires a small Phillips-head screwdriver, a pin vise, and, optionally, needle-nose pliers for positioning nuts within the chassis cavity.

S2.2. Step-by-Step Assembly Procedure

The mechanical assembly is completed in seven steps, described below and illustrated in Supplementary Figure S1.

(1) **Fabricate the 3D-printed components.** Print the chassis, two syringe pushers, and the pinion gear from the supplied STL design files using the settings listed in Section S2.1. After printing, remove all support material and clean up any stringing or rough edges along the rail slots and gear teeth. Verify that the two pusher blocks slide freely along the chassis rails and that the rack gear teeth on the chassis are cleanly defined.

(2) **Mount the servo motor to the chassis.** Position the servo motor into the designated cavity on the chassis and align the four mounting holes on the servo flanges with the corresponding holes on the chassis. Insert four M4 socket head cap screws through the chassis and into the servo flanges, securing each with an M4 hex nut on the opposite side. Tighten all four fasteners evenly until snug; the motor output shaft should protrude from the chassis and rotate freely.

Caution: *Do not overtighten the screws, as excessive torque may crack the PLA mounting points.*

(3) **Insert the syringe pushers into the chassis rails.** Slide the two pusher blocks into the parallel rail slots on the chassis. Confirm that each pusher translates smoothly along its rail with minimal lateral play. Orient each pusher so that the flat face, which will contact the syringe plunger, faces outward, away from the central gear mechanism.

(4) **Mate the pinion gear to the servo disc.** Take the 3D-printed pinion gear and the servo disc (horn) included in the servo motor package. Align the pinion gear concentrically on the servo disc, matching the center hole and any keying features. Press the two components together until the pinion gear sits flush against the disc surface.

(5) **Fasten the pinion gear to the servo disc.** Use a pin vise to drill a pilot hole through the pinion gear, making it easier to insert the self-tapping horn screw. Take the self-tapping horn screw from the servo package, then drive it through the pinion gear and into the servo disc until it is snug. This creates a secure pinion gear–disc subassembly.

(6) **Secure the gear–disc assembly to the servo shaft.** Place the pinion gear–disc subassembly onto the servo motor output shaft, aligning the spline. Press firmly until the disc seats fully. Thread the socket head cap screw into the center of the servo shaft, then tighten it with the screwdriver. Verify that the pinion gear teeth mesh cleanly with the rack gear on the chassis and that the drive train rotates without binding or skipping.

(7) **Install the solenoid pinch valves.** Seat the 3 solenoid pinch valves into the integrated valve holders on the chassis. Thread the silicone tubing through each valve body so it passes through the valve aperture. Route the free ends of the tubing as required for the fluidic configuration. Route the valve wiring leads (yellow and black) toward the electronics mounting area for subsequent connection.

S2.3 Post-Assembly Verification

Before proceeding to electronics integration, the following checks should be performed:

(1) Manual rotation: Rotate the servo shaft by hand and verify that both pushers translate linearly along the chassis rails without binding or tooth skipping.

(2) Gear mesh: Confirm that the pinion gear is firmly seated on the servo disc and that no gear teeth are chipped or deformed.

(3) Valve seating: Verify that both pinch valves are fully seated in their chassis holders and that tubing passes through without kinks.

(4) Fastener integrity: Confirm that all screws and bolts are snug and that no components are loose.

S3. Electronics and Fluidic Integration

S3.1 Circuit Design

The DIY pump circuit is assembled on a standard breadboard using an Arduino Uno R3, three N-channel MOSFETs (RFP30N06LE), three 1 kΩ resistors, three IN4007 flyback diodes, male-to-male jumper wires, solenoid pinch valves, and a servo motor. Two separate power supplies are used: a 12V 2A supply for the solenoid pinch valves and a 6V 2A supply for the servo motor. The complete circuit schematic is shown in Figure S2.

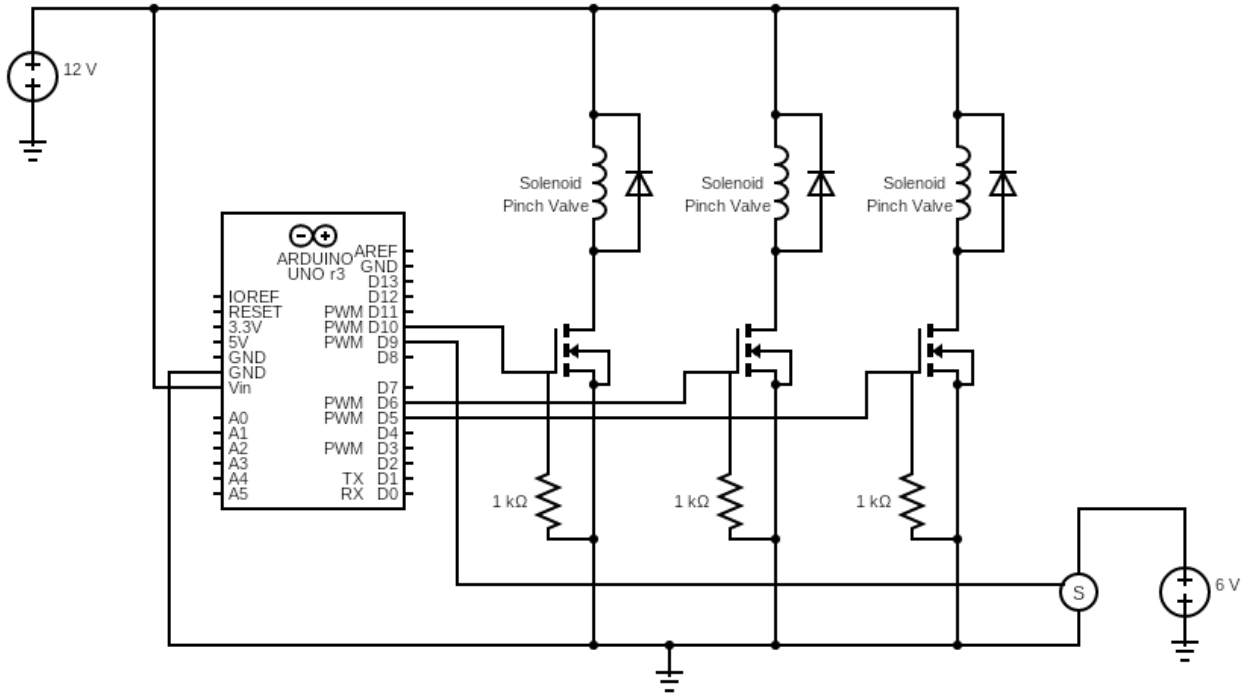

Figure S2. DIY pump Circuit Schematic. All diodes are IN4007, and the Servo Motor is a 6V High-Torque Servo Motor. Two-amp power supplies were utilized. When making a DSCPM 2-2, the left-most solenoid (connected to Pin D10) is omitted.

S3.1.1 Design Steps

(1) **Component Positioning**. Position the Arduino Uno R3 beside the breadboard. The breadboard has two power rail pairs (positive and negative buses on each side). Designate one rail pair for the 12V power input and the other for the 6V input.

(2) **Power Connections.** Connect a jumper wire from the Arduino GND pin to the negative (-) rail of the breadboard to establish a common ground. Then run a wire from the 12V positive rail to

the Arduino's Vin pin, which powers the microcontroller via its onboard regulator. Connect the 6V supply to the opposite rail pair in the same manner.

(3) **MOSFET Placement.** Insert the three N-channel MOSFETs (RFP30N06LE) into the breadboard. Each MOSFET acts as a switch for one solenoid pinch valve. With the flat face of the package facing you, the pinout from left to right is Gate, Drain, Source.

(4) **Gate Resistor and Digital Pin Connections.** For each MOSFET, connect a 1 kΩ resistor in series with the Gate pin. One end of the resistor connects to the Gate and the other end extends via jumper wire to a PWM-capable digital pin on the Arduino. The three gates connect to pins D10, D6, and D5, respectively. Connect the Source pin of each MOSFET to the common ground rail.

(5) **Solenoid Valve Connections.** Connect the Drain (center pin) of each MOSFET to one terminal of its corresponding solenoid pinch valve. Connect the other terminal of each solenoid valve to the 12V positive rail. When the Arduino sets a digital pin HIGH, the MOSFET conducts, completing the circuit and energizing the solenoid.

(6) **Flyback Diode Installation.** For each solenoid valve, connect an IN4007 diode in reverse bias across the valve terminals: the cathode (banded end) connects to the 12V positive rail and the anode connects to the MOSFET drain side. This flyback diode absorbs the voltage spike generated when the solenoid's magnetic field collapses upon de-energization, protecting the MOSFET from damage.

(7) **Servo Motor Connection**. The servo motor has three wires: signal, power, and ground. Connect the signal wire (typically white or orange) to Arduino PWM digital pin D9. Connect the power wire (red) to the 6V positive rail on the breadboard. Connect the ground wire (brown or black) to the common ground rail. The 6V power supply provides dedicated power to the servo, preventing current draw from affecting the Arduino's logic circuits.

(8) **Verification**. Before powering on, verify all connections against the circuit schematic (Figure S2). Ensure that no short circuits exist between the 12V and 6V rails, that all MOSFET orientations are correct (flat face, Gate-Drain-Source left to right), and that all flyback diodes are oriented with the cathode toward the 12V positive rail. Upload the firmware to the Arduino and confirm the "READY" handshake appears in the serial monitor at 9600 baud.

S3.2 Fluidic Connections and Tubing

After completing the mechanical assembly and verifying proper operation, the fluidic connections must be established to enable liquid handling. This section describes the tubing routing, adapter fabrication, and valve connections required to build a functional dual-syringe pump system. The fluidic assembly is illustrated in Figure S3.

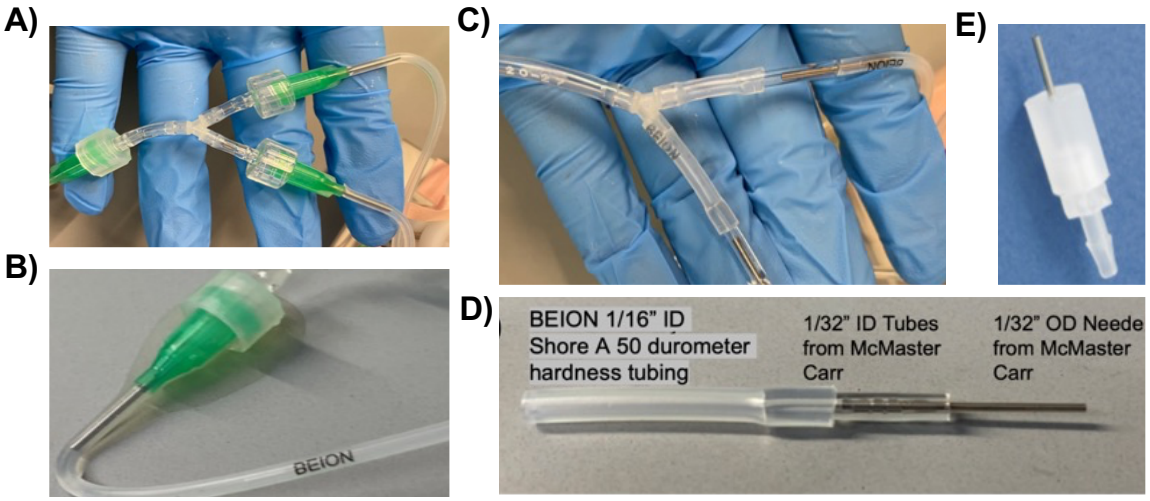

Figure S3 - (A) Adapting from a Y-connector (meant for 1/16" ID tubing) to 1/32" ID tubing with multiple Luer connections, which have a lot of dead volumes, which can introduce the risk of residual bubbles in the system from the assembly. (B) Leakage of media from a Luer connection. (C) Adapting from a Y-connector to 1/32" ID tubing with custom adapters. (D) The custom adapter schematic. The needle was obtained from a Luer tip/hub to needle adapter by compressing the plastic with pliers until it cracked and pulled the needle out. (E) A market-available adapter from 1/16" ID to 1/32" ID has no dead volume but is quite expensive compared to the custom adapters we made.

S3.2.1 Tubing and adapter selection

The fluidic system uses two tubing sizes: 1/16″ inner diameter (ID) Shore A 50 durometer silicone tubing (BEION) for the main inlet and outlet lines, and 1/32″ ID tubing (McMaster-Carr) for the intermediate lines routed through the pinch valves. Y-connectors are used at four junction points to split or merge flow. Transitioning between the 1/16″ and 1/32″ tubing diameters requires appropriate adapters.

Commercial Y-connectors designed for 1/16″ ID tubing can be adapted to 1/32″ ID lines using standard Luer fittings (Figure S3a). However, this approach introduces excessive dead volume at the multiple Luer junctions and increases the risk of residual air bubbles in the system. Additionally, Luer connections are prone to leakage under the pressures generated by the syringe pump (Figure S3b).

S3.2.2 Custom adapter fabrication

To minimize dead volume and eliminate leak-prone Luer connections, we fabricated custom adapters that transition directly between 1/16″ and 1/32″ tubing (Figure S3 C, D). A blunt-tip needle (1/32″ OD; McMaster-Carr) was extracted from a commercial Luer tip/hub-to-needle adapter by compressing the plastic hub with pliers until it cracked, then pulling the needle free. The extracted needle was inserted into one end of a short segment of 1/16″ ID silicone tubing, while 1/32″ ID tubing was press-fitted onto the opposite end of the needle. This creates a low-dead-volume, leak-free transition without requiring Luer fittings.

Note: A commercially available 1/16" OD barbed-tip-to-1/32" OD needle adapter (Component Supply) can serve as an alternative but is considerably more expensive (Figure S3 E).

S3.2.3 Fluidic routing

(1) **Syringe to Y-connector split.** Each syringe outlet is connected to a length of 1/16″ ID tubing, which feeds into a Y-connector via a custom adapter. The Y-connector splits the flow into two 1/32″ ID lines: one directed upward (toward the reservoir) and one directed downward (toward the output).

(2) **Valve-controlled lines.** Each of the four 1/32″ lines (two per syringe) pass through a three-way solenoid pinch valve. The valves control whether each syringe is drawing fluid from the reservoir (refill mode) or pushing fluid to the output (dispense mode). By alternating which valves are open, one syringe can refill while the other dispenses, enabling continuous uninterrupted flow.

(3) **Reservoir merge.** The two upward 1/32″ lines (one from each syringe) converge at a Y-connector and merge into a single 1/16″ ID line connected to the fluid reservoir.

(4) **Output merge**. The two downward 1/32″ lines (one from each syringe) converge at a Y-connector and merge into a single 1/16″ ID output line. A two-way solenoid pinch valve is installed on this final output line to provide an additional point of flow control before the fluid reaches the target vessel or microfluidic device.

(5) **Prime the system.** Before beginning an experiment, prime all tubing lines by manually advancing the syringe plungers until fluid flows continuously from the output and no air bubbles remain in the lines. Inspect each connection point and adapter for leaks.

S4. Calibration

S4.1 Continuous Mode

In continuous actuation mode, calibration consists of (i) determining the neutral PWM value of the servo and (ii) establishing the relationship between PWM offset and resulting flow rate.

S4.1.1 Neutral PWM Calibration

Before performing flow calibration, the neutral PWM value of the servo must be determined experimentally, as this value may vary between motors. The neutral PWM is defined as the input at which the servo produces no sustained rotation. To determine this value, first allow the syringe pusher to move freely and ensure that the system is not connected to a sensitive experiment. In the Continuous.ino code, set the PWM offset variable (*offset*) to zero so that the applied signal corresponds directly to the neutral PWM value (*SERVO_NEUTRAL_US*). Initialize *SERVO_NEUTRAL_US* to 1500 µs, which corresponds to the expected neutral region. Run the system and observe the pusher motion. If the pusher moves forward, decrease the value of *SERVO_NEUTRAL_US* slightly; if it moves backward, increase it slightly. After each adjustment, re-run the system and observe the response. Continue this process in small increments until no sustained pusher motion is observed in either direction. Once this condition is achieved, record the corresponding value of *SERVO_NEUTRAL_US* as the neutral PWM for that motor. In this work, the neutral PWM value was determined to be 1492 µs, and all PWM offsets were defined relative to this value.

S4.1.2 Flow Calibration

Following neutral calibration, the relationship between PWM offset and flow rate was determined experimentally. To perform the calibration, operate the system in continuous mode and allow the syringe pusher to move freely. Identify the two extreme positions of the pusher travel (forward and backward limits) and mark these locations. The distance between these two marks defines the reference travel distance. Calibration was carried out by assigning different values to the PWM offset variable (offset) in the Continuous.ino code, while keeping the neutral PWM value (SERVO_NEUTRAL_US) fixed. For each calibration point, the offset variable was set to a

predefined value, starting from 57 (the minimum stable offset) and increasing uniformly up to 87. The offset values used were: 57, 62, 67, 72, 77, 82, and 87. For each offset value, run the system continuously and allow the pusher to traverse between the marked positions repeatedly. Measure the total time required for five consecutive traversals using a timer. The pusher velocity is then calculated as:

$$v = \frac{5 \times \text{distance}}{\text{time}} \quad (4)$$

To improve accuracy, this measurement is repeated twice for each offset, and the average value is used. For each offset, the calculated pusher velocity is converted to flow rate using the syringe cross-sectional area, resulting in a discrete set of calibration pairs (offset, Q). These values should be set as parameters in the Continuous.ino code. To enable continuous control, piecewise linear interpolation is applied between adjacent calibrated points to estimate the PWM offset required for intermediate flow rates. For a desired flow rate Q^* between two calibrated points (offset_i, Q_i), (offset_{i+1}, Q_{i+1}) , the corresponding offset is calculated as:

$$\text{offset}^* = \text{offset}_i + \frac{(Q^* - Q_i)}{(Q_{i+1} - Q_i)}(\text{offset}_{i+1} - \text{offset}_i) \quad (5)$$

This approach provides a simple, robust, and computationally efficient mapping from desired flow rate to PWM input, enabling accurate and reproducible operation in continuous mode.

S4.2. Positional Mode

In positional actuation mode, calibration is performed during system setup by defining the syringe geometry and aligning the actuator with the syringe plungers. The inner diameter of the syringe is specified in the Positional.ino code (*innerdiameter parameter*) so that the correct cross-sectional area is used in flow calculations. If a different syringe is used, this parameter must be updated accordingly. Similarly, if a different servo motor is used, the actuator scaling parameter (*degper180*) is adjusted based on the motor specifications.

Mechanical alignment is then carried out to ensure that the actuator displacement corresponds accurately to syringe plunger motion. The actuator is first moved to its minimum position, after which the syringe is placed, and the pusher is inserted and aligned with the plunger. The actuator is then moved to its maximum position, and the same procedure is repeated for the second syringe. This process ensures that both syringes operate within the full usable stroke range of the actuator and that the pushers are correctly engaged with the plungers.

S5. Software Control Architecture and GUI

The DIY pump software stack consists of a Python desktop application built with PyQt5 (pump_app.py), a serial communication wrapper (arduino_cmds.py), a USB auto-detection module (autoport.py), and the Arduino firmware (Continuous.ino and Positional.ino). These components are organized around a serial protocol operating at 9600 baud over USB. On power-up, the Arduino transmits the string “READY” over serial; the Python software blocks until this handshake is received before issuing any commands. All subsequent communication follows the same path regardless of which control mode is active: the Python layer encodes a command as a newline-terminated ASCII string, transmits it over the serial connection, and the Arduino firmware parses and executes it by actuating the servo motor and solenoid pinch valves.

Three software control modes are supported, corresponding to increasing levels of autonomy (Figure 4 in the manuscript). In manual mode, the user interacts directly with the graphical user interface to set the flow rate, select a flow behavior, toggle the pump state, and change the flow direction; changes are transmitted to the Arduino immediately and take effect without interrupting the experiment. In pre-programmed mode, the user uploads a plain-text command file containing a series of timed instructions; the software parses the file and dispatches each command to the

Arduino at the scheduled time, allowing unattended operation over the full duration of the experiment. In ML-connected mode, an external Python script imports the serial communication module and sends commands programmatically, enabling closed-loop control in which a machine learning model or other adaptive algorithm reads sensor data, computes an optimal flow rate, and adjusts the pump in real time without human intervention. All three modes share the same downstream serial path and Arduino firmware; they differ only in how the command originates.

S5.1 Graphical User Interface

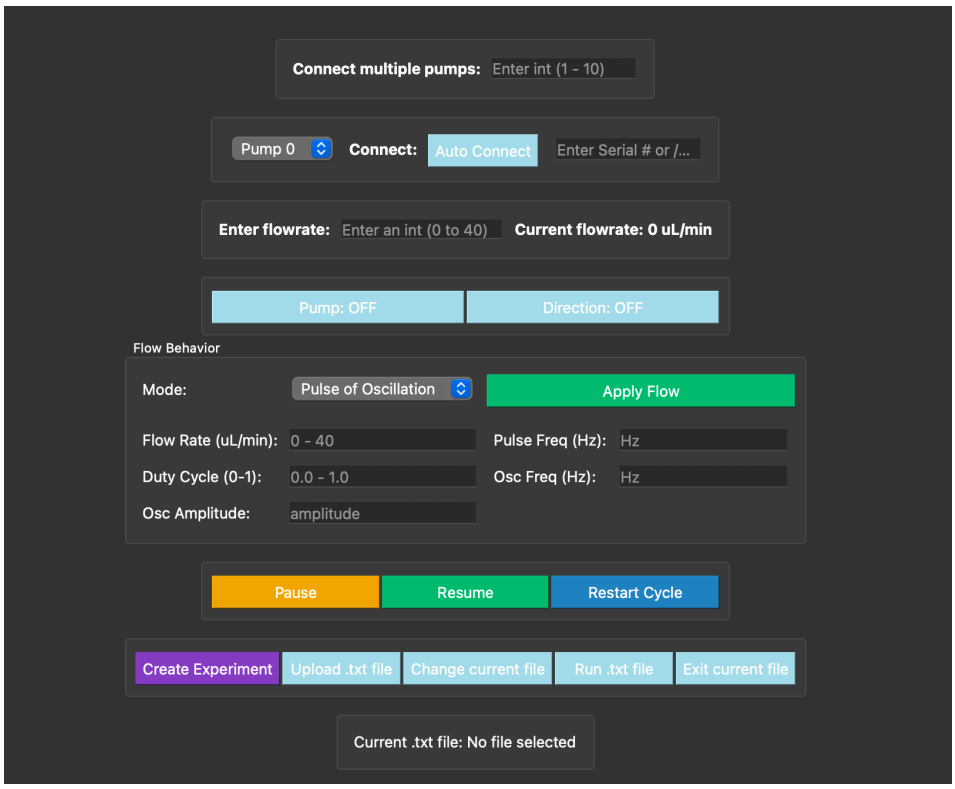

Figure S4 Graphical User Interface - The DIY pump controller GUI showing (A) the connection and pump selection panel, (B) flow rate and pump state controls, (C) flow behavior mode selection with dynamic parameter fields, and (D) the file upload and experiment scheduling section with command preview.

The GUI is implemented as a single scrollable window organized into functional sections from top to bottom (Figure S4). The connection section provides two methods for establishing communication with the Arduino: auto-detection, which scans connected USB devices and connects to the first likely Arduino port, and manual entry, in which the user types a USB serial number or device path. For multi-pump setups, the user specifies the number of pumps (up to ten), enters the serial number for each, and selects between them using a dropdown. All connected boards are addressed independently, allowing multiplexed control from a single GUI instance.

Below the connection section, the flow rate field accepts a value in microliters per minute (0-40), which is sent directly to the Arduino as a numeric string. The Arduino recalculates the servo step delay from the syringe geometry and begins operating at the new rate without pausing the experiment. The pump state section provides toggle buttons for turning the pump on or off (which

sends the command 123 or 0 to the Arduino, respectively) and for reversing the flow direction (command 321). When the pump is turned off, the Arduino saves its current position, direction, infuse state, and valve state to EEPROM, allowing the pump to resume from its last known configuration after a power cycle.

The flow behavior section provides a dropdown for selecting one of four modes: constant, pulse, oscillation, or pulse of oscillation. Parameter fields appear dynamically depending on the selected mode. Constant flow requires only the flow rate. Pulse flow adds duty cycle (0-1) and pulse frequency (Hz). Oscillation adds oscillation frequency (Hz) and amplitude. Pulse of oscillation combines all parameters. Clicking the Apply button constructs a comma-separated command string (e.g. FLOWA,10.0 for constant flow at 10 µL/min, or FLOWB,15.0,0.5,2.0 for pulse flow) and transmits it to the Arduino.

The file control section allows the user to upload a command file, view its contents in a human-readable preview, run the scheduled commands, merge additional files into a running schedule, and stop execution. A “Create Experiment” dialog provides a visual interface for building command sequences without manually writing the text file: the user selects a pump, enters a time, chooses a behavior, fills in parameters, and adds the step to a table. The dialog generates and saves the file in the required format. The scheduling engine executes commands on a background thread and supports pause (freezes the scheduler and shifts all remaining command times forward to preserve relative timing), resume (continues from the adjusted schedule), and restart (recalculates all command times from the original delays and begins the full sequence again). A video demonstration of the graphical user interface and software operation is available at: https://samoliveiralab.github.io/DIY_DSCPM/.

S5.2 Command File Format and Experiment Scheduling

Pre-programmed experiments are specified in plain-text files using a three-field format. Each entry contains the target Arduino’s USB serial number, the command to execute, and the time in seconds from the start of the experiment, separated by the delimiters ********* and **#########**. Multiple entries are joined by **%%%%%%%%%**:

SERIAL*********COMMAND#########DELAY

%%%%%%%%%

SERIAL*********COMMAND#########DELAY

For example, the following file turns the pump on at t = 0 s, starts constant flow at 10 µL/min at t = 2 s, switches to pulse mode at t = 30 s, and stops at t = 60 s:

054433...B7D8*********123#########0

%%%%%%%%%

054433...B7D8*********FLOWA,10.0#########2

%%%%%%%%%

054433...B7D8*********FLOWB,15.0,0.5,2.0#########30

%%%%%%%%%

054433...B7D8*********0#########60

Alternatively, users can build command sequences visually using the Create Experiment dialog within the GUI, which generates the file automatically. The scheduling engine dispatches commands on a background thread and supports pause, resume, and restart. Multiple files can

be merged into a running schedule; new commands are inserted in chronological order alongside existing ones.

S5.3 - Installation and Setup

The software requires Python 3.8 or higher. Dependencies are installed with:

```
pip install PyQt5 pyserial
```

The Arduino code is compiled and uploaded to the Arduino Uno using the Arduino IDE. The firmware uses pins 9 and 11 for the servo motors and pins 5, 6, and 10 for the solenoid pinch valves, with communication at 9600 baud. If a syringe other than the default 10 µL Hamilton is used, the inner diameter variable (innerdiameter) must be updated. If a different servo motor is used, the degrees-per-180 variable (degper180) should be adjusted to match the motor's range.

To launch the application:

```
cd "Python code"
python GUI.py
```

On startup, the user connects to the Arduino via the Auto Connect button or by entering the device serial number manually. Once connected, the pump is controlled through the GUI. For pre-programmed experiments, the user uploads a command file or builds one with the Create Experiment dialog. For external script integration, the serial communication module can be imported directly:

```
import arduino_cmds, autoport
board, _ = autoport.connect()
board.sendcommand('123')          # pump on
board.sendcommand(str(new_rate))  # set flow
```

This enables integration with machine learning frameworks, PID controllers, or any Python-based algorithm for closed-loop adaptive flow control. All source code, firmware, and documentation are available in the project repository.

S6. Thermal Stability of Solenoid Pinch Valves

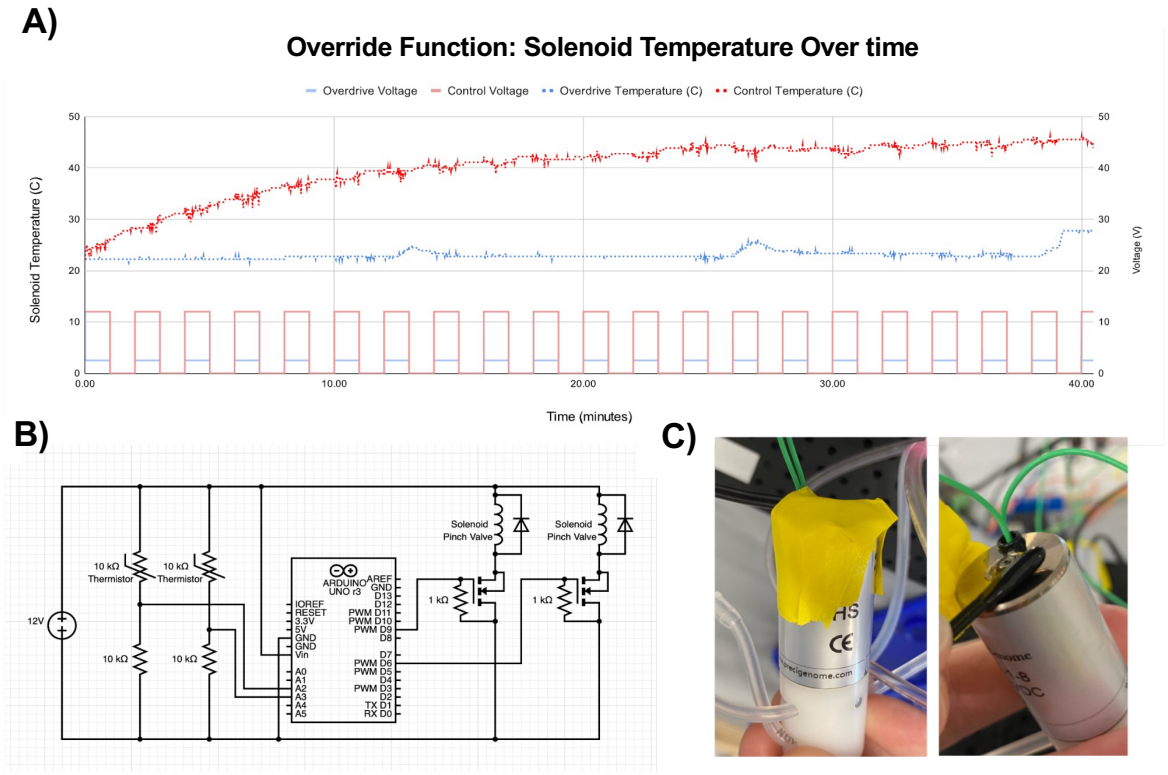

Figure S5 - Overdrive Function for long-term temperature control of BEION/PreciGenome 12V Solenoid Pinch Valves. Measurements were taken with a 10KΩ thermistor. (a) Solenoid Temperature versus time transposed with voltage duty cycle versus time. (b) Temperature Measurement setup circuit. (c) Thermistor placement on the base of the solenoid to maximize surface area in contact with the thermistor.

The thermal behavior of the solenoid pinch valves was evaluated to ensure reliable long-term operation of the system, as these components are repeatedly actuated during continuous pump operation. The valves were integrated with flyback diodes and driven using an overdrive control strategy to reduce steady-state power dissipation while maintaining sufficient actuation force. Temperature measurements were obtained using a 10 kΩ thermistor placed at the base of the solenoid to maximize thermal contact and capture representative device heating. As shown in Figure S5, the valve temperature was monitored over time alongside the applied voltage duty cycle. The overdrive scheme enables an initial high-power actuation followed by a reduced holding signal, preventing excessive heat accumulation during extended operation. This approach maintains the valve within a stable thermal regime, ensuring consistent actuation performance and minimizing the risk of thermal degradation during long-duration experiments.

S7. Microfluidic Application

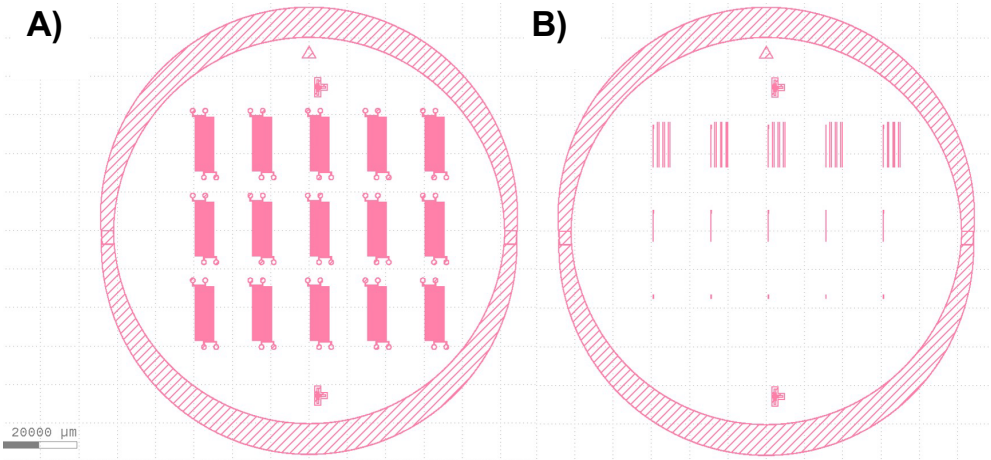

Figure S6 - (a) CAD model of 15 chips used to imprint on a photomask using the Mask Writer. This layer acts as the second layer of the PDMS chips, the channels. (b) The first layer of the PDMS chips with different numbers of MCs per row.

PDMS-based microfluidic devices containing monolayer chambers (MCs) were fabricated using a two-layer photolithographic process consisting of a chamber layer and a channel layer. The device geometry was designed in CAD and transferred to a photomask using a mask writer, as shown in Figure S6, where the first layer defines the monolayer chambers with varying chamber densities and the second layer defines the channel network for fluid delivery. These devices enable controlled confinement of microbial populations while supporting continuous media flow across the chambers. To evaluate the platform under biologically relevant conditions, Escherichia coli cells constitutively expressing cyan fluorescent protein (CFP) were introduced into the MCs, and continuous media flow was maintained at 1.5 µL/min for 24 hours using the DIY pump. This configuration enables sustained nutrient delivery and removal of excess cells, demonstrating that the system supports stable operation and is compatible with long-term microfluidic culture.

S8. Edge Detection Algorithm

Fluid front position was extracted from time-lapse videos using an intensity-gradient-based edge detection algorithm. The fluid front position is defined as the horizontal pixel coordinate corresponding to the leading boundary between the infused (dyed) fluid and the unfilled region within the tubing, identified as the first location where the intensity gradient exceeds a predefined threshold.

Videos were converted into sequences of frames, and each frame was preprocessed by grayscale conversion followed by Gaussian blurring (radius = 2 pixels) to reduce noise. For each frame, rows were scanned sequentially from top to bottom. Within each row, adjacent pixel intensities were compared, and the first column index satisfying

$$| I(x) - I(x-1) | > T \quad (4)$$

was identified as the fluid front position, where $I(x)$ is the grayscale intensity at column x, and T is an empirically selected threshold. The first valid detection across all rows was used as the fluid front position for that frame.

The detected positions (in pixels) were converted to volumetric displacement using spatial calibration based on the known tubing length and internal radius. Flow rate was computed as the discrete time derivative of the volume signal and scaled by the frame rate to obtain units of µL/min. A moving average filter (window size = 50 frames) was applied to reduce noise introduced by differentiation.

S9. Bill of Materials

Table S4: Bill of Materials. *The DIY pump in this table, uses BEION syringes purchased from PreciGenome, which cost about $100; however, the same valves are available from other suppliers for cheaper.

Part	Vendor	Amount	Price per Unit
Electronics Components:			
Arduino Uno	Any	**1**	~ $30
IN4007 Diodes	Any	**3**	< $1
1 kΩ Resistors	Any	**3**	< $1
N-Channel MOSFETS	Any	**3**	< $1
3-Way Solenoid Pinch Valve	PreciGenome* (USA) BEION (China)	**2**	$118* (PreciGenome) ~ $30 (BEION)
2-Way Solenoid Pinch Valve	PreciGenome*(USA) BEION (China)	**1**	$108* (PreciGenome) ~ $30 (BEION)
High Torque Servo Motor Dual Mode	GoBilda (USA)	**1**	~ $30
6V, 2A AC/DC Power Supply	Any	**1**	~ $12
12V, 2A AC/DC Power Supply	Any	**1**	~ $12
Fluidics Components:			

10 µL Syringe (Model 701 LT)	Hamilton (USA)	**2**	$36
Y-Connectors for 1/16” ID Tubing	Cole Parmer (USA)	**4**	< $1
1/16” ID Silicone Tubing	Any	**n/a**	< $1
1/16” ID Silicone Tubing: Shore A 50 Durometer	Any	**n/a** (comes with PreciGenome valves)	< $1
1/32” ID Silicone Tubing: Shore A 50 Durometer	Any	**n/a** (comes with PreciGenome valves)	< $1
Luer Tip/Hub - 1/16” Barbed Tip Adapter	Any	**2**	~ $2
Luer Tip/Hub - 21 Gauge Needle Adapter	Any	**4**	< $1
Total Prices:	**DIY Pump:**	**1**	**~ $500***