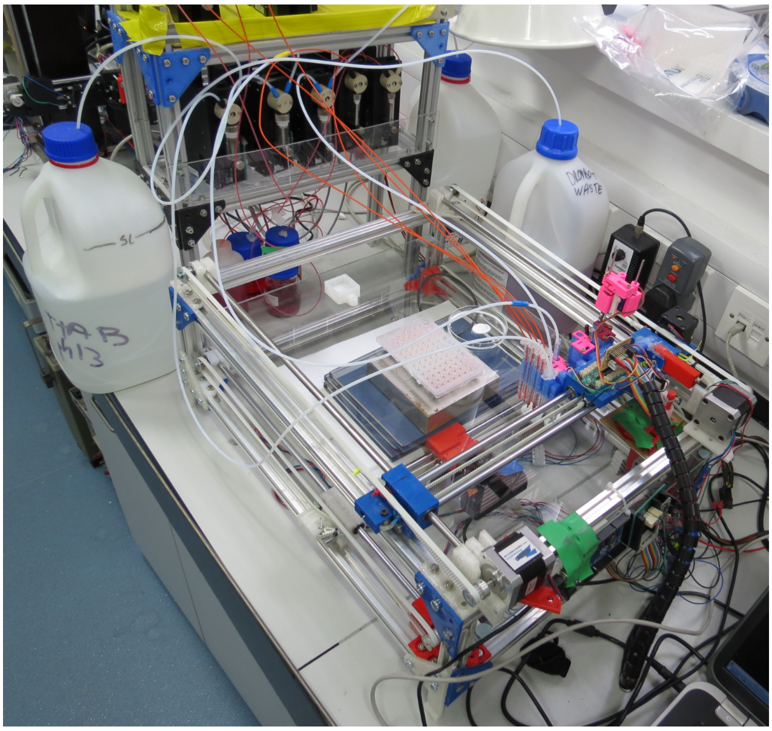
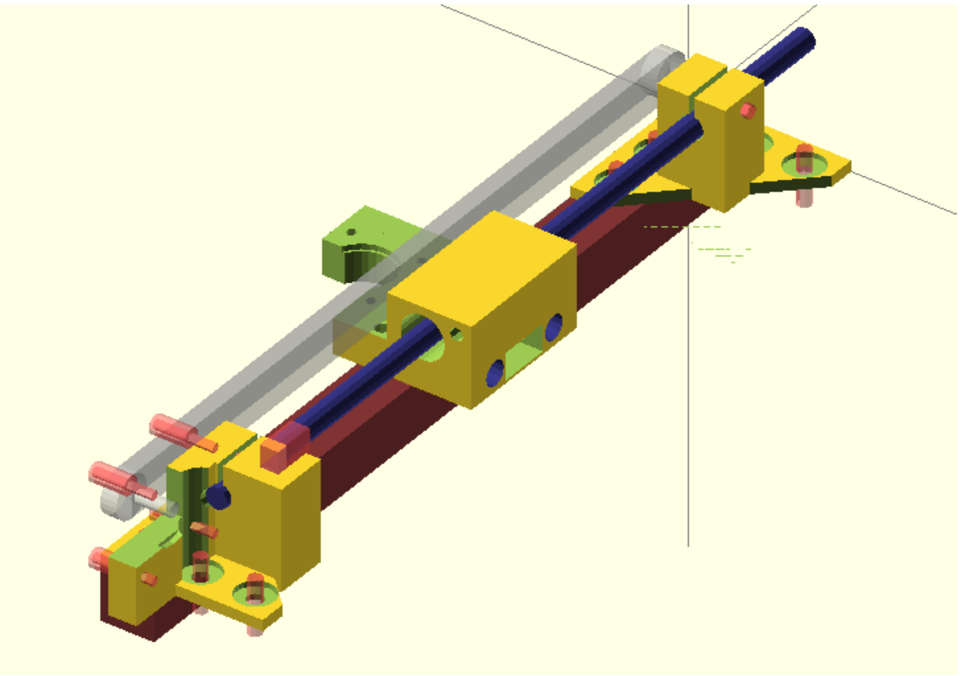
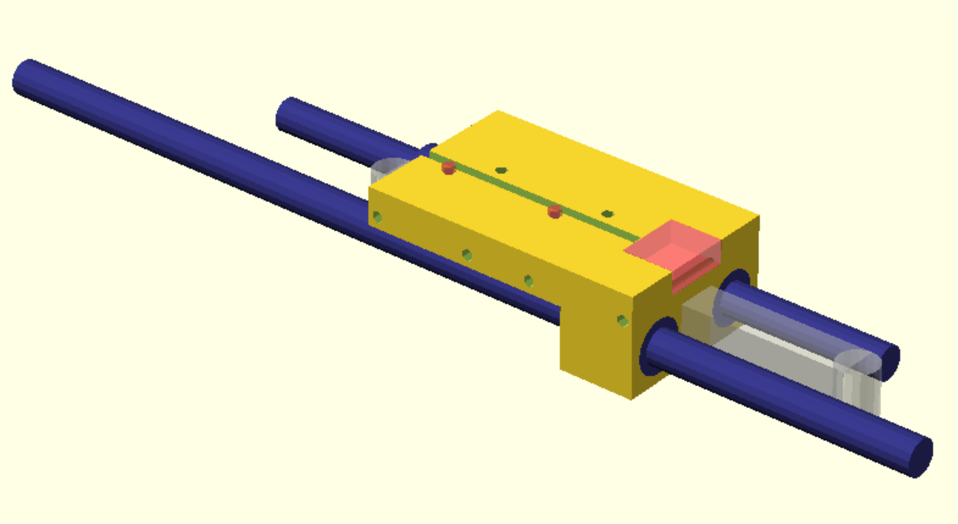
Supplementary Figures



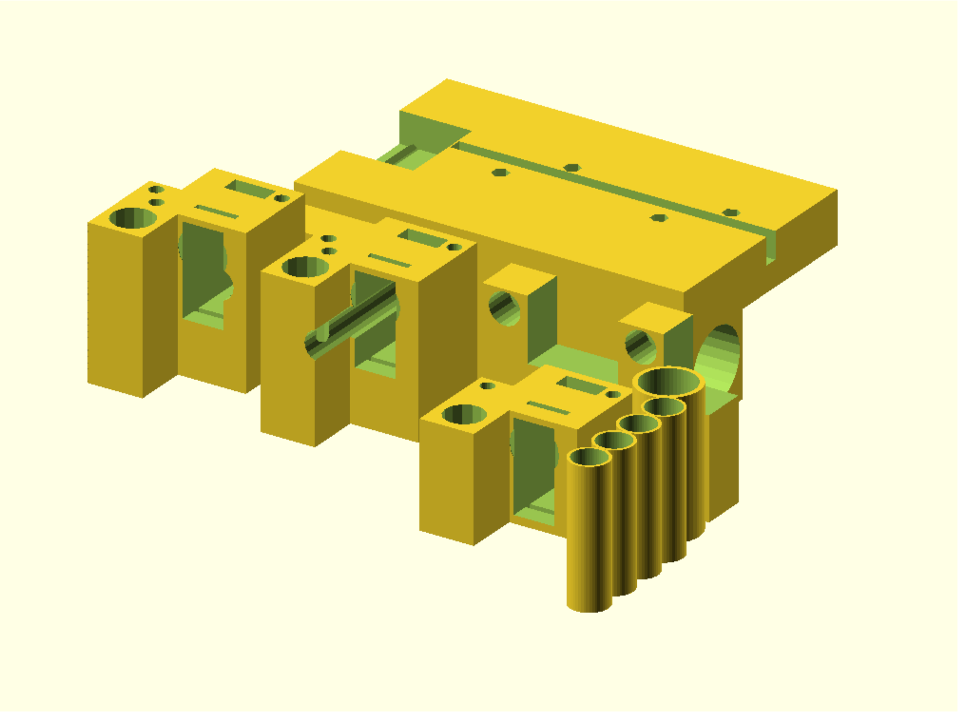
Supplementary Figure 1: Overall picture of the robot. Strut profiles were used to build the frame. They were joined using 3D printed pieces. The carriage which holds the syringes and tubes was also 3D printed. All the mechanisms were based on RepRap 3D printers. The long axis shown here is the X axis, with the shorter being the Y axis. The carriage is shown at home (0,0) position.



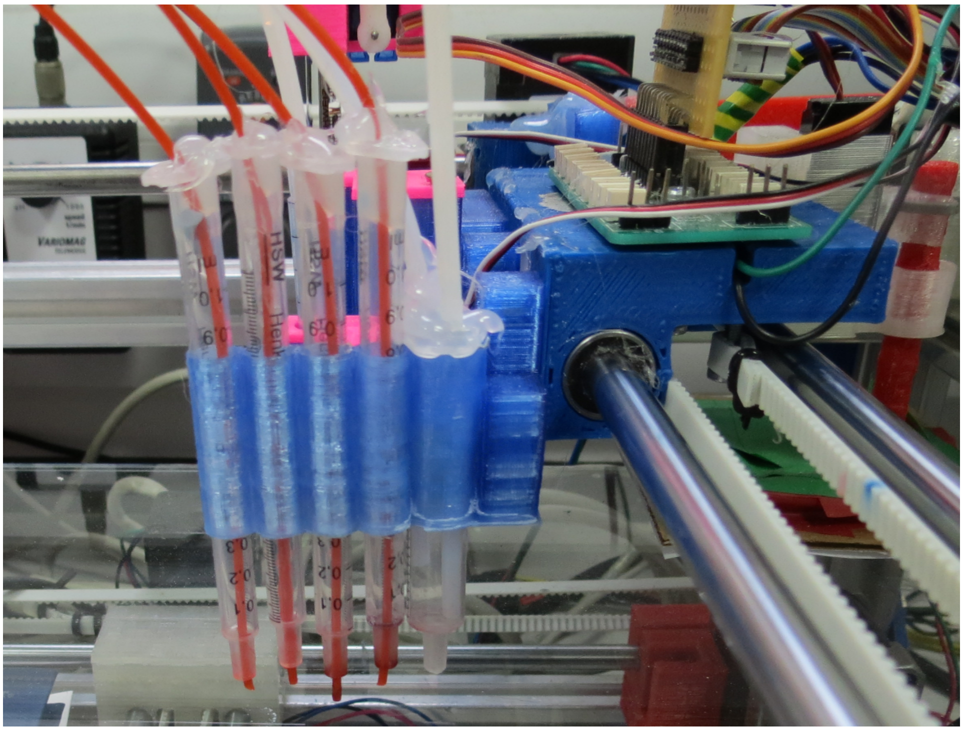
Supplementary Figure 2: X axis 3D design. **Blue**: The precision rod (8h7). **Grey**: The timing belt (T2.5x6mm). **Yellow**: The 3d-printed parts. **Dark red**: Aluminium strut profile (20x20mm). **Transparent red**: The parallelepiped near the motor is where the end stops were placed; other parts in this colour are machine screws used to fasten the parts together.



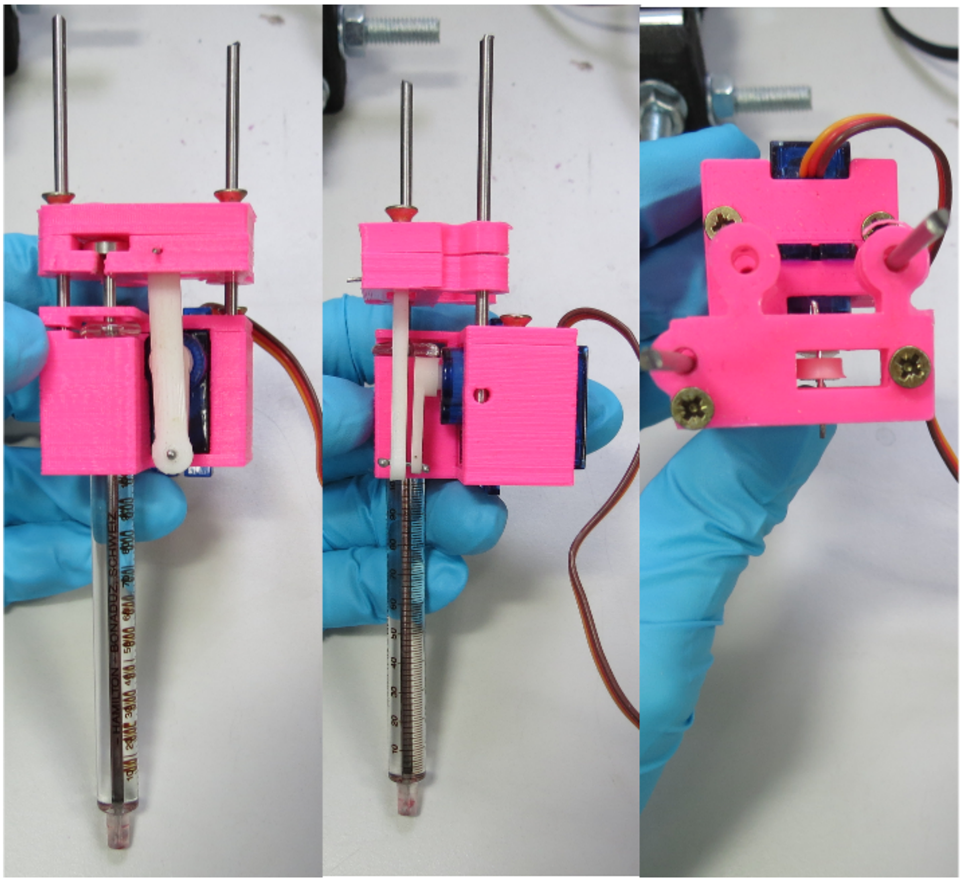
Supplementary Figure 3: Y axis 3D design. **Blue**, the precision rod. **Grey**, the belt. **Yellow** the printed parts. The motor that moved the carriage around the Y axis could be seen on Supplementary Figure 2. The red transparent parallelepiped is where the end stops were placed. The four holes on its side, and on the other side where it cannot be seen, were used to place the structures that would handle the syringes or any other equipment, making the design customizable.



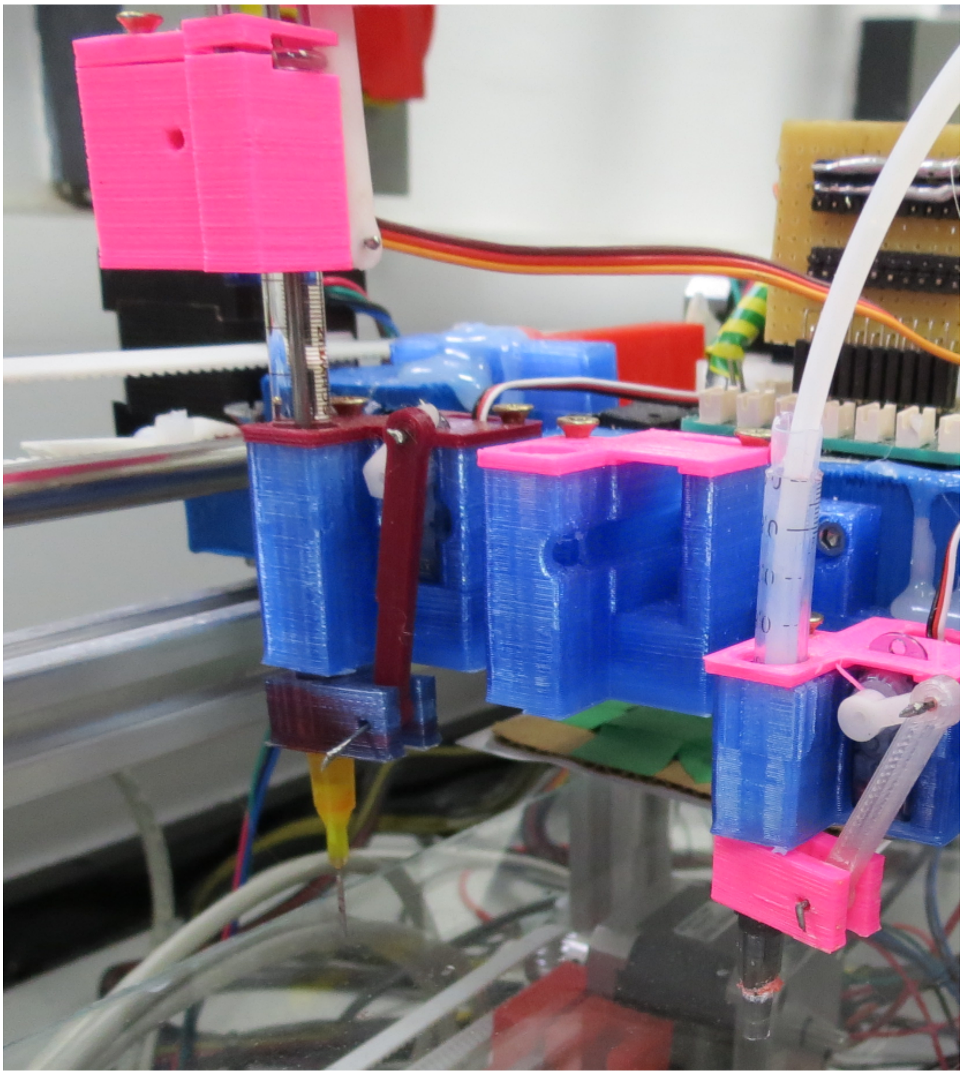
Supplementary Figure 4: The X-Y carriage. Assembled but without syringes. The locations for actuated syringes and fluidic connections to the fluid platform can be seen in the foreground. In the background is the attachment to the Y-axis.



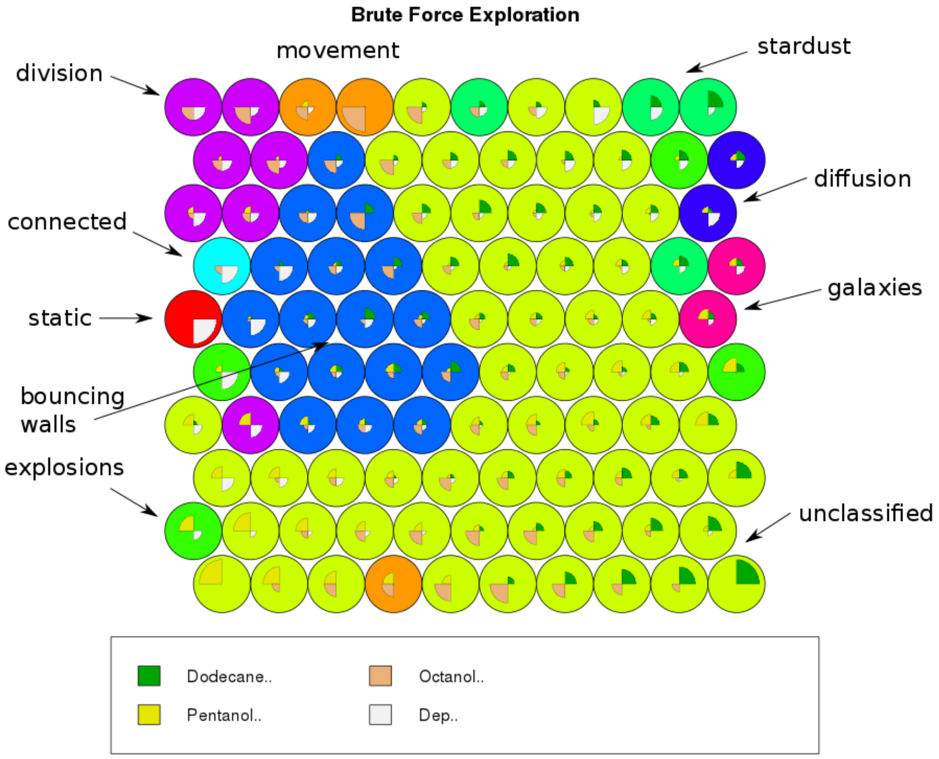
Supplementary Figure 5: Tubing output setup. Plastic syringes barrels were used to guide the tubes. First the four oils, and at the end acetone and aqueous phase share the same barrel.



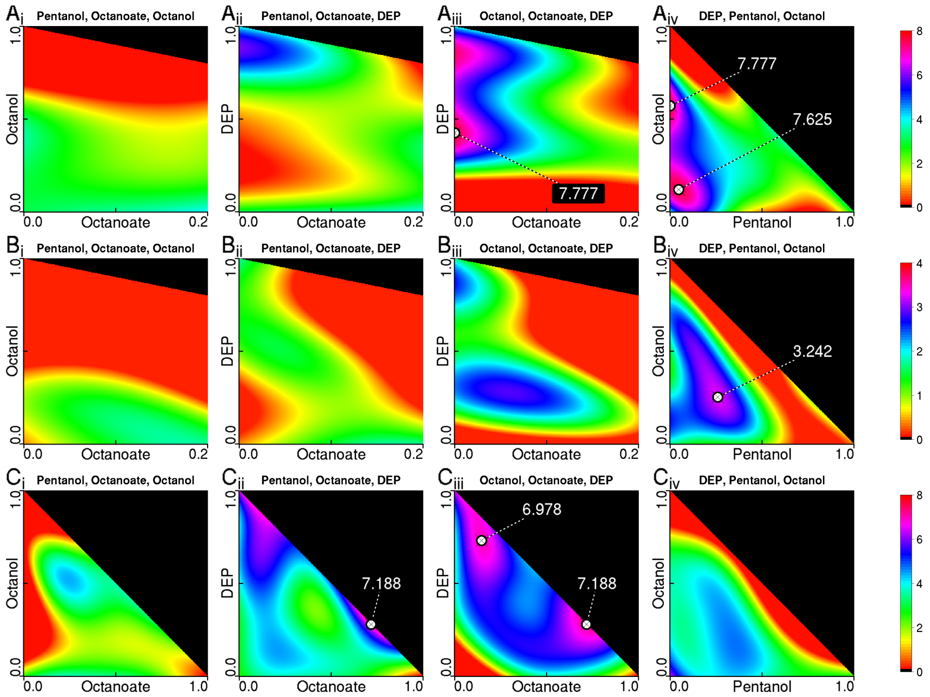
Supplementary Figure 6: Automated syringe prototype. The plunger was actuated using a 9g servo motor. Guide rods were used to avoid bending.



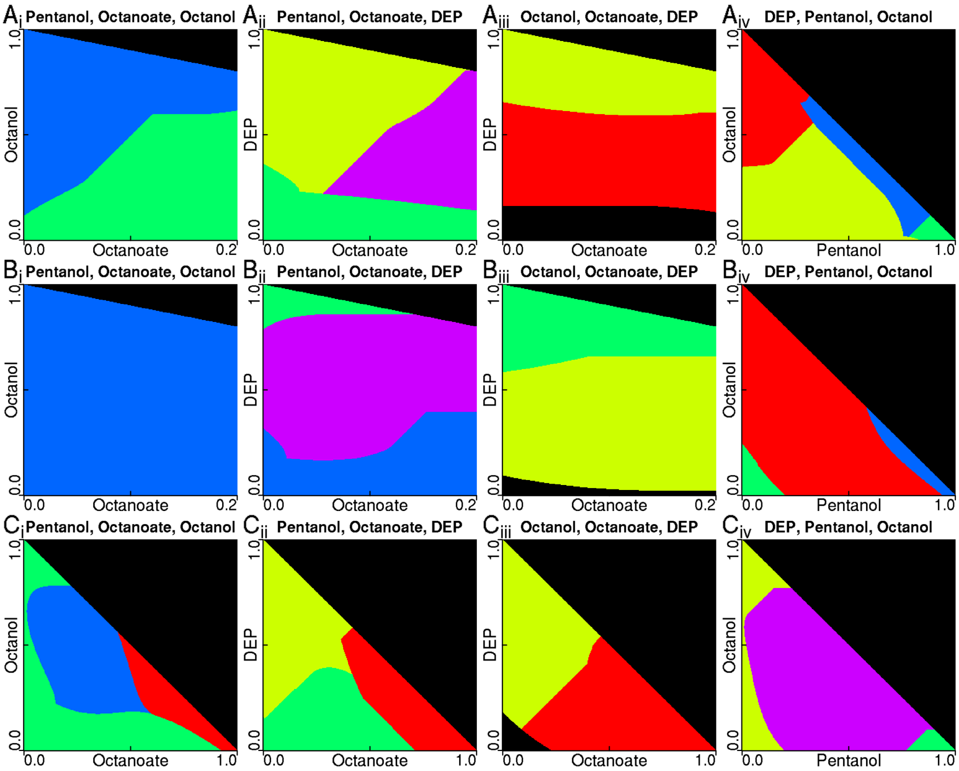
Supplementary Figure 7: Syringe attached to needle and lifting mechanism. The same lifting mechanism was used to raise and lower a plastic needle to remove liquid from the Petri dish



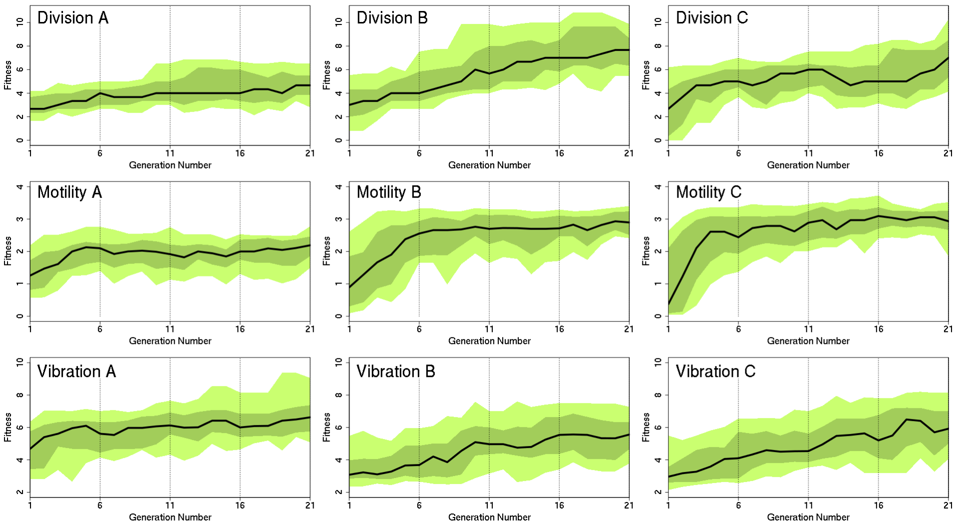
Supplementary Figure 8: To better visually represent the behavioural space exhibited during the lattice-search, a self-organizing map analysis was applied to the resultant data. Readers unfamiliar with the technique are referred to [17] and [18] as introductory texts. Behaviours were manually assigned, based on visual assessment by one of the researchers. Each circle in Supplementary Figure 8 represents a "node": The fundamental unit of output from the SOM. Chemical composition varies across both the X and Y axes in a spatially significant, but visually non-obvious pattern. Each behaviour is assigned an individual colour and it can be seen that behaviour cluster together in space, and therefore in composition. Each cluster of behaviours therefore represents a phenotypic "island" within the composition/behaviour mapping.



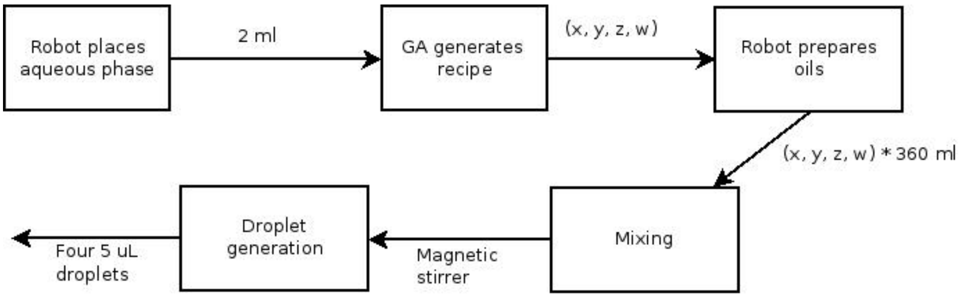
Supplementary Figure 9: Fitness landscapes for the environments division (**Ai-iv**), motility (**Bi-iv**) and vibration (**Ci-iv**. In each plot, three substances are displayed as the plot title. The fourth substance can therefore be assumed to be held at a constant zero. The projection used is equivalent to the ternary plots shown in the main text. Each axis shows the proportion of a substance (indicated). The proportion of the third substance (**Z**) is calculated as $Z = 1 - X - Y$. The numerical indications show the location of fitness peaks. In division and vibration two major fitness peaks were discovered. Minor fitness peaks are not indicated. The scale bar corresponds to the fitness functions for each environment and are not comparable between environments.



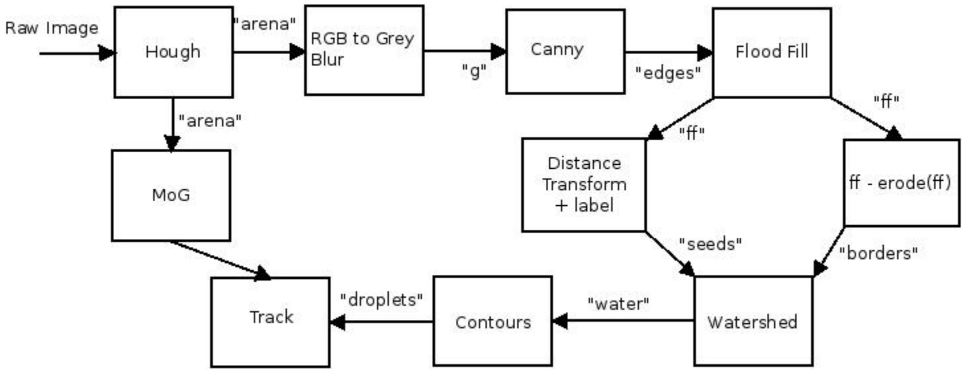
Supplementary Figure 10: Fitness island map derived from the kernel analysis for the fitness landscapes division (**Ai-iv**), motility (**Bi-iv**) and vibration (**Ci-iv**). Each colour (Red, yellow, green, blue, purple) corresponds to a stable fitness island (colours stated in sorted order of island-maximum from highest to lowest [i.e. red is the fittest island and purple the least fit]). Five islands were observed in each landscape, although it is unknown if further islands are present in the interior of the full simplex.



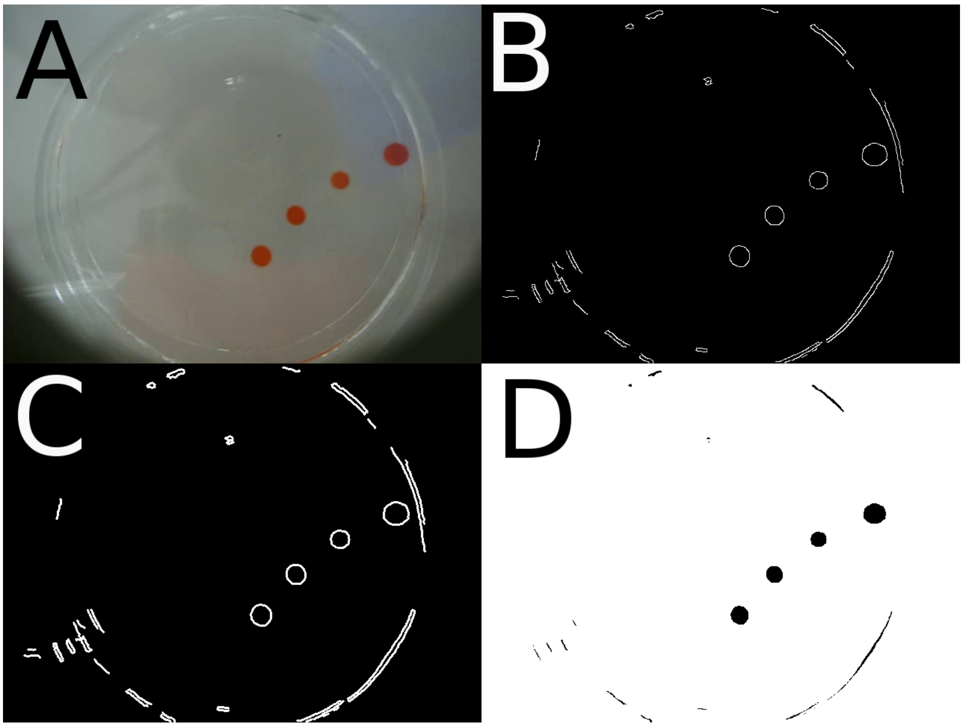
Supplementary Figure 11: Fitness progression throughout each of the nine optimization experiments, derived from three repeats for each of the three fitness landscapes. Fitness is specific to each landscape and not comparable between landscapes. The black line corresponds to the median for each generation, dark green bounds the distribution between the upper and lower 25th percentile and light green bounds between the upper and lower 10th percentile.



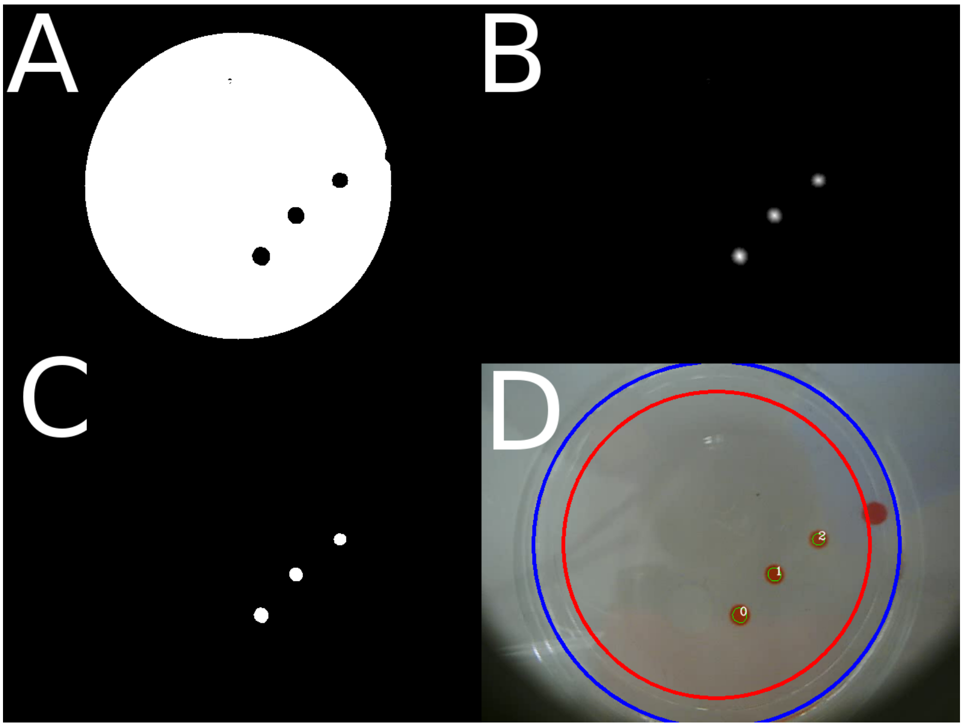
Supplementary Figure 12: Outline of the droplet generation. First the aqueous phase is placed, and then the droplets are dropped over it.



Supplementary Figure 13: Scheme outlining the pipeline of the different techniques and algorithms involved in the droplet detection.



Supplementary Figure 14: Droplet detection, image processing pipeline. Top left: Raw frame. Top Right: Canny edge detection. Bottom Left: Dilate morpho operation. Bottom Right: Flood fill operation, seed at pixel 0,0.



Supplementary Figure 15: Droplet detection, image processing pipeline. Top left: Hough transform, dish detection. Top Right: Distance transform. Bottom Left: Watershed algorithm. Bottom Right: Final result. Blue circle marks the dish detection, red circle represents the arena.

Supplementary Tables

Supplementary Table 1: Parameters used to generate all GA-derived data presented in this paper.

Parameter	Value
Generations	21
Genome length	4
Population size	25
Carry-overs	15
Per-locus mutation rate	0.3
QTL mutation (SD)	0.1
Selective pressure	1

Supplementary Table 2: ANOVA analysis of the first generation individuals in the upper half of the fitness distribution against the last generation individuals in the upper half of the fitness distribution. All fitness landscapes showed highly significant improvement in fitness from the beginning, to the conclusion of the experiment.

Fitness	F-value	p-value
Division	104.1	$< 10^{-15}$
Motility	74.9	1.7×10^{-13}
Vibration	43.6	2.7×10^{-13}

Supplementary Table 3: ANOVA analysis of the 11^{th} generation against the last generation, under the same method of analysis as in Supplementary Table 2. Division and vibration both show significant improvement in fitness during the latter half of the experiment (Under the Holm–Bonferroni multiple testing correction[20]).

Fitness	F-value	p-value
Division	5.51	0.0207
Motility	0.611	0.436
Vibration	7.08	0.00902

Supplementary Table 4: ANOVA analysis of all generations; the entire distribution for each generation was tested as a function of generation, expressed as a categorical variable. All fitness landscapes show highly significant differences in variation between generations.

Fitness	F-value	p-value
Division	407.8	$< 10^{-15}$
Motility	259.1	$< 10^{-15}$
Vibration	297	$< 10^{-15}$

Supplementary Table 5: Kendall rank-correlation test of non-linear dependence of fitness on generation, expressed as a continuous variable. All fitness landscapes show a highly significant, positive correlation between the two variables.

Fitness	τ-value	Z-value	p-value
Division	0.283	18.89	$< 10^{-15}$
Motility	0.210	14.33	$< 10^{-15}$
Vibration	0.256	17.50	$< 10^{-15}$

Oil	Density ($g\,mL^{-1}$, 20 °C)		Solubility ($g\,L^{-1}$)		Surface Tension ($mN\,M^{-1}$)		Viscosity ($mPa\,s^{-1}$)	
	Value	Ref.	Value	Ref.	Value	Ref.	Value	Ref.
1-octanol	0.824	[10]	0.46	[11]	27.1	[11]	7.288	[11]
1-pentanol	0.811	[10]	22	[11]	25.36	[11]	3.619	[11]
DEP	1.12	[10]	1.08	[11]	19.6	[14]	10.625	[15]
Octanoic Acid	0.91	[10]	0.68	[12]	27.9	[9]	5.020	[11]
Dodecane	0.78	[10]	Insol.	[12]	25.35	[13]	1.383	[11]

Supplementary Table 6: Physical properties of oils used in the experiment

Supplementary Methods

Supplementary method 1: Construction of robotic hardware

The robotic system specified in this document, *DropBot*, was built on the foundations laid by the RepRap 3D printer project[1]. This project was chosen as a starting point due to its open-source philosophy, expansive documentation and large community. These reasons were considered advantageous as they are expected to contribute to the easy replication and adoption of the *DropBot* paradim and implementation in other laboratories, as compared to a proprietary product. Early designs of *DropBot* (data not shown) aimed to use a RepRap 3d printer directly, through the replacing of the thermoplastic extruder with a liquid handling system. After early prototyping phases, it was decided that this platform was unsuitable and so a new design was formulated, re-using modular components from the RepRap project rather than its monolithic design. This section provides an overview of the methodology applied to the design and implementation of *DropBot*: Subsection 1.1 gives an introduction to the limitations encountered in the direct conversion of RepRap into a liquid-handling robot and the development of the final design used for this publication, subsection 1.3 gives an overview of the digital control systems used and subsection 1.2.3 describes the 3d-printed, servo-actuated syringe design that was developed for this robotic platform.

1.1 Robot Frame

The primary limitations encountered in early prototyping phases, which attempted the direct modification of a RepRap 3d printer into a liquid-handling robot, were the limited working area and the trans-location of the target stage, in the Y axis, rather than the manipulation apparatus. The most common design for a RepRap 3d printer has a working area of 20x20cm, which was was unsuitably small for the target experiments. The limitations imposed by the size of the experimentation apparatus, prescribed a target stage of around 50x40cm; no extant printer design, which also met the other demands, was discovered that would fulfil this criterion, at the time of project initiation. The second factor, the movement of stage along the Y-axis, rendered the design unsuitable for a liquid handling robot, as it introduced extreme turbulence to the liquid-phase target chemistry; essentially, the stage was behaving similarly to a shaker plate. In addition, the design involved the majority of the Y-actuation mechanism being physically located underneath the stage, occupying space that would optimally be allocated to the inclusion of a camera, for the analysis of the experiments, with a clean field of view.

To correct these deficiencies, it was decided that the manipulation apparatus should be located on an overhead XY-axis, above a static, glass staging area, with a camera located on a separate XY-axis being located underneath this stage, viewing the experiments through the glass from the bottom. Whilst this restricted the robot to working with glass apparatus, it was decided that, since this would be the normal mode of operation in any case, this limitation would not impact the real-world performance.

1.1.1 Mechanical design

Supplementary Figure 1 shows the final design, used for all methods in this publication. The primary modules of the design are:

1. The staging area, on which the experimentation apparatus is placed.

2. The X-axis, which actuates motion along the long edge of the robot.

3. The Y-axis, which actuates motion along the short edge of the robot.

4. The manipulation apparatus (the mobile carriage), which performs formulation mixing, aqueous-phase handling, droplet placement and cleaning.

5. The fluid-handling platform, which handles the introduction of liquids to the main stage of the robot for further manipulation.

1.1.2 Staging area

The staging area consisted of a large glass plate, fixed to the robot frame. This area was defined by being that space over which the X-Y carriage could move and hence all apparatus that needed manipulation by the carriage was placed on the stage. At the centre of the stage was a 96-well plate, in which fluids were placed for formulation mixing, prior to droplet placement. There was a magnetic stirrer underneath the well-plate, to actuate miniature magnetic stirrer bars inside each well. Also atop the stage were two petri dishes, one was used to carry out the experiments and the other was used to collect waste after experiments were concluded. Manual intervention was required after a series of 48, or 96, experiments (dependant on whether experiments ran overnight) to clean the well-plate and waste petri dish.

1.1.3 X-axis

Supplementary Figure 2 shows a 3D-rendering of the X-axis translation mechanism. The same design was mirrored on both sides of the frame. This mechanism is static in relation to the staging area and frame of the robot

1.1.4 Y-axis

Supplementary Figure 3 shows a 3d-rendering of the Y-axis translation mechanism. This mechanism runs along the X-axis via the X-axis belt, rod and linear bearing system. The two round steel bars seen in Supplementary Figure 3 were inserted into the complimentary holes seen in Supplementary Figure 2. A single motor was mounted on one of the X-axis carriages to actuate the central belt and provide linear motion along the Y-axis.

1.1.5 Mobile carriage

Supplementary Figure 4 shows a 3d-rendering of the X-Y carriage. This components runs along the Y-axis via the Y-axis belt, rod and linear bearing system. The "ridge" running from left to right along the Y-axis runner served as an attachment point for the circuitry (subsection 1.3). The design of the syringe actuation mechanism is detailed in subsection 1.2.3.

1.1.6 Fluid platform

The fluid-handling platform is a simple frame, constructed from aluminium strut profile and the 3d-printed assembly pieces used in the main robotic frame. Sheet-polycarbonate was attached to the strut-frame to support the weight of the pumps. The frame was designed so that reagents could be places underneath the pumps; This design was chosen to minimize the effect of any chemical spills by prevented contact with the electronics. The design of the pumps themselves is detailed in subsection 1.2.1.

1.2 Liquid Handling

Both servo-actuated syringes and syringe pumps were used to transport liquid phases. The servo-syringes were more precise but were unable to carry as much volume as the syringe pumps. Servo-syringes were used to mix the liquid in the well-plates and to carry small volumes of liquid into the petri dish via droplet formation. Syringe pumps were used to transport liquids from source reagent bottles to components on the experiment stage.

1.2.1 Pumps

In the course of extraneous and prior research, the authors had accumulated a number of defective commercial syringe pumps, rendered inoperative through faults in their logic boards. Its original electronics were removed and the motors were connected to custom components (see subsection 1.3). A total of seven such pumps were used by the robot, mounted on a single platform (see subsection 1.1.6). Each pump was equipped with a three-way valve: Allowing the syringe to be connected through one port to either of the other two. One of these port was designated as an input port and the other as an output port. Three of the pumps were equipped with 5ml syringes: One was used for introducing the aqueous phase, and the other two for the introducing and removing solvents as part of the cleaning procedure. All pumps except one were connected via the input port to a reagent bottle and via the output port to the X-Y carriage; the sole except was the syringe pump used for removing the acetone from the petri dish as part of the cleaning cycle. The remaining four pumps were equipped with 1ml syringes; these were used to introduce organic phases.

The tubing set-up at the X-Y carriage can be seen in Supplementary Figure 5. The syringe barrels were used as guides, to maintain accurate positioning of the tubing above the working area. Acetone and aqueous phase shared the same barrel, with two independent tubes being fixed inside by hot glue. The support structure was 3d-printed in PLA and was attached to the carriage with brass screws.

1.2.2 Mixing stage

The robot used a standard 96-well plate. Each well had its own miniature magnetic stirrer bar and the entire plate was mounted over a stirrer plate. This design was chosen so that multiple experiments could be concluded before manual intervention was necessary to clean the mixing area.

1.2.3 Carriage-mounted syringes

As shown in Supplementary Figure 6, an automatic syringe was designed to be used with the X-Y carriage. The casing and structure was 3d-printed from PLA. Independent crank mechanisms were used to actuate the plunger of the syringe and to lower and raise the syringe. These consisted of the default servo motor-arm and a 3d-printed piece, joined via steel pins. The crank was aligned with the centre of the syringe itself to avoid unwanted lateral torque. Small-diameter steel rods and teflon linear bearings were used to mitigate rotation away from and towards the servo.

The syringes were fitted with a metal needle-tip, as shown in Supplementary Figure 7. The mechanism to raise and lower the syringe, via a crank, can also be seen in this Supplementary Figure. On the right of Supplementary Figure 7, the same mechanism can be seen to be actuating an unactuated syringe. This syringe has had its plunger removed and is instead fitted with a piece of plastic tubing, which connects it to a syringe pumps.

This syringe was used to remove solvents from the Petri dish during a cleaning cycle. For this reason, the needle taper tip was cut, to effect a larger absorption diameter. Whilst *DropBot* only used one of each syringe type, it would be possible to expand the robot by adding additional syringes to the carriage.

1.3 Electronics

The electronics can be divided into three groups. One group controls the movement of the robot along the X and Y axes,another group controls the servo motors which actuate the syringes, and a third group control the pumps.

1.3.1 Axis control

The movement along the X and Y axes was performed using a stripped down version of the electronics used by a RepRap 3D printer, where only the parts required by the X and Y movement were kept: an Arduino Mega[2] board and "A4988 Stepper Motor Driver Carrier". A PCB shield was designed, which interfaced the Arduino board to the motor drivers and provided connectors for the motors to the drivers. In addition, connections were provided for two end-stops, each assigned to a particular axis, for homing purposes. Each stepper driver had 16 pins, but only 3 of them were needed to control the motor: enable, step and direction, with the other pins being connected to high (5V), low (GND), 12V or to the motor itself. The board therefore used only eight outputs (the end-stops used two outputs) from the Arduino directly and interfaced the remainder of digital connectors, and a power supply, to a header output for the servo daughter-board.

1.3.2 Carriage control

The header input to the servo daughter-board consisted of twenty pins, connected to the Arduino via a ribbon cable to the bypass on the XY driver board. The servo board itself was mounted on the X-Y carriage in order to provide both power and control to the syringe-actuation servo motors. Each servo motor took power from a common supply and were connected to a common ground, with the only individual connection being a single PWM data pin to the Arduino. The servo daughter-board was therefore able to service 20 unique servo motors.

1.3.3 Pump control

The pumps mentioned in subsection 1.2.1 were faulty only in the function of their logic boards; the mechanical components and motors were fully operational. As motion was provided by two stepper motors, from the NEMA family, these were therefore compatible with the shield developed for the

X-Y axis control (Subsection 1.3.1). It was therefore possible to re-use the mechanical components of these syringe pumps by the replacement of their electronics with custom PCBs. A second Arduino was assigned entirely to pump control.

1.4 Bill of Materials

- The frame was built using Bosch-Wrexroth 20x20mm aluminium strut profile, fastened together using custom, 3d-printed PLA[1] pieces.
- The motors used were NEMA[2] 14 for the Y-axis and two NEMA 17s for the X-axis.
- The linear motion mechanics were derived from the RepRap printer, using a belt (Timing belt T2.5x6mm) and pulley (T2.5 pulley, 5mm bore) system connected to the motors
- Round-profile hardened-steel bar (8h7, chrome plated) and round-profile linear bearings (LME8UU) were used to achieve smooth linear motion.
- The syringe pumps modified and used were "TriContinent C-Series"
- "IDEX Health Science PEEK 1/8"" tubing was used to connect those pumps used introduce or remove cleaning solvent to/from the petri dish arena.
- "IDEX Health Science FEP Ora 1/16 x 0.20"" was used to carry organic phases from reagent bottles to syringe pumps and from syringe pumps to the X-Y Carriage.
- The syringe used to direct the tubing from the syringe pumps at the carriage the "1 ml NORM-JECT".
- These syringes were fitted with "Weller KDS16TN25 Needle Taper Tip 16G".
- The syringe used in the servo-actuated syringes was a "Hamilton 710 LT 100 μL".
- The needles used were "Weller KDS2012P Dispensing Needle GA20 ID 0.66 MM".
- The syringe was raised and lowered by a "New Power XL-3.7" servo-motor.

[1] Polylactic acid.

[2] National Electrical Manufacters Association. `http://www.nema.org`

- The plunger was actuated with a “9g servo motor”. In this specific case, a “TowerPro Micro Servo 9g SG90”.

- The motor arm used was the default arm, which shipped with the servo motor.

- The well plate used was “Nunc U96 PP 0.5 ml”.

- Inside every well, a “Magnetic stir bar micro PTFE 6 mm x 3 mm” was used to provide liquid turbulence.

- Below the well plate, was a single “Variomag Compact” stirrer plate, used to turn the stirrer bars.

Supplementary method 2: Software implementation

A hierarchical/deliberative paradigm was followed when designing the control software. Software components resident on the host computer was programmed entirely in Python, whilst software components on the Arduinos (referred to henceforth as the firmware) were programmed in C++. Software was divided into three conceptual modules: *planning*, *acting*, and *sensing*.

The artificial intelligence (AI) component is responsible for coordinating the overarching experimental plan, selecting droplet formulations based on prior (if any) data and passing these formulae on to the next component. Thus the AI acts as globally, with overview of an entire experimental series, and the RC, FW and CV components have purview only of single experiments.

The robotics controller (RC) takes these formulae as input and, acting as an interface between the AI and the physical robot, transforms these numerical recipes into a G-code representation. G-code is a standardized scheduling language[3] used in the automation industry to plan mechanical operations (see subsection 2.2 for more details).

The firmware (FW) consists of software resident on the Arduino board. It converts the symbolic G-code representation into a series of experimental operations, instantiated as a sequence of analogue and digital signals sent to the mechanical parts through the Arduino boards.

The computer vision (CV) component interfaces with the camera and converts raw image data, through an algorithmic image-processing pipeline, into quantitative-numerical fitness data, which is then returned to the AI.

A multiprocessing software architecture was chosen to make maximum usage of time and resources, and to ease development by modularizing the software. Software interprocess communication was used between modules running on the host computer, whilst USB-serial protocols were used to communicate between the software and firmware.

2.1 Robot Controller

The robot controller functions to translate experimental procedures form a high-level description into a G-code representation. The G-code can be considered as an intermediate layer between the description of a recipe and the digital actuation, effected by the electronics. By using this translation pipeline, a number of expert-written modules could be leveraged, significantly reducing development costs and time. The core component of the RC is the PrintRun[4] library, developed for controlling RepRap printers (via G-code) and and which constitutes the most popular choice of library for this purpose. As this central library is written in the Python programming language[5], this language was used for the entire development of this layer. PrintRun communicates with the Arduino-based Firmware via a USB emulated-serial connection, facilitated by the pySerial library on the host computer[6].

The RC's interface to exterior code was encapsulated within a library called "RobotCtl". This library contains functions which relate to the fixed set of modular lab operations (e.g. "form droplet" or "clean petri dish") required to perform all experiments. The interface functioned to convert a sequence of these operations into a sequence of G-code instructions. The robot could move the syringes to any position around X or Y with a precision of 0.1 mm and this precision was accounted for in the conversion. Since there were multiple components on the X-Y carriage, the relative distance between these components was hard coded and used to modify the final X-Y position when one specific apparatus was selected. In contrast to the operational mode of a 3d-printer, which produces the entire batch of G-code from a 3d model in one run, the RC runs in online mode, compiling operations into G-code as they are received. This mode of operation is a necessity, due to the iterative nature of the procedures, whereby future operations are unknown until data from present operations have been processed. The G-code produced in real-time is communicated to the robot via the PrintRun core.

2.2 Firmware

The firmware layer is directly responsible for the actuation of mechanical parts. The firmware was written specifically for the target Arduino boards, hence the Arduino development environment was used for this layer. The native language of this environment is C++; therefore, this was the language used to develop the firmware layer.

There were two separate Arduino boards, one to control the X-Y carriage (subsection 1.1.5) and one to control the fluid platform (subsection 1.1.6) As with the RC layer, the carriage firmware was built on top of extant 3d-printer modules. In this case, the firmware core used was the Sprinter package[7], commonly used with Reprap 3D printers. Functionality with regards to Z-axis movement, temperature control and extrusion of thermoplastic was removed from the code-base. In its stead, functionality was added for control of the syringes (See subsection 1.2.3), via the servo library provided by Arduino[8]. The modified Sprinter firmware was therefore capability of X-Y motion, able to actuate all servo-motors and synchronize these actuations with the RC component, via the receipt and parsing of G-code instructions.

The fluid platform firmware was developed *de novo*. Each pump contained two components: A three-way valve and lead-screw actuated plunger. Both of these components sourced motive force from a NEMA stepper motor. These motors were controlled, as with other motors on the robot, via an "A4988 Stepper Motor Driver". A G-code interface was added, with new symbols for pump movement.

Very little sanity checking was performed and so the RC was relied upon for coherent operation.

2.3 Artificial Intelligence

For each behaviour to be tested, three Genetic Algorithm (GA) runs were used. Each GA run performed 21 generations, with a fixed-population size of 25 individuals and 15 individuals being propagated from the previous generation, for a total of 225 recipes. Each recipe was repeated three times, and the minimum between the mean and the average of these 3 experiments was returned. Each experiment generated four droplets.

A complete test ran the GA 3 times, therefore in total it generated 675 recipes. Each recipe was repeated 3 times, for a total of 2025 experiments. Each experiment generated 4 droplets, for a total of 8100 droplets.

Individuals were fixed-length genomes of 4 floating-point numbers (i.e. Quantitative trait loci). The GA used a per-locus probability of mutation (resulting in a poisson-distributed number of mutations per individual). For each locus selected for mutation, a normal-distributed noise function, with a mean of 0 and an SD of 0.1 was additively applied. Each child was always the product of a single crossover recombination between two distinct parents, with the crossover being uniformly distributed along the genome and the same genetic location being used for each parent. Individuals were birthed with parents being selected, without replacement, from the extant pool with probability proportional to the fitness (to the power of some parameter). After birthing and fitness measurement, the population was culled to a fixed size, with individuals being chosen for death with probability inversely proportional to the fitness (to the power of some selective pressure parameter).

Supplementary method 3: Construction of fitness landscapes

The fitness landscapes seen in Figure 6 in the main text were produced through a multi-step analytical pipeline. The results from the GA were first collated and passed through a radial basis function kernel ridge regression to produce a model. This model was then queried through a grid search along each face of the parameter-space simplex to estimate the fitness at each location.

3.1 Kernel Ridge Regression

General Linear Regression (GLR) performs model fitting by minimizing the sum-of-squares error over a space of linear coefficients, in an equation of the form

$$\widehat{y}_i = \widehat{\boldsymbol{\phi}}^\top \boldsymbol{x}_i, \tag{1}$$

with the sum-of-squares error being given for N points as

$$\left(\widehat{\boldsymbol{y}} - \boldsymbol{y}\right)^\top \left(\widehat{\boldsymbol{y}} - \boldsymbol{y}\right). \tag{2}$$

Ridge-regression, also known as Tikhonov regularisation[16], is a commonly used method of regularization of ill-posed problems. This form of regression augments the minimization with a weighted penalty on the size of the coefficient vector $\widehat{\boldsymbol{\phi}}$:

$$\widehat{\boldsymbol{\phi}} = \arg\min_{\widehat{\boldsymbol{\phi}}} \left(\widehat{\boldsymbol{y}} - \boldsymbol{y}\right)^\top \left(\widehat{\boldsymbol{y}} - \boldsymbol{y}\right) + \lambda \widehat{\boldsymbol{\phi}}^\top \widehat{\boldsymbol{\phi}} \tag{3}$$

Kernel ridge-regression is a re-formulation of ridge-regression, used in situation where the number of dimensions exceeds the number of data-points. Equation 1 is substituted by

$$\widehat{y}_i = \boldsymbol{\theta}^\top \boldsymbol{X} \boldsymbol{x}_i. \tag{4}$$

With equation 3 taking the equivalent substitution:

$$\widehat{\boldsymbol{\phi}} = \arg\min_{\widehat{\boldsymbol{\phi}}} \left(\widehat{\boldsymbol{y}} - \boldsymbol{y}\right)^\top \left(\widehat{\boldsymbol{y}} - \boldsymbol{y}\right) + \lambda \widehat{\boldsymbol{\theta}}^\top \widehat{\boldsymbol{\theta}} \tag{5}$$

An important property of this reformulation is that it requires the calculation of a dot-product between each input vector and every other. These dot-products can be collated in the Gram matrix:

$$\mathbf{G} = \boldsymbol{X}\boldsymbol{X}^\top \tag{6}$$

However, the dot-product does not need to be computed in input space. Higher predictivity can often be obtained by transforming the input vectors into a higher-dimensional space, known as "feature space":

$$\boldsymbol{z}_i = f(\boldsymbol{x}_i) \tag{7}$$

the Gram matrix $\boldsymbol{G}$ is then replaced by the kernel matrix $\boldsymbol{K}$, whose entries are given by

$$K_{ij} = \boldsymbol{z}_i^\top \boldsymbol{z}_j. \tag{8}$$

Rather than calculating the dot products through an explicit mapping from input space into feature space, it is usually quicker to use an implicit feature-space inner product g, calculated directly on the input vectors:

$$K_{ij} = g(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{9}$$

The function g is then known as the "kernel function" and its use, with associated improvements in computational speed, is often referred to as the "kernel trick".

3.2 Radial Basis Function

The kernel function that we apply to extend the sampled points into a smooth fitness landscape is the Gaussian Kernel. This kernel function falls into the broader category of Radial Basis Functions (RBFs), whose members are defined by being real-valued function that depend only on the distance, in input space, between the input vectors. Distance is here defined between points i and j as

$$r_{ij} = \left\| \mathbf{x}_i - \mathbf{x}_j \right\|. \tag{10}$$

The Gaussian RBF then defines the kernel function between two points i and j as

$$K_{ij} = e^{-\frac{r_{ij}}{2\sigma^2}}. \tag{11}$$

Where σ is a problem-specific parameter, roughly equivalent to the standard deviation of the Gaussian probability distribution. One peculiar property of the Gaussian kernel function, not shared by other RBFs, is its implicit mapping of the input vectors into an infinite dimensioned feature space. This property arises from the expansion of the exponential term, here demonstrated for a single-dimension input vector:

$$K_{ij} = e^{-\frac{\left(x_i - x_j\right)^2}{2\sigma^2}} \tag{12}$$

$$= e^{-\frac{x_i^2}{2\sigma^2}} e^{-\frac{x_j^2}{2\sigma^2}} \sum_{k=0}^{\infty} \left[\frac{2^k x_i^k x_j^k}{2\sigma^2 k!} \right] \tag{13}$$

As a result, whereas other kernel functions can be used to derive a set of feature space coefficients, the Gaussian RBF requires that the kernel function be evaluated for every N training vector with a new vector, for a prediction to be made for that vector:

$$\widehat{y}_k = \sum_{i=0}^{N} K_{ik}\theta i \tag{14}$$

3.3 Complete Landscapes

Three reduced fitness landscapes are shown in the main text in figure 6. Each of these subfigures shows a three-dimensional “facet” of the four-dimensional space of oil composition. Each facet was chosen so that the global maximum, for each environment, would be shown, in each case. However, as each facet is derived by holding one of the oil proportions at zero, there are three other facets per environment. All facets for the three environments are therefore shown below.

Supplementary Figure 9 shows four fitness landscapes per environment, displaying a greater proportion of the analysis than that found in the main-text. As “true” representation of the data consists of a solid four-dimensional simplex, graphical representation is inherently difficult. The authors therefore opted to display on the faces of this simplex. This was considered a reasonable approximation as, for each environment, the global maximum was found on one of these faces and not internally. Indeed, for division and vibration environments, the larger of the two major fitness peaks was found on an edge between two faces (a two-substance composition). These maxima can therefore be seen, on two faces, for division in **Aiii** and **Aiv** (7.777) and for vibration in **Cii** and **Ciii** (7.188). Numerous sub-optimal local maxima are also discovered by the analysis in all environments and are explored in subsection 3.4.

3.4 Catchment

In an evolutionary fitness landscape, multi-modality corresponds to the concept of *fitness islands* [19]. Such a feature is defined as being that volume surrounding a local (or the global) maximum, such that for any point within that volume, consistent, upward progress along the gradient will converge at that specific maximum.

To analyse the number and boundaries of the fitness landscapes that underly the experimental evolution component of this manuscript, a discovery algorithm was run on the fitness hyperplanes derived through the kernel modelling. The hyperplanes were represented as 4 301 × 301 quantized lattices, as plotted in figure 4 in the main text. For each unique maximum, an active set was maintained, starting with a single location at that maximum.

For each location in the active-set, all eight surrounding, quantized locations were tested, such that for each location tested, the 8 locations around *that* location were tested, to discover which of them corresponded to the neighbouring maximum. For each location whose neighbouring maximum was the current location from the active-set, that location was marked as belonging to the current fitness island and then added to the active-set for analysis of its own neighbourhood. It was simple to allow this search to extend over the boundaries of one hyperplane to another hyperplane, where the locations were equivalent (i.e. where one of the components was 0).

The results of this algorithm are presented in Supplementary Figure 10. For each of the three landscapes, exactly five fitness islands were observed, although there may be fitness islands, in the interior of the fitness space, not revealed by the search across the exterior hyperplanes. There is insufficient data to specify whether the consistency of the occurrence of five islands per landscape is a product of the dimensionality or a coincidental artefact.

Supplementary References

[1] Jones, R., Haufe, P., Sells, E., Iravani, P., Olliver, V., Palmer, C., and Bowyer, A.,: RepRap - The Replicating Rapid Prototyper, Robotica **(2011)** volume 29, pp. 177–191. Cambridge University Press.

[2] http://www.arduino.cc/

[3] Multiple authors **(1982)**. *Numerical Control of Machines* ISO 6983-1:1982

[4] Yanev, K. *et al* **(2011)** https://github.com/kliment/Printrun

[5] G. van Rossum and F.L. Drake (eds), Python Reference Manual, PythonLabs, Virginia, USA, 2001. Available at http://www.python.org

[6] Liechti, C. **(2001)** http://pyserial.sourceforge.net/

[7] Yanev, K. *et al* **(2010)** https://github.com/kliment/Sprinter

[8] http://playground.arduino.cc/ComponentLib/Servo

[9] http://www.dynesonline.com/visc_table.html

[10] http://sigmaaldrich.com

[11] Haynes, W. M. **(2013)**. CRC Handbook of Chemistry and Physics, 94th Edition. CRC Press, 94th edition,

[12] Budavari, S., Smith, A., Heckelman, P., Kinneary, J. and O'Neill, M. **(1996)** The Merck Index. Merck .

[13] Jasper, J. J., and Kring, E. V. **(1955)** *The Isobaric Surface Tensions and Thermodynamic Properties of the Surfaces of a Series of n-Alkanes, C5 to C18, 1-Alkenes, C6 to C16, and of n-Decylcyclopentane, n-Decylcyclohexane and n-Dcylbenzene.* Journal of Physical Chemistry, 2142(3):3–5, **1955**.

[14] Thomsen, M., Carlsen, L. and Hvidt, S. **(2001)** *Solubilities and Surface Activities of Phthalates Investigated by Surface Tension Measurements.* Experimental Toxicology and Chemistry, 20(1):127–132.

[15] Rostamkolahi, A. M., Rostami, A. A., Koohyar, F., and Kiani, F. **(2013)** *Thermodynamic prop + vinyl acetate, diethyl phthalate + vinyl acetate or bromocyclohexane, and dibutyl phthalate + vinyl acetate or 1,2-dichlorobenzene at 298.15–308.15 K.* Chemical Papers, 67(11):1433–1441

[16] Tikhonov, A. N. **(1963)**. *Solution of incorrectly formulated problems and the regularization method.* Doklady Akademii Nauk SSSR 151: 501–504.. Translated in Soviet Mathematics 4: 1035–1038.

[17] Kohonen, T. **(1982)** *Self-Organized Formation of Topologically Correct Feature Maps* Biological Cybernetics 43 (1): 59–69.

[18] Haykin, S. **(1999)**. *9. Self-organizing maps.* Neural networks - A comprehensive foundation (2nd ed.). Prentice-Hall.

[19] Jain, K. and Krug, J. **(2007)** *Deterministic and Stochastic Regimes of Asexual Evolution on Rugged Fitness Landscapes.* Genetics 175: 1275–1288

[20] Holm, S. **(1979)**. *A simple sequentially rejective multiple test procedure.* Scandinavian Journal of Statistics 6 (2): 65–70.

[21] Kendall, M. **(1938)**. *A New Measure of Rank Correlation.* Biometrika 30 (1–2): 81–89.