Cloud-Enabled, Reactive Liquid Handling Robot

A DISSERTATION PRESENTED
BY
FARZAD NEJATIMOHARRAMI
TO
THE DEPARTMENT OF COMPUTER SCIENCE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

SUPERVISOR
PROF. KASPER STØY

IT UNIVERSITY OF COPENHAGEN
COPENHAGEN, DENMARK

AUGUST 2017

Thesis advisor: Prof. Kasper Stoy Farzad Nejatimoharrami

Cloud-Enabled, Reactive Liquid Handling Robot

Abstract

We introduce a robotic system developed to perform artificial chemical life experiments based on droplet systems. These experiments are particularly well suited for automation because they often stretch over long periods of time, possibly hours, and often require that the human takes action in response to observed events such as changes in droplet velocity, size, count, shape, or clustering or declustering of multiple droplets.

We describe a robotic system designed to automate this type of artificial chemical life experiment. We verify the design by successfully applying it to three different experiments: i) automating an experiment whose purpose it is to measure the movement response of a droplet as a function of distance to a reagent, ii) aspirating a moving droplet when the droplet speed goes below a specified threshold, and iii) detecting clustering of droplets and dispensing a salt droplet at a specific distance to the cluster. We demonstrate how our robot cannot only automate these experiments, collect data, and interact with the experiments in a closed feedback loop, but also enable chemists to perform formerly infeasible experiments.

Another aspect of our work is that we designed, implemented, and deployed a cloud-based user interface for a distributed system of liquid handling robots. The user interface is intuitive, as well as being easily extensible for new experimental protocols in biological and chemical laboratories. The user interface developed can control the robot on multiple platforms and devices. Remote control of the robotic

system enhances effective use of time and usability. The open source software for the user interface allows users to customize the experiment protocols based on their requirements. Furthermore, as multiple users use our robotic platform, quality, reusability, deployability, and maintainability of software become central and crucial for efficient use of the robot. Our cloud solution enables users to share experiment code, and reuse already developed protocols for experiments. Therefore, the robotic system can be used simultaneously by multiple users, and users can work on the same experiment collaboratively. Furthermore, a user is able to run experiments on several robotic systems at a time and hence increase throughput through parallelism.

Performing reactive experiments in artificial chemical life research is made possible with this robotic platform. In addition, a cloud-based user interface provides new opportunities by enabling real time control of the robot on multiple platforms, making collaborative work on the robot possible as well as parallelizing experiments.

Thesis advisor: Prof. Kasper Stoy Farzad Nejatimoharrami

Cloud-Baseret Reaktiv Væskehåndteringsrobot

Resumé

Vi har udviklet et robot system til at udføre kemiske forsøg indenfor kunstigt liv baseret på droplet systemer. Specifikt er robotten fokuseret på forsøg med droplets, som er i stand til at bevæge sig selv pga. deres kemiske sammensætning. Denne slags forsøg er oplagte at automatisere, fordi de typisk tager lang tid, endda timer, og ofte kræver at mennesket skal gøre noget som reaktion på observerebare hændelser som f.eks. ændring i dråbe størrelse, antal, form, eller gruppe dannelse eller gruppe opbrud i multi-droplet systemer.

Vi beskriver et robot system, der er designet til at automatisere denne slags kunstig liv eksperimenter. Vi har verificeret vores robot ved succesfuldt at anvende den i tre forskellige eksperimenter: 1) automatisering af et eksperiment hvis formål det er at måle, hvordan en droplet bevæger sig som funktion af afstanden til en reagens, 2) at aspirerer en dråbe i bevægelse når dens hastighed når under en specificeret grænseværdi, og 3) detektere gruppedannelse af droplets og pipettere en salt droplet i en specificeret afstand fra gruppen. Vi demonstrere ikke kun hvordan robotten kan automatisere disse eksperiments, indsamle data, og interagere med eksperimenter i en lukket kontrol sløjfe, men også hvordan kemikere kan lave eksperimenter, som ikke før var mulige.

Et andet aspekt af vores arbejde er at vi designede, implementerede, og installerede en cloud-baseret brugergrænseflade til et distribueret system af væske håndteringsrobotter. Brugergrænsefladen er intuitiv og kan nemt udvides til at håndtere

nye eksperimentprotokoller i biologiske og kemiske laboratorier. Brugergrænsefladen kan kontrollere robotten fra mange forskellige typer af digitale enheder. Fjernbetjening af robot systemet gør det mere tidseffektivt at bruge samt forbedre dets brugbarhed. Open-source softwaren som robotten er baseret på gør det muligt for tekniske brugere at tilpasse systemet til deres behov. Endvidere, givet at flere brugere bruger vores robot platform er kvalitet, genbrug, og vedligeholdelse af software centralt for effektivt brug af robotten. Vores cloud løsning gør brugere i stand til at dele kode og genbruge allerede eksisterende kode. Det betyder at robot systemet kan blive brugt samtidigt af flere brugere. Endvidere kan brugerne samarbejde om at udvikle eksperimenter. Endeligt, er en bruger i stand til at kører et eksperiment på flere robot systemer på samme tid og dermed øge antallet af eksperimenter der kan gennemføres igennem parallelism.

Vores arbejde har gjort det muligt at udføre reaktive eksperiment indenfor kunstig kemisk liv forskningen. Udover dette giver den cloud-baserede brugergrænseflade nye mulighed ved at gøre det muligt at arbejde på og udføre eksperimenter parallelt.

Contents

Listing of figures

To Atefeh, My lovely wife.

Acknowledgments

The years of my PhD were a long fun journey. First and foremost, I would like to express my special appreciation and thanks to my supervisor, Kasper Støy. His good judgement has been always helpful and inspiring to me. I am so happy (and so lucky) of having had this experience. Owing to you, the take home of my PhD years transcends academic progress, and will persists for the years to come. Thank you for giving me the opportunity to work on this research!

I am indebted to Andres Faina, who has been like my older brother. I have had the opportunity to learn a lot from him, and he has been supportive in every possible way. Working with him the late hours on this project have been fun instead of being tiring.

I would like to thank all my colleagues in the Robotics, Evolution, and Arts (REAL) LAB for their support and valuable input at various stages of my research. Particularily, Frank Veenstra, Laura Beloff, Jonas Jørgensen, Sebastian Rici, with whom I have a lot of unforgettable memories.

The work presented in this dissertation was supported by the E.U. Future and Emergent Technologies who supported this work through EVOBLISS grant no. 611640, and also IT University of Copenhagen. I would like to also thank the members of the EVOBLISS consortium. In particular, Jørn Lambertsen who was involved in building EvoBot, Jitka Cejkova, Martin Hanczyc, and Silvia Holler who helped us perform artificial chemical life experiments, some of which are featured in this work. In addition, I would like to thank Florian Blauert, Pavlina Theodosiou,

and Silvia Holler who provided feedback on the interface.

I am also Thankful for the support I received from Prof. Mark Chignell during my research stay in his Lab, Interactive Media Lab, at University of Toronto, Canada, for six months. Mark Chignell, Olivier St-Cyr, and Andrea Jovanovic helped me learn a lot about user interface design. I had the opportunity to learn a lot form very distinguished researchers in the lab including, Jenna Blumenthal, Andrea Wilkinson. In addition, special thanks to Naveen Venayak, and Brian Nguyen at department of biology and chemistry at the University of Toronto who constantly provided constructive feedback on the user interface of the robot.

Also many thanks to the kind people from our administration, especially Freja Krab Koed Eriksen Christina Rasmussen, and Julie Lyngsø Berg Jacobsen for always being helpful, and Peter Eklund, and Lara Beloff heads of PhD school, and the PhD school for being so generous with my funding requests.

I would like to acknowledge my parents who have made available their supports remotely in a number of ways. I would also like to show my deepest gratitude to my beloved wife Atefeh for her constant support during the long hours and lost weekends required to write this thesis.

1 Introduction

1.1 Outline

We commence this chapter by introducing artificial chemical life research. We will demonstrate the advantages of automation, and review the liquid handling technologies and also the user interfaces for these systems. We discuss the inadequacy of current robotic systems to perform artificial chemical life experiments, and consequently the motivation for developing a robotic system. We will elaborate on the requirements for such a robotic platform. Subsequently, we enumerate the scientific, and engineering contributions of this work. We conclude this chapter by an overview of the chapters in this thesis.

1.2 Artificial Chemical Life Research

Artificial life research is the study of the emergence of life-like behaviours (like chemotaxis) in highly reduced and simple physico-chemical systems. Researchers explore if such systems could represent a primitive form of life on the early Earth or elsewhere in the universe [69]. A key objective of artificial life research is grasping how non-living matter can recreate the essential properties of life [71]. An understanding of the essential properties of life not only may result in synthesizing artificial life and self-reproduction [46], but may also lead to understanding the complexity of natural life.

Droplet experiments are a common type of artificial chemical life experiments. These experiments are aimed to create life-like behaviors by combining simple chemicals. The behaviors seen by these droplets resembles the basic elements of life, e.g. moving, dividing [56], etc. These droplets can even form aggregated structures by embedding DNA molecules on their surfaces [68].

There exist distinct types of artificial chemical life experiments, and diverse droplet behaviors can be observed. Following are few examples describing some of these behaviors.

1.2.1 Moving Droplet Experiment

Figure 1.2.1 [69] shows a typical moving droplet experiment. Heavy nitrobenzene (NB) oil containing 0.5 M oleic anhydride and Oil Red O as colourant is placed in a glass-bottom Petri dish containing both 10 mM oleate pH 12 micelles and a pH indicator, thymolphthalein. The pH indicator is blue at high pH and colourless below pH 11. As soon as the droplet is introduced to the water phase, it breaks symmetry and begins to move directionally around the dish (see supplementary video 1 in Appendix A). As the droplet moves, it leaves a trail of low pH solution. The droplet in this system is moving in response to its own self-generated pH gradient.

This moving droplet experiment is an example of artificial chemical life experiments where the experiment is not just finished by placing the the chemical in the

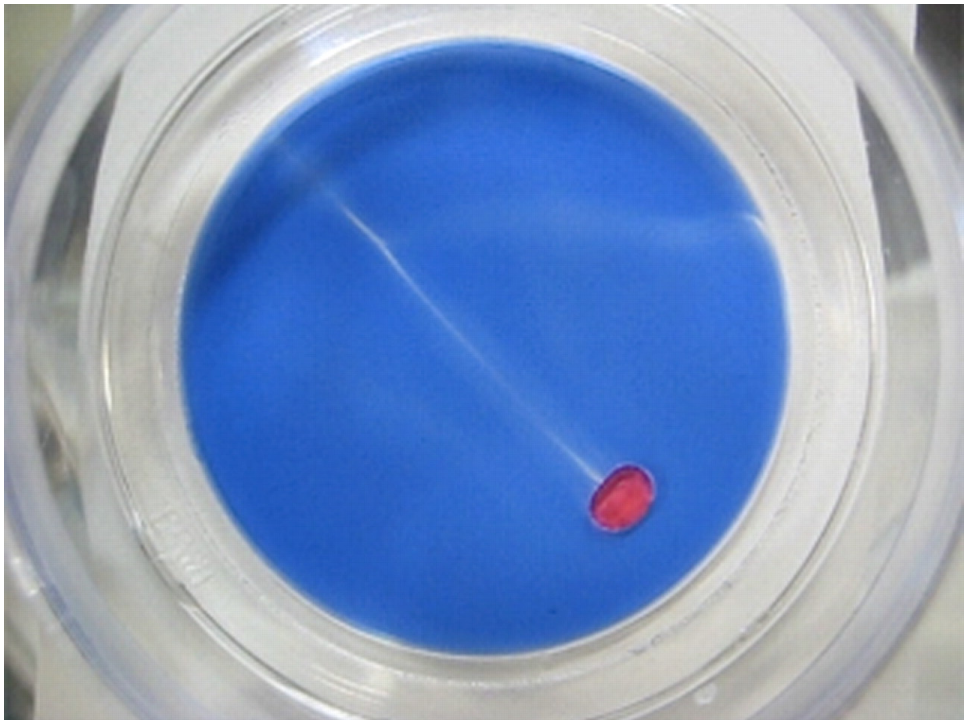


Figure 1.2.1: Droplet moving in an aqueous pool containing surfactant. The droplet (red) moved from the upper left to the lower right edge. The low pH trail was visualized by using the pH-sensitive dye thymolphthalein. Diameter of the section containing the aqueous pool was 27 mm.[69]

vessel, but there is an observation phase which takes a significant amount of time. This has important consequences for automation. This observation phase has great potential for automation, as if automated will save users a significant amount of time.

1.2.2 REACTION TO EXTERNALLY APPLIED pH

In [70] researchers have also shown that the droplet is sensitive to externally applied pH gradients, with movement towards high pH. A droplet can respond to pH gradients in its local environment and move towards high pH solution using its own particular form of chemotaxis. As shown in Figure 1.2.2 [69], a droplet

moving through the aqueous phase senses a pH gradient (blue), climbs the gradient and stops at the point of highest concentration (supplementary video 2 in Appendix A). The droplet has an interface that can sense its local chemical environment and an internal convective flow acting as a motor.

This system is an example of the movement of droplets in concentration gradients, which can be viewed as a simplistic model of life that mimics the behavior of cells that move away from their metabolic waste into regions with fresh nutrients [70]. This system contains only five chemical components, including water.

1.2.3 DROPLET CHEMOTAXIS IN A COMPLEX TOPOLOGY

Figure 1.2.3 [57] shows an example of droplet chemotaxis in a complex topology. In this experiment, the decanol droplet follows the shortest path toward the source of the salt and eventually fuses with the salt-loaded nitrobenzene droplet (supplementary video 3 in Appendix A). The decanol droplet has a purple color, and the nitrobenzene droplet has a yellowish color.

1.2.4 DYNAMICS OF DROPLET SYSTEMS

In [67], to explore the emergent dynamics of droplet systems, a search is conducted over a four-chemical (dodecane, pentanol, octanol and DEP) search space. After mixing the components at different proportions, the formulations are placed into a Petri dish containing the aqueous phase to form droplets. The researchers discovered and characterized a total of nine distinct behaviours, as described in Figure 1.2.4, displaying a great deal of complex, unexpected emergent behaviours (supplementary video 4 in Appendix A). As described in this experiment, it is usually desired in artificial chemical life experiments to examine experiment conditions, and modify them to potentially find new behaviors.

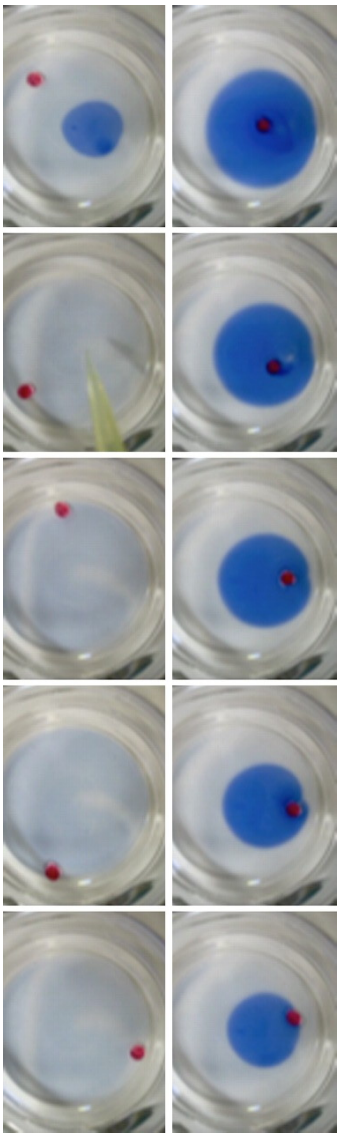


Figure 1.2.2: Chemotaxis of a droplet in a pH gradient. Each frame represents a 10 s interval. The droplet (red) was moving around the aqueous pool until a droplet was introduced that created a pH gradient in fourth frame. The dispersing pH gradient was visualized using the pH-sensitive dye, thymolphthalein. The diameter of the section containing the aqueous pool was 27 mm. (order bottom up, left to right) [69]

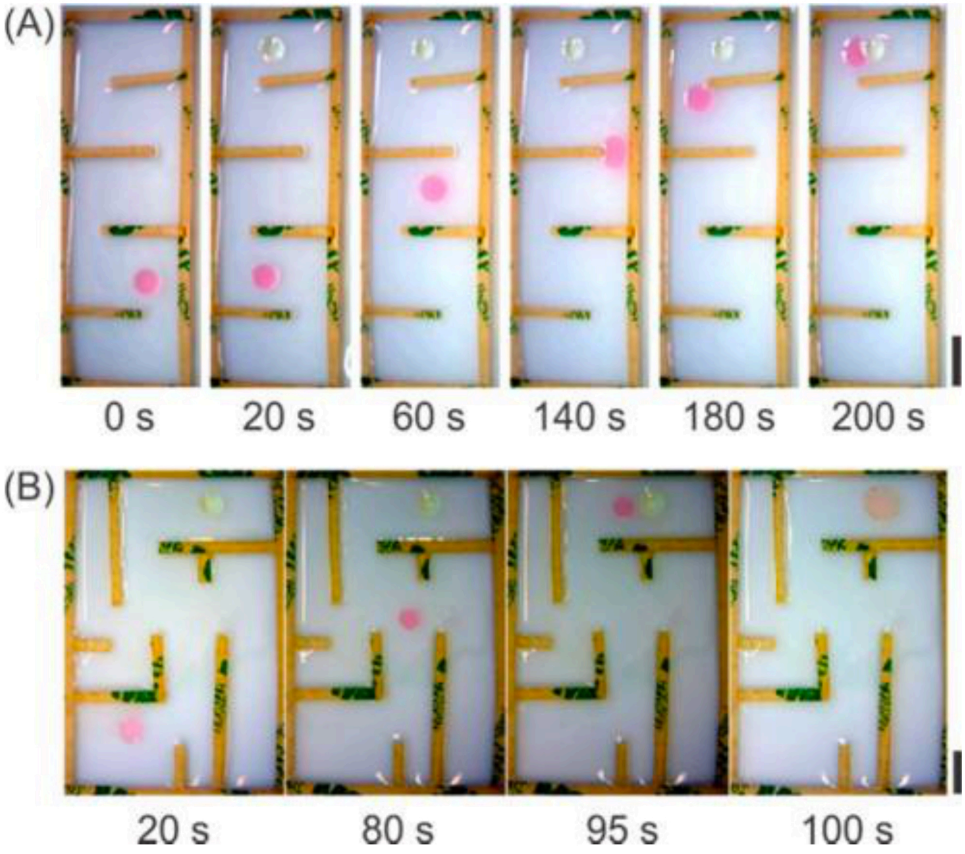


Figure 1.2.3: Examples of droplet chemotaxis in a complex topology: (A) nonlinear path in a channel; (B) nonlinear path in a simple "labyrinth" with dead-end channels. The decanol droplet has a purple color, and the salt diffuses from a stationary nitrobenzene droplet (yellowish color). Scale bars on the right represent 1 cm.[57]

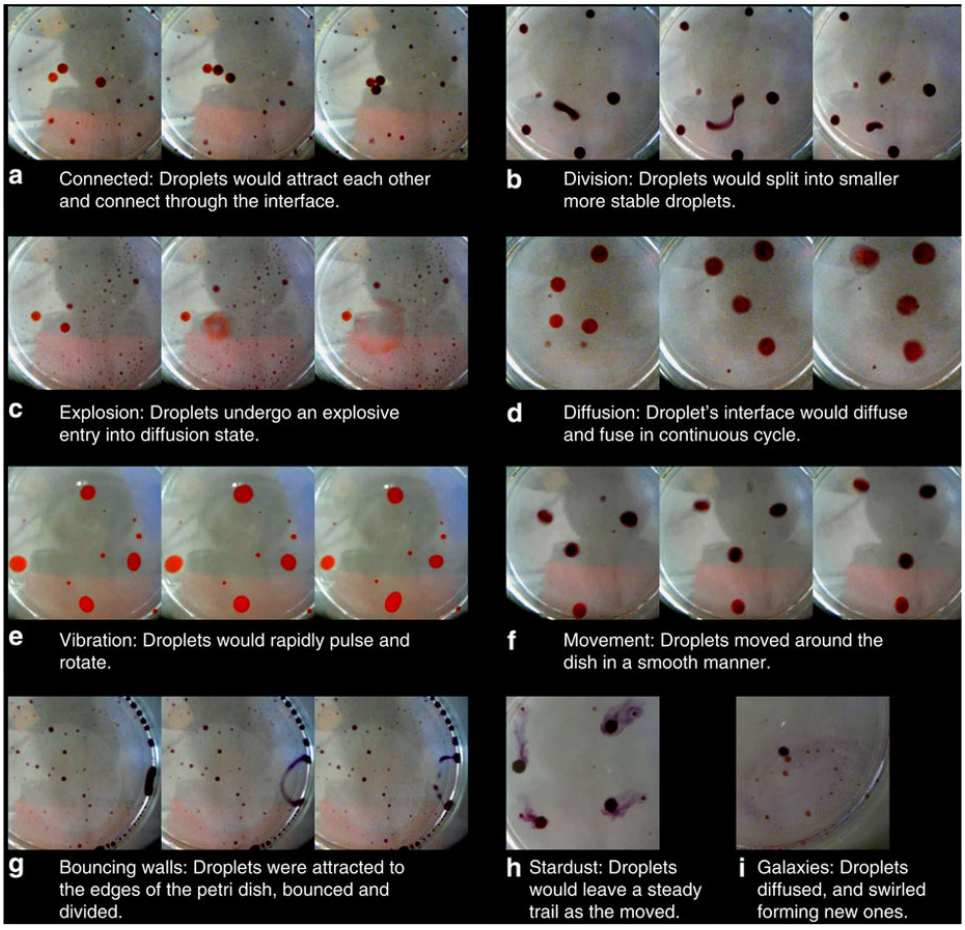


Figure 1.2.4: Variety of behaviours observed in lattice search. Photographs of the droplet behaviour as a function of time (from left to right) for all the traits (given in a–i) except the 'stardust' and 'galaxies' where just one image is shown [67].

1.2.5 Characteristics of Artificial Chemical Life experiments

As described in the preceding experiments, droplet experiments are long lasting experiments in comparison to ordinary liquid handling experiments that are more often high throughput and short. Furthermore, a main characteristic of artificial chemical life experiments is the requirement for online/real-time observation and reaction that is different from ordinary experiments. These experiments require precise timing and relative positioning of reagents with respect to motile droplets. Furthermore, experimental parameters, such as distance between droplet and reagent, time of adding reagent, and concentration of liquids, influencing droplet behavior play an important role in these experiments.

1.3 Motivation for a New Robotic System

In this section we discuss the motivation for developing a new liquid handling robot for performing artificial chemical life experiments. We discuss the advantages of automation for artificial chemical life experiments. Then we discuss why the wide range of available liquid handling robots are not suitable for reactive droplet experiment, and therefore the need to develop a novel robot for artificial chemical life experiments. We then discuss the usability issues regarding liquid handling robots, and the necessity for an intuitive user interface.

1.3.1 Lab automation advantages

Artificial life experiments are particularly well-suited for automation because they often stretch over long periods of time, possibly hours, and often require that the human takes action in response to observed events such as change in droplet size, count, shape, or clustering/ declustering of multiple droplets.

Furthermore, artificial chemical life experiments are prone to systematic errors and noise introduced by humans. This originates from the fact that humans are unable to precisely perceive spatial distances and dynamics. Even if they could locate the precise distance relative to a motile, or non-motile droplet, placing the

droplets on the exact same location by hand introduces significant imprecision. Furthermore, the angle of the pipette tip, the time to dispense liquid, the force of dispensing, and the distance between pipette tip and liquid surface are parameters that could potentially influence the experiment negatively. In addition, over the course of long experiments, human exhaustion could further influence the experiment.

In case of more complicated experiments the need for automation becomes more salient as the experiments become more difficult for humans to perform. An example of these complicated experiments is when the chemical diversity of the components increases. As a result of increasing the diversity in components, new and interesting phenomena may emerge, which depends mainly on the specific combinations of chemicals used. An example is fluid-dynamically controlled self-division [56].

1.4 Review of Liquid Handling Technologies

Delivering liquids, and samples efficiently and accurately is a necessity for many applications in chemistry laboratories. Liquid handling machines facilitate these tasks by aspirating, i.e. drawing, a specific volume of liquid from a source vessel, repositioning the liquid, and dispensing, i.e. delivering, this liquid to the destination vessel. The fluid transferred is also called the sample fluid. Automated liquid handling was first utilized in the late 1980s and 1990s, and it was driven by the needs for genomic sequencing studies, and also the pharmaceutical industry. Since then, this technology has been growing rapidly.

Lab automation has extensively helped chemists perform chemical experiments, which are often repetitive tasks in chemical laboratories. Owing to lab automation, the behavior of the experiment can be verified over many runs and sufficient data can be collected to build a model or attain statistically significant results. Therefore liquid handling technologies alleviate the hardships of dealing with dull, time consuming tasks of doing the same experiment over and over. This prevents experiments from being prone to the numerous types of human error that could be

introduced into the experiment. On the other hand, lab automation is even more advantageous when performing experiments with hazardous chemicals. All the aforementioned advantages, has made lab automation popular and led to development of many liquid handling technologies.

It should be noted that most liquid handling systems are outside the academic domain and therefore this section is a review instead of a literature review. In the absence of academic resources for liquid handling robots, we provide a review of the state of the art in liquid handling robots. Thus the references to the website of these products are provided, should the readers need to obtain more information about the technologies.

We have categorized liquid handling systems regarding the level of automation, reactive functionality, being standalone, and single-channel VS multi-channel. These are important considerations in the choice of a liquid handling systems for chemical experiments, and thus artificial chemical life experiments. In section 1.6, we discuss whether these technologies are appropriate for artificial life experiments.

1.4.1 Level of Automation

Based on level of automation, liquid handling systems can be categorized as bench-top semi-automatic pipettors, automated dispensers, robotic workstations or fully integrated workstations. We describe the technology and considerations regarding each category.

Bench-Top Semi-Automatic Pipettors

Bench-top semi-automatic pipettors are the least automated liquid handling systems. They provide improvements over the manual systems by reducing the difficulties associated with liquid handling when operated by hand. These systems do not use an external computer. These pipettors are usually used along with external machines such as shakers. There exist different types of semi-automatic pipettors to handle 96 or 384 wellplates. As can be seen in Figure 1.4.1, and Table 1, the

Figure 1.4.1: CyBio® SELMA by Analytik Jena semi-automatic pipettor.

CyBi-SELMA [33] is an example of a fast semi-automatic pipettor. The Eppendorf epMotion 96 [19] is another device in this category capable of handling a range of volumes with the same head.

Automated Dispensers

Automated dispensers, also called automated pipettes are another liquid handling system. They improve the automation capabilities of semi-automatic pipettors, but are not still fully automated. They are the least complex type of liquid handlers. In these systems the vessels can be repositioned. As can be seen in Figure 1.4.2 and table 1, the CyBi-Well Vario from Analytik Jena [38] is an example. Vertical pipetting station from Agilent Technologies [42], and VIAFLO ASSIST from Integra [41] are also systems in this category.

Robotic Workstations

Robotic workstations have automation capability not only to aspirate or dispense liquid, but they can reposition the dispenser or the containers. This adds more

functionality to the system to perform more experiments in comparison to automated dispensers. On the other hand, these systems are more complex, and occupy more space. Figure 1.4.3 shows 4LAB™ Automated Low Volume Liquid handling system [1]. As can be seen in table 1, the TTP Labtech mosquito HTS [28] is another example of a robotic system for high-throughput liquid handling. More systems in this category include aBioBot [14], S-PIPETTE from Apricot Design [35], Bravo from Agilent Technologies [17], Andrew from Andrew Alliance [4], EzMate from Arise Biotech [22], Versa 10 from Aurora Biomed [39], Versa 110 from Aurora Biomed [40], Biomek 4000 from Beckman Coulter [15], Micro b processor from Bee Robotics [31], PIRO from Dornier LTF [9], SOLO from Hudson Robotics [34], Beeline from HTZ [5], BenchSmart™ 96 from Mettler Toledo [6], OT.One from OpenTrons [106], Seymotion from Seyonic [6], and X150 from Xiril AG [13].

Table 1

Bench-Top Semi-Automatic Pipettors					
Product	Manufacturer	Pipet no.	Channel no.	Dimensions (W*D*H) in mm	weight
CyBi®-SELMA	Analytik Jena	1	96, 384	307x325x480	18
epMotion® 96	Eppendorf	1	96	215x344x525	20

Bench-Top Semi-Automatic Pipettors			
Product	Open-source	Reactive Functionality	Affordability (for small labs)
CyBi®-SELMA	No	No	No
epMotion® 96	No	No	No

Automated Dispensers					
Product	Manufacturer	Pipet no.	Channel no.	Dimensions (W*D*H) in mm	weight
Vertical Pipetting Station	Agilent Technologies	1	96	381x259x787	32
CyBi®-Well Vario	Analytik Jena	1	96, 384, 1536	960x370x772	40
VIAFLO Assist	Integra	1	8, 12, 16	340x360x400	10

Automated Dispensers			
Product	Open-source	Reactive Functionality	Affordability (for small labs)
Vertical Pipetting Station	No	No	No
CyBi®-Well Vario	No	No	No
VIAFLO Assist	No	No	No

Robotic Workstations					
Product	Manufacturer	Pipet no.	Channel no.	Dimensions (W*D*H) in mm	weight
4LAB	4titude	1	1	590x440x460	25
aBioBot	aBioBot	1	1	NA	NA
S-PIPETTE	Apricot Design	1	96	292x330x521	20
Bravo	Agilent Technologies	1	96	NA	NA
Andrew	Andrew Alliance	1	1	535x29x25	10
EzMate	Arise Biotech	1	1, 8	292x330x521	25
Versa 10	Aurora Biomed	1	4, 8	650x430x520	27
Versa 110	Aurora Biomed	1	1	600x600x450	35
Biomek 4000	Beckman Coulter	1	8	885x1225x505	41
Micro b processor	Bee Robotics	1	1	1000x500x600	NA
PIRO	Dornier LTF	1	1	600x515x470	45
SOLO	Hudson Robotics	1	1, 8, 12	755x500x 610	25
Beeline	HTZ	1	1	NA	NA
BenchSmart™ 96	Mettler Toledo	1	96	500x500x800	30
OT.One	OpenTrons	1	1, 8	600x500x600	20
Seymotion	Seyonic	1	1	800x800x850	50
Mosquito® HTS	TTP Labtech	1	1	390x470x690	27
X150	Xiril AG	1	1, 2, 4, 96	750-1500x700x720	115

Robotic Workstations			
Product	Open-source	Reactive Functionality	Affordability (for small labs)
4LAB	X	X	X
aBioBot	X	X	X
S-PIPETTE	X	X	X
Bravo	X	X	X
Andrew	✔	✔	✔
EzMate	X	X	X
Versa 10	X	X	X
Versa 110	X	X	X
Biomek 4000	X	X	X
Micro b processor	X	✔	X
PIRO	X	X	X
SOLO	X	X	X
Beeline	X	X	X
BenchSmart™ 96	X	X	X
OT.One	✔	✔	✔
Seymotion	X	X	X
Mosquito® HTS	X	X	X
X150	X	x	x

Fully Integrated Workstations						
Product	Manufacturer	Dimensions (W*D*H) in mm	weight	Open-source	Reactive Functionality	Affordability (for small labs)
Biomek i7 Base Unit	Beckman Coulter	1700x810x1120	234	No	No	No
PIPETMAX	Gilson	544x655x696	25	No	No	No
Freedom EVO 200	Tecan	2050x780x870	230	No	No	No
STARplus	Hamilton	2160x795x903	NA	No	No	No
Cell explorer gene pro	PerkinElmer	NA	NA	No	No	No
Cytomat™ 10 C450 Series Automated Incubator	thermofisher	800x822x915	NA	No	No	No
Nano-PlotterTM NP 2.1/E	GeSiM	645x400x375	50	No	No	No

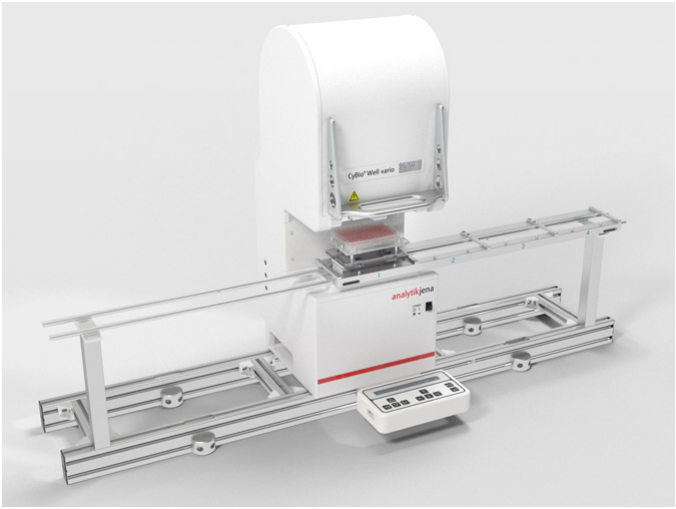


Figure 1.4.2: CyBio® Well Vario by Analytik Jena automated dispenser.

Fully Integrated Workstations

Fully integrated workstations are the most advanced automated systems for liquid handling, and often other modules can be integrated into these systems. Additional modules may include plate readers, centrifuges, bar code readers, PCR devices, colony pickers, shaking instruments, and incubators. Based on the application, appropriate modules can be used. Figure 1.4.4 shows Biomek i7 Integrated System [16] an example of a fully integrated workstation. As can be seen in Table 1, the PIPETMAX automated liquid handling system from Gilson [8], is another example of a multi-functional modular system. More systems in this category include Freedom EVO 200 from TECAN [7], STARPlus from HAMILTON [11], Cell explorer gene pro from PerkinElmer [32], Cytomat™ 10 C450 Series from Thermo Fisher [12], and Nano-Plotter 2.1 from Gesim [27].

1.4.2 Reactive functionality

The liquid handling systems can also be investigated according to their reactive functionality. Reactive functionality can be either getting feedback from the sys-

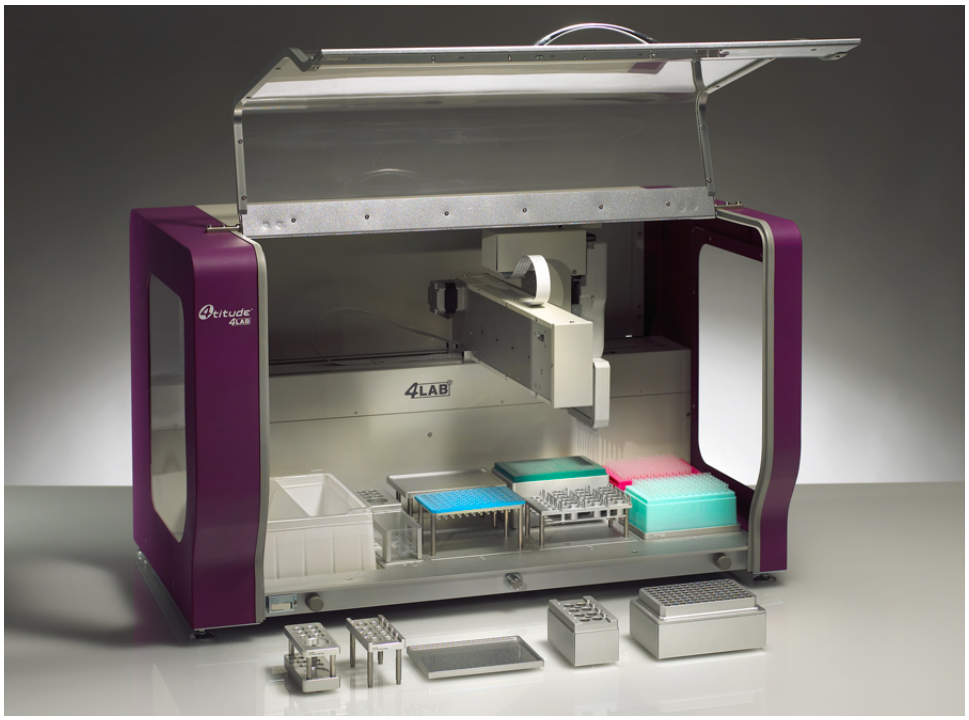


Figure 1.4.3: 4LAB™ Automated Low Volume Liquid handling system.

Figure 1.4.4: Biomek i7 Integrated System.

tem, or feedback from experiment.

Feedback from the system

Reactive functionality of a liquid handling system is to satisfy the functionality requirements in order to improve the capabilities of the system. For example, Andrew has a reactive functionality to read QR codes in order to recognize the type of vessels, or it has a camera to analyze the image to read the amount of liquid aspirated by the pipet.

Another example for getting feedback from the system is detecting the liquid level. Capacitive liquid level detection is one way to determine the height of the liquid surface, and interact with the experiment based on that. Alternatively, pressure based liquid level detection uses data obtained from a pressure sensor to detect whether the tip is approaching the surface, touching the surface, or penetrating through the liquid, and interact with the experiment based on the data.

Feedback from experiment

Some robotics systems available in the market have cameras to take images or videos from the experiment, but most of them do not use the data from the experiment as feedback for the system. For instance, there are applications where there is limited and simple processing on samples to recognize color changes.

1.4.3 Standalone

Many robotic systems require an external computer to operate. However, there are some stanadlone robotic platforms that do not need the computer. As can be seen in the table, BenchSmart™ 96 from Mettler Toledo is an example of a standalone liquid handling robotic system.

1.4.4 Single-Channel VS Multi-Channel

The liquid handling systems provide single-channel or multi-channel functionality.

Single-Channel

Single-channel instruments have only one channel to deliver liquid. It means they are more flexible. However, their throughput is limited. The single-channel machines are less common than multi-channel liquid handling machines

Multi-Channel

Multi-channel systems have either a small number of channels (4, 8, 12, 16) or a large number of channels (96, 384). The smaller number channels are more flexible as the distance between the channels can be set. Multi-channel systems are more common than single-channel liquid handling systems. Multi-channel systems provide high throughput liquid handling functionality.

1.5 Review of User Interfaces for Liquid Handling Robots

It should be noted during the development of our platform and the corresponding user interface, none of the liquid handling robots provided an accessible version of their user interface if you had not purchased their product. Therefore our investigation was based on visiting labs that had purchased liquid handling machines, and asking actual users of these interfaces regarding their usability. An informal investigation showed the users were often dissatisfied regarding the user interface of these systems. However, we introduce the user interfaces which have been recently accessible (after we actually developed our interface). They are state-of-the-art examples of user interfaces. However, they need to be extended in several ways. Similarities in these user interfaces with the final user interface developed in this thesis proves the effectiveness of our design.

Andrew [4] is a successful example of recent semi-affordable commercial low-throughput liquid handling platforms. This platform can be versatile. Andrew is designed with DOMINO™ blocks, magnetic tiles on the working deck of Andrew that are designed to host consumables, e.g. bottles, tubes, microplates, etc. These DOMINOS of different kinds can be configured as needed. AndrewLab is the

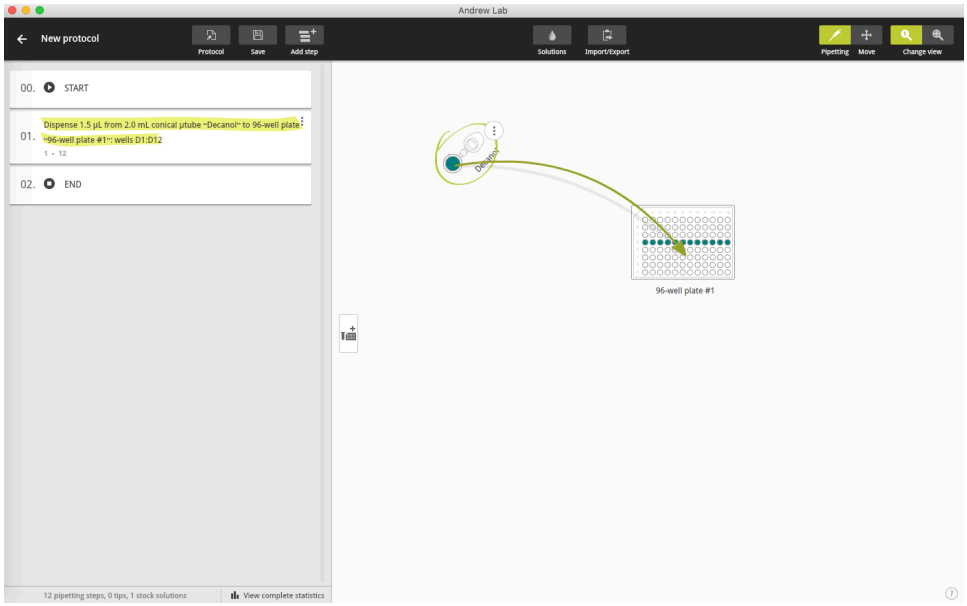


Figure 1.5.1: Screen shot of AndrewLab application.

software for from Andrew Alliance. It is not directly downloadable, but Andrew sends a link to the application upon registration. It is a desktop software, and requires installation of Adobe AIR runtime, and also the AndrewLab application. So the user interface cannot be run out of the box on any platform, i.e. requires installation. Figure 1.5.1 shows a screen shot of the application.

In the AndrewLab software, the users have to design protocols. Designing protocols starts by dragging consumables into the virtual bench. As can be seen in the Figure 1.5.1, the virtual bench is the workspace in this application. The user can choose from various consumables including 96-wellplates, 384-wellplates, 1.5mL conical μ tubes, 2.0mL conical μ tubes, flat bottom vials, etc.

As the next step, the user indicates the possible stock solutions. This step includes choosing the name, concentration, and volume of the solution. Finally, the user drags among the consumables. For instance, starting a drag from a tube and ending the drag on a wellplate. Having done this, the user will be prompted for some actions. For instance, the user needs to specify the destination, e.g. if it is a

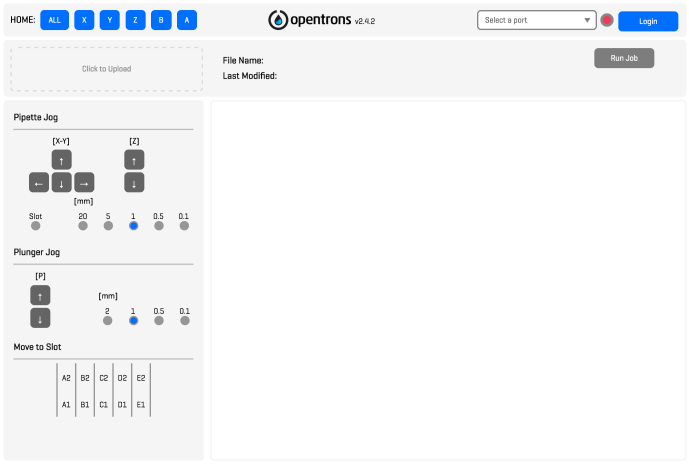


Figure 1.5.2: Screen shot of OpenTrones application.

wellplate, the wells should be selected. The user also needs to specify the volume of the liquid to be transferred, the pipet, and the parameters for this transfer, e.g. the tip touching the liquid, or an air cushion for high viscosity liquids. The user can add steps to the protocol, or investigate the contents of a consumable at different steps.

This application cannot be used to run multiple robots in parallel, nor tasks can be run in parallel. Furthermore, tasks cannot be performed collaboratively. The use of Adobe AIR provides code reuse on Mac OS, and Windows. However, we encountered unexpected crashes on the latest version of the application (Version 1.5.3) when designing protocols in this application on a Mac. Moreover, AndrewLab does not come with any sample protocols. Furthermore, the AndrewLab software is not open-source, and there is no support for Linux.

OpenTrones [106] is another successful example of liquid handling robots making the price affordable even for small labs, through open-source software. OpenTrones provides the OT App, as can be seen in Figure 1.5.2 . It is a desktop applica-

tion, which can be run on Mac OS, Windows, and Linux. The OT App comes with a protocol library, providing sample protocols for basic pipeting, sample preparation, etc. However, accessing protocols is not well integrated into the software. The users need to go to the OpenTrones website to download a specific protocol. Then, they have to upload this protocol on the OT Software. Another limitation is that the users of the OT App cannot modify the uploaded protocol. They can only modify the downloaded protocol which is in the Python programming language file. This may not be feasible for all users, as they may not be familiar with programming. Having edited this Python file in an IDE (Integrated Development Environment), or text editor, they can upload it into the OT App. In other words, to develop a protocol for a custom application, users need to have programming skills.

This application cannot be used to run multiple robots in parallel, nor tasks can be run in parallel. Furthermore, tasks cannot be performed collaboratively. OpenTrones' application programming interface is open source.

Next we look at the user interface of Universal Robots robot arms. Universal Robots is a manufacturer of smaller flexible industrial collaborative robot arms. These robot arms can automate various tasks, including assembly, painting, screw driving, labeling, injection molding, welding, packaging, polishing, etc.

The Universal Robots software is called PolyScope. When the users need to program the robot, the page 1.5.3 is presented to the users. Here the users need to click "Empty program" to start programming a new task. The proceeding page can be seen in Figure 1.5.4. On the left pane, the robot program is displayed which is called the program tree. The program tree contains all the commands executed by the robot during the program. The commands are executed sequentially. More details regarding the Universal Robots user interface is provided in Appendix B.

This user interface shares the same limitaion as previous ones we have discussed. This application cannot be used to run multiple robots in parallel, nor tasks can be run in parallel. Furthermore, tasks cannot be performed collaboratively.

Figure 1.5.3: New program screen.

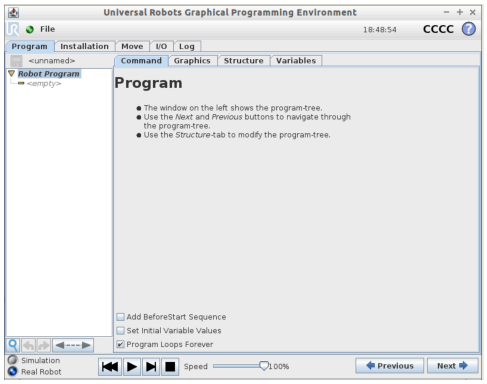


Figure 1.5.4: Robot programming initial page.

1.6 Limitations of Current Liquid Handling Technologies for Artificial Chemical Life

In this section we describe the limitations of current liquid handling technologies for artificial chemical life in regards to their robotic technology, and their user interfaces.

1.6.1 Technology

The wide range of available liquid handling robots in the market are not suitable for reactive droplet experiment. Firstly, most of the robots in the market have been designed to maximize accuracy and throughput for short-duration experiments, which results in high price of these robots. Although, high throughput can be useful for artificial chemical life experiments, it is less of a priority because the liquid handling operations only use a fraction of the time of an experiment. In other words, high throughput liquid handling technologies are over optimized in case of artificial chemical life experiments, and therefore unnecessarily costly. Consequently, these robots are not suitable for artificial chemical life experiments. In addition, easy modification of experiments based on the result of previous experiments is necessary in artificial chemical life research. For instance, a new droplet behavior may be observed or become of interest to scientists. This is not a possibility with available robots, as they are designed to optimize performing specific types of experiments. Therefore these robots do not address the needs of artificial chemical life.

The robots in the market are not capable of performing artificial chemical life experiments due to the reactive nature of these experiments. As discussed in the previous section, some liquid handling systems have sensing functionality. However, this sensing functionality is for the performance of the system, e.g. barcode readers for determining the location of vessels, or a pressure sensor, and does not provide feedback for performing experiments. Therefore, even the emergence of affordable commercial low-throughput liquid handling platforms in recent years, has not con-

tributed to artificial chemical life research. Andrew [4], and OpenTrones [106] are successful examples with a semi-affordable price. However, there is little sensing functionality involved in these platforms to provide feedback for reactive experiments. For instance, the sensing functionality in Andrew is reading barcodes for locating vessels, or determining the contents of a vessel as well as processing images to read the numbers on a pipet in order to determine the aspirated/dispensed liquid. The sensing functionality in OpenTrones is only limited to taking images or recording videos, and processing them later offline when required.

The robots developed so far cannot be generalized to perform different types of artificial chemical life experiments because of the wide range of these experiments. The need to perform reactive low throughput liquid handling experiments has resulted in the development of custom platforms for specific tasks. DropBot [66] is an example of robots developed to perform artificial life experiments in a chemistry laboratory. However, these platforms are not open-source, and as they are not modular, they are not easy to extend or versatile enough to be used for experiments for which they were not designed.

1.6.2 USABILITY ISSUES OF LIQUID HANDLING ROBOTS

Like many complex engineering systems, liquid handling robots have traditionally had complex and difficult to use interfaces [65]. Thus in addition to providing a low cost and extensible open source system, we wanted to provide a good user interface that would be more intuitive and easier to use. It was particularly important to make the use of the robot more efficient [99]. Furthermore, a good interface design was needed to reduce errors in setting up and programming experiments. When dealing with chemicals, errors are potentially dangerous, and a good user interface can improve safety by providing good feedback. Last but not least, the long-term costs of ownership and use of the robot should be lower with a good interface, reducing the need for user training, and will lead to greater efficiency and accuracy in performing experiments.

Usability of liquid handling robots is another challenge of the technology. Human-

Robot Interaction for these systems, i.e. interaction of chemists and biologists with a liquid handling robot, involves many challenges both with respect to the technical challenges, as well as with respect to the human-centred aspects involved. The former relates to technical requirements for users in order to be able to use the interface, and operate the robot. The latter includes issues such as people's expectations of robots, acceptability of interaction, and interaction with the robot that is comfortable for humans [60].

Furthermore, we intended to develop an interface to enable us to teleoperate the robot. Owing to such user interface, users can access the robot from any location of their choosing, and from any browser-enabled device. On the other hand, teleoperation can be a challenging task as the operator is remotely located, the operator's situation awareness of the remote environment can be compromised and resulting objective effectiveness can suffer [58]. This should be carefully addressed in the design of the interface.

Scalability of the user interface in terms of the number of robots to be controlled is another challenge. Human-robot interfaces for interacting with multiple robots are very different from single-robot interfaces. The central design challenge is developing techniques to maintain, program, and interact with the robots without having to handle them individually [81].

1.7 Requirements for our Robotic Platform

In section 1.3.1, we discussed the advantages of automation for biochemical experiments. This forms the motivation for developing a robot to perform feedback based experiments. We discussed the shortcoming of available automation solutions for these types of experiments in section 1.6.1. In order to address these shortcomings our solution has to satisfy the following requirements.

1.7.1 Routine liquid handling functionality

The basic required functionality for our solution is routine liquid handling functionality, i.e aspirating a specified amount of liquid from one reaction vessel, e.g.

a Petri dish, or well plate, and dispensing the desired amount of liquid in another reaction vessel. This core functionality is present in all liquid handling robots, and is not a novel aspect of our solution. However, as aspirating, dispensing, and moving liquids is the basis for a lot of chemical experiments, it should be present in our solution for artificial chemical life experiments, too.

1.7.2 Performing Feedback based experiments

In order to be able to perform artificial chemical life experiments, we must interact with the experiment based on the visually observed behavior. Therefore, our solution requires to be able to perform feedback based experiments.

1.7.3 Interacting with experiments in real-time

Another requirement for users performing artificial chemical life experiments is to be able to analyze experimental data in real time, and use this data to interact with the experiment in real time. This is in contrast to conventional experiments where the data is processed after the conclusion of the experiment. This functionality opens doors to new possibilities for biochemical researchers.

1.7.4 Affordability

The platform to be developed has to be affordable so it can be used in laboratories with limited resources. Most available liquid handling robots in the market are expensive. A low price can increase the application of such a platform, hence promoting research in the field of artificial life.

1.7.5 easy software management for the system, and fast setup time

Short setup time, and ease of software management for the system is another requirement for our solution. A platform capable of processing data for artificial life experiments would require different software libraries. Installing, and managing these dependencies on different hardware, and operating systems would be difficult and time consuming. Furthermore users should be able to use our platform

on any system they wish, and not require to spend a lot of time for installing all the software if they use a new computer. Therefore we need to decouple control software from user software.

1.7.6 Intuitive user interface

Another requirement for our robotic platform is an intuitive user interface enabling users to use the functionality of our platform. Most of the people performing artificial chemical life experiments, are chemists, biologists, etc. The users would prefer a graphical user interface rather than programming the experiments. Therefore enabling our intended users to use the features of our robot is another requirement. Furthermore, we had to choose a programming language they preferred, if they were required to program an experiment.

Most of the liquid handling robots in the market lack an intuitive user interface. Providing a good user interface for our platform would mean less documentation for users, and more efficient use of their time.

1.7.7 Open source multi platform user interface enabling remote real time control

An open source multi platform user interface enabling remote real time control is another requirement for our robotic system. We need a user interface which can be accessed on various platforms, e.g. Mac OS, Windows, Linux, or different devices, for instance, Mac, PC, mobile phone or tablet. This enhances the portability of our robotic system.

Remote real time control of the robotic system provides several advantages. The users can access the robot at a location that suits them, and not necessarily by the robot. This is crucial as experiments in chemistry and biology labs usually take a long time, and such an interface grants users freedom to switch between their desired tasks, as they can access the robot remotely. Furthermore, the user interface provides users with notifications, so they are informed if an experiment is finished, or if there is an error during the experiment, such as a vessel running out of liquid,

or a pipette missing, so they can take the appropriate action.

Furthermore, an open source solution needs to be provided for our platform. Software for the user interface of liquid handling robots in the market is proprietary, which has several downsides. The licensing fee of proprietary software is expensive. Furthermore, users are reliant on the program's developer for all updates, support, and fixes. Software updates may not be fast depending on the size of the developer team. In addition, customization is very limited or not possible for proprietary software as access to the source code is restricted. An open source user interface solves these problems.

1.7.8 resource Sharing and reusability of experiment protocols

The ability to share the resources among different users is another requirement for our platform. Many of the artificial chemical life experiments have a lot of commonality and are performed by many users. Therefor it is useful to enable users to reuse code, and take advantage of already developed experiments.

1.8 Contributions

The main contributions of this thesis fall into two main categories: 1) scientific contributions, and 2) engineering contributions. Furthermore, this work can be used as a guideline for users going through developing a similar liquid handling robotic system.

1.8.1 Scientific contributions

The scientific contributions of this thesis include 1) Showing that the quality of data obtained from reactive experiments can be improved with the use of a robotic platform, 2) Demonstrating real-time user interaction with a chemical experiment through a robotic platform, 3) User interface design for liquid handling robots, 4) Open source multi-platform user interface for remote real time control of a robotic system, and 5) Cloud-based user interface for a distributed robotic system.

These contributions resulted in nine publications. Furthermore the developed liquid handling system has been patented, and has lead to the spin off Flow Robotics A/S [105].

Showing that the quality of data obtained from reactive experiments can be improved with the use of a robotic platform

We demonstrate in this thesis that the quality of data from artificial chemical life experiments is improved using automation. We have developed a robotic system to automate a large class of droplet experiments in artificial chemical where the behavior of motile droplets in a reaction vessel such as a petri dish is of interest. To enable feedback-based experiments, we have integrated computer vision into the design of a liquid handling robot, and developed the required software functionality. Computer vision enables the robotic system to detect relevant changes, either in individual droplet behavior, e.g. change in droplet area, position, speed, direction, acceleration, color, shape, number of droplets, or group droplet behavior, e.g. droplets clustering or declustering. Having detected the specified behavior change, precise automation enables the robot to interact with the experiment, e.g. dispense or aspirate a chemical at a specific point relative to a droplet center.

The following papers are published in regards to this part of our contribution:

- Nejatimoharrami, F., Faina, A., & Støy, K. (2016). Robotic Automation to Augment Quality of Artificial Chemical Life Experiments. In Proceedings of the Artificial Life Conference 2016 (pp. 634-635). Massachusetts Institute of Technology Press.

- Faina, A., Nejatimoharrami, F., Stoy, K., Theodosiou, P., Taylor, B., & Ieropoulos, I. (2016). EvoBot: An Open-Source, Modular Liquid Handling Robot for Nurturing Microbial Fuel Cells.

- Nejatimoharrami, F., Faina, A., & Støy, K. (2016). An open-source, low-cost robot for performing reactive liquid handling experiments. In 2016

Society for Lab Automation and Screening International Conference & Exhibition.

- Theodosiou, P., Faina, A., Nejatimoharrami, F., Stoy, K., Greenman, J., Melhuish, C., & Ieropoulos, I. (2017, July). EvoBot: Towards a robot-chemostat for culturing and maintaining Microbial Fuel Cells (MFCs). In Conference on Biomimetic and Biohybrid Systems (pp. 453-464). Springer, Cham.
- Janská, P., Zadražil, A., Nejatimoharrami, F., Faina, A., Støy, K., & Čejková, J. (2016). Collective behaviour in droplet systems. In 43rd Conference of Slovak Society of Chemical Engineering.

Demonstrating real-time user interaction with a chemical experiment through a robotic platform

The platform we have developed makes real-time interaction with the experiment possible. This is because the robot can record accurate experimental data in real time. The possibility of providing real time experimental data, in contrast to the offline data processing at the end of the experiments, the ability to place new droplets a precise distance from any droplet center, and the ability to track fast moving droplets open doors to new possibilities in laboratory research. Our robot's data logger provides various kinds of data about the experiment with 0.1 millimeter position accuracy, and 4% droplet area accuracy at millisecond time steps. This enables building precise models for chemical experiments, and accurate verification of hypotheses. It should be noted that data from other types of sensors such as voltage or pH sensors can also be readily utilized to provide feedback for a specific experiment if required as explored by colleagues [61].

Furthermore, the user interface provided with our robotic system provides a real time video feed of the experiment, therefore enabling users to interact with the experiment while it is happening. Therefore if the users observe an interesting behavior, they don't need to wait until the end of the experiment. They can modify the remaining steps of the experiment on the fly.

The following papers are published in regards to this part of our contribution:

- Nejatimoharrami, F., Faina, A., & Stoy, K. (2017). New Capabilities of EvoBot: A Modular, Open-Source Liquid-Handling Robot. SLAS TECHNOLOGY: Translating Life Sciences Innovation, 2472630316689285.

- Nejatimoharrami, F., Faina, A., & Støy, K. (2016). A Low Cost Standalone Open-Source Robotic Platform for Performing Feedback based Liquid Handling Experiments. In 2017 Society for Lab Automation and Screening International Conference & Exhibition.

- Faíña, A., Nejatimoharrami, F., & Støy, K. (2014). Towards EvoBot: A liquid-handling robot able to automatize and optimize experiments based on real-time feedback. Exploiting synergies between biology and artificial life technologies: tools, possibilities, and examples.

- Faíña, A., Nejatimoharrami, F., & Støy, K. (2017). EvoBot: An Open-Source, Modular Liquid Handling Robot. IEEE Transactions on Mechatronics (under review).

User interface design for liquid handling robots

In this work, we come up with a user interface design for liquid handling robots that is intuitive, as well as being easily extensible for new experimental protocols in biological, and chemical laboratories. In order to design an intuitive user interface, we have studied different users in various fields dealing with liquid handling tasks to fully understand their needs and concerns. This user study, and analysis of requirements have led to the design of a user interface based on UCD (User centered design) process. Using a special version of heuristic evaluation designed for robot interfaces, and iterative feedback from users, we were able to identify key usability problems in the user interface prototypes and develop a user interface to be implemented on our robotic system that represents a significant advance of prior user interfaces for liquid handling robots.

The following paper is published in regards to this part of our contribution:

- Nejatimoharrami, F., Faina, A., Jovanovic, A., St-Cyr, O., Chignell, M., & Stoy, K. (2017, April). UI Design for an Engineering Process: Programming Experiments on a Liquid Handling Robot. In Robotic Computing (IRC), IEEE International Conference on (pp. 196-203). IEEE.

Open source multi-platform user interface for remote real time control of a liquid handling robotic system

We have developed a user interface which can control the robot on multiple platforms and devices. Remote control of the robotic system enhances effective use of time, and usability, and provides notification of potential errors. The open source software for the user interface allows users to customize the experiment protocols based on their needs.

Cloud-based user interface for a distributed liquid handling robot

In this thesis, we deploy a cloud-based user interface for a distributed automation system. Multiple users use our robotic platform. Therefore quality, reusability, deployability, and maintainability of software become central and crucial for performing the required tasks. We provide a design, implementation, and deployment that enables users to share experimental code, and reuse already developed protocols for experiments. Therefore, the robotic system can be used in real time by multiple users. In addition, users can work on the same experiment collaboratively, and simultaneously. They can modify the same experiment as a team, or receive notifications regarding experiment progress.

The following paper is published in regards to this part of our contribution:

- Nejatimoharrami, Andres Faina, & Kasper Stoy, "A Low-Cost Open-Source Cloud-based Liquid Handling Robotic Platform for Performing Remote Real-Time Collaborative Experiments", under review in the seventh Annual International SLAS (Society for Laboratory Automation and Screening) Conference (SLAS 2018)

1.8.2 Engineering contributions

The engineering contributions of this thesis include 1) Developing software for experimental support , 2) Computer vision for droplet tracking, 3) a robotic platform with integrated controller hardware, 4) Integrating controller hardware into the robotic platform

Software for experimental support

The API (application programming interface) developed for our liquid handling robot provides the basis for performing a large class of artificial chemical experiments. The software developed for our platform, addresses feedback based experiment requirements. These requirements include detecting interaction with the experiment based on the observed changes. The developed software equips our robotic system with a data logger which is able to log different types of experiment data. In addition, the software provides a world coordinate system which transforms coordinate systems of different modules to each other, enabling moving different modules to the same location. It also enables customization for different robots by enabling users to use the same software independent of the specific parameters of the robot they use. In addition, our software provides an interactive graphical user interface, which enables chemists to interact with the experiment. We provide numerous examples in our codebase, demonstrating the use of the API. We also maintain the documentation for the hardware and software of our robotic system in our wiki.

Computer vision for droplet tracking

The vision software developed for the liquid handling robot, enables detecting changes in the experiment condition based on sensor input. The vision API of our developed robotic system is responsible for processing the camera frames from the experimental layer to extract data about the experiment, e.g. droplet behaviors. In order to recognize droplets in artificial chemical life experiments, and analyze their behavior we use a series of image processing operations for image data anal-

ysis. Furthermore, in order to extract relevant experiment data, our robot's vision API calibrates the coordinate system of the robot and the camera, as they are in millimeters and pixels, respectively.

Integrating controller hardware into the robotic platform

In this work, we have developed a standalone robotic platform with integrated controller hardware. Therefore user software for programing experiments is detached from control software. This platform eases package management as installing, and managing software libraries required for feedback based experiments on different hardware, and operating systems would be difficult. Therefore, regardless of the choice of system users want to use, they don't need to install any libraries. Such a platform is much more affordable owing to elimination of the need for an external computer.

Integration of modern software technology into a real robotic system

Another contribution of this work, is applying modern software design principles, software technology, and practices to a real robotic system. Software Engineering is fast growing, and there have been a lot of advances. Important developments relevant to robotic systems include those related to component based software design, event-driven architectures, separation of concerns, software frameworks for real time robotics systems, and combining high level decision making with low level control systems. Applying the software architecture design, software technologies, and standard development processes to a real robotic system comes with challenges. We discuss the challenges, and our approach to solve them in order to achieve a fast responsive robotic system.

1.9 Contributors

EvoBot was designed by Andres Faina, a postdoctoral researcher, in the context of the EVOBLISS EU project. Prof. Kasper Støy, my supervisor, has also had a great contribution in developing software for the robot. We have built seven copies of EvoBot in the Robots, Evolution, and Arts Lab (REAL), to be used by our EVOBLISS partners. The partners are University of Trento, Italy, University of glasgow, UK, University of Bristol, UK, Karls Ruhe Institute of Technology, Germany, University of Chemistry and Technology Prague, Czech Republic, and University of Toronto, Canada.

While chapters 2, and 3 have been collaborative work, chapters 4 to 9 are the scientific, and engineering contributions of the author. In chapter 2, the author has specifically been involved in building robots, and performing tests for verifying the performance of the robot. In chapter 3, section 3.2.1 is the contribution of Prof. Kasper Stoy, and assistant professor Andres Faina. Section 3.2.2 through section 3.2.9 are the contributions of the author.

The partners of the EVOBLISS consortium, including Prof. Martin Hanczyc, Prof. Jitka Cejkova, and Silvia Holler helped us perform artificial chemical life experiments. Jørn Lambertsen was involved in building EvoBot.

Prof. Mark Chignell, Prof. Olivier St-Cyr, and Andrea Jovanovic have contributed to the design of the user interface of the robot. The members of the EVOBLISS consortium helped evaluate the user interface for the robot. In particular, Florian Blauert, Pavlina Theodosiou, and Silvia Holler provided feedback on the interface, and evaluated it. Also, researchers in University of Toronto, BioZone, and department of chemistry, Particularly, Naveen Venayak, Brian Nguyen, and Jenna Blumenthal provided constant constructive feedback on the user interface of the robot.

1.10 Dissertation Overview

This dissertation is organized into two main parts. In part I, we demonstrate how we can perform artificial chemical life experiments by addressing hardware and software requirements of a liquid handling robot. We demonstrate the applicability of the robot to a comprehensive class of feedback based experiments. Then we demonstrate the steps towards developing a user interface for liquid handling robots. First, we discuss the implementation of a platform which provides the software infrastructure for developing the user interface. This infrastructure includes both hardware and software for a standalone robotic platform. We continue this route by designing the user interface, developing the user interface, and then deploying the user interface. In Appendix E, we include our scientific contribution presented as papers.

This thesis consists of 9 chapters:

Chapter 2 *Hardware functionality needed for artificial chemical life*

In this chapter, we introduce EvoBot, a robot developed to perform liquid handling experiments.We commence this chapter with a physical description of the robot. We will introduce the modular design of EvoBot. We then report on its performance. We explain how a sensing layer for EvoBot provides us with the hardware functionality needed in artificial chemical life.

Chapter 3 *Required Software for Artificial Chemical Life Experiments*

In this chapter, we discuss the software functionality required in artificial chemical life experiments. Addressing these software requirements allows us to use a liquid handling robot, introduced in chapter 2, for artificial chemical life experiments.

The software developed for this liquid handling robot, EvoBot, addresses artificial chemical life experiment requirements. These requirements include detecting changes in the experimental conditions, based on sensor input, and interacting with the experiment based on the

observed changes. In this chapter, we introduce the software architecture including firmware, vision system, camera calibration, coordinate system transformation, logging experiment data, customization for different robots, graphical user interface for interactive experiments, experiment templates, and the robot's wiki.

Chapter 4 *Interactive experiments*

In this chapter, we discuss how we have made a liquid handling robot usable for artificial chemical life experiments by addressing hardware, and software requirements explained in chapters 2 and 3. We demonstrate the general non-reactive liquid handling functionality of the robot by performing the routine liquid handling experiments with the robot. We provide examples of precise droplet placement, and investigate the reactive functionality of the robot. EvoBot has been used for performing numerous reactive liquid handling experiments. We include three use cases, as droplet behaviors in these experiments and the required type of robot interaction are comprehensive enough to be generalized to various reactive experiments. The use cases are examples of what is possible with EvoBot, and are functionalities requested by our partners in Evobliss EU project to enable them not to perform a specific experiment but a class of experiments. We compare the way non-automated experiments were performed without the robot for demonstrating the ease, accuracy and precision of results obtained by the robot. Furthermore, easy configurability of the robot for "needed to be modified" experiments is examined. We discuss how EvoBot has realized experiments that formerly were difficult to perform.

Chapter 5 *Robotic Platform with Integrated Controller Hardware*

This chapter comprises our first step towards developing a user interface for liquid handling robots. In this chapter, we discuss the implementation of a platform which provides the infrastructure for developing the user interface. This infrastructure includes both hardware and software for a stan-

dalone robotic platform. In this chapter, we focus on the system design.

Chapter 6 *User Interface Design*

In this chapter we continue our goal for developing a user interface for liquid handling robots. We focus on the the user interface design and evaluation. We demonstrate the significance of user interface design. Then we talk about user requirement analysis, and user studies. We demonstrate the important design principles taken into account for designing the user interface. Then we talk about the mock up for the user interface. This is followed by the heuristic evaluation of the user interface. We conclude this chapter by evaluating the user interface.

Chapter 7 *Software Architecture for an Open Source Multi-platform User Interface for Remote Real Time control of a Robotic System*

In this chapter focuses on the software architecture of a user interface for a liquid handling robot. We describe the different user interfaces for our robot, such as a Python based user interface, an iPad user interface, and a web interface using LAMP stack, and finally a MEAN stack user interface. We demonstrate the implementation, advantages, and disadvantages of each user interface.

Chapter 8 *Cloud-based software architecture for a distributed liquid handling system*

This chapter addresses another requirement for our robotic system which is resource sharing and reusability of experiment protocols, the ability to work on the robotic system collaboratively, and parallelizing experiments on different robotic systems.

In this chapter, we describe a cloud based software architecture for deploying our user interface. A cloud based implementation is a paradigm shift from single user single platform concept to single user multi platform, multi user single platform, and multi user multi platform approaches. This chapter also comprises our final step towards developing a user interface for liquid handling robots, which is deploying the user interface on the cloud.

Chapter 9 *Conclusions & Future Work*

This chapter concludes my findings in this thesis and proposes some future research directions.

In appendix E, we include our scientific contribution presented as following papers:

Paper 1 *New Capabilities of EvoBot: A Modular, Open-Source Liquid Handling Robot*

Paper 2 *Robotic Automation to Augment Quality of Artificial Chemical Life Experiments*

Paper 3 *UI Design for an Engineering Process: Programming Experiments on a Liquid Handling Robot*

Paper 4 *An Open-Source, Low-Cost Robot for Performing Reactive Liquid Handling Experiments*

Paper 5 *A Low Cost Standalone Open-Source Robotic Platform for Performing Feedback based Liquid Handling Experiments*

Paper 6 *EvoBot: An Open-Source, Modular Liquid Handling Robot for Nurturing Microbial Fuel Cells*

Paper 7 *Towards EvoBot: A liquid-handling robot able to automatize and optimize experiments based on real-time feedback*

Paper 8 *EvoBot: Towards a robot-chemostat for culturing and maintaining Microbial Fuel Cells (MFCs)*

Paper 9 *Collective behaviour in droplet systems*

Paper 10 *EvoBot: An Open-Source, Modular Liquid Handling Robot (under review in IEEE Transactions on Mechatronics)*

Paper 11 *A Low-Cost Open-Source Cloud-based Liquid Handling Robotic Platform for Performing Remote Real-Time Collaborative Experiments (under review in SLAS 2018)*

2

Hardware functionality needed for artificial chemical life

2.1 Introduction

This chapter is the first step towards building a robotic system capable of performing artificial chemical life experiments. In section 1.7.1, we described that one of the requirements of such a platform is performing routine liquid handling experiments. In section 1.7.2, we explained another requirement is performing feed back based experiments. This chapters addresses these two requirements.

In this chapter, we introduce EvoBot, a robot developed to perform liquid handling experiments. We commence this chapter with a physical description of the robot. We will introduce the modular design of EvoBot for liquid handling, and how it addresses the routine liquid handling functionality requirement. We then

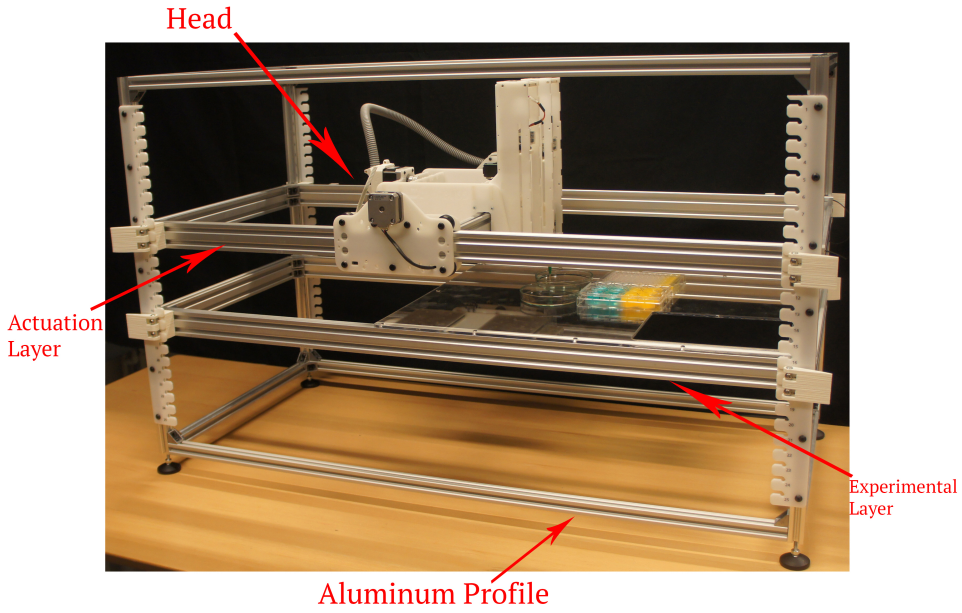


Figure 2.2.1: Overview of EvoBot.

report on the performance. We explain how a sensing layer for EvoBot provides us with the hardware functionality needed for feedback based experiments in artificial chemical life. This sensing layer provides the required infrastructure to address the requirement for feedback based experiments.

2.2 Robot Implementation

In this section we provide an overview of the mechanics, electronics, and performance of EvoBot.

2.2.1 Mechanics

EvoBot consists of an actuation layer on top, an experimental layer in the middle, and a sensing layer at the bottom. Figure 2.2.1 shows an overview of EvoBot, and Figure 2.2.2 shows an overview of the actuation and experimental layers, as well as

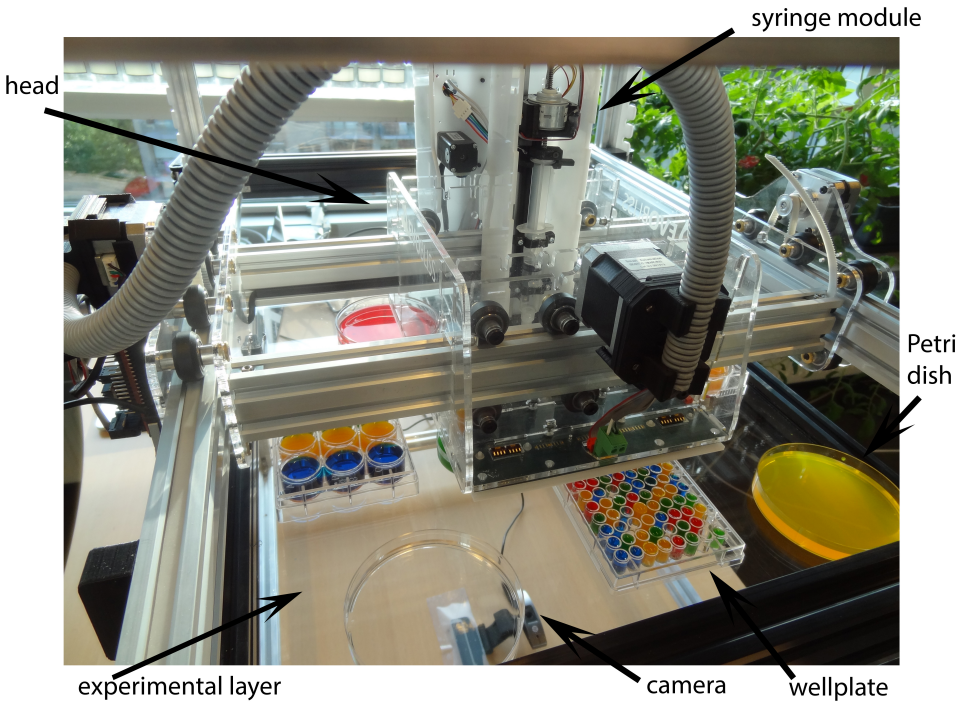


Figure 2.2.2: Overview of the actuation, experimental, and sensing layers. EvoBot's actuation layer consists of a moving head on which various modules, such as syringe modules can be mounted. The experimental layer accommodates different vessels, and the camera at the bottom acts as the sensing layer by collecting experiment data.

the head, syringe modules, and the camera.

The actuation layer comprises the robot head and modules mounted on it. The modules are plugged into the head, and are usually designed to perform an action on the experiments. However they may have sensor functionality, e.g. OCT (optical coherence tomography) scanning, an imaging technique which allows for optical sectioning of the sample. The head which holds the modules can be moved in the horizontal plane. EvoBot's modularity allows for support of modules of different kinds for various applications. The experiment-dependent modules could entail syringe modules for liquid dispensing or aspirating, grippers to move the containers over the experimental layer or dispose dirty containers, an OCT scanner module to perform OCT scans, an extruder module to 3D print, and other potential experiment-specific tools.

The experimental layer consists of a transparent Poly methyl methacrylate (PMMA) sheet on which reaction vessels are positioned. Various types of reaction vessels can be placed on the experimental layer such as, Petri dishes, well plates, beakers, volumetric flasks, graduated cylinders, Erlenmeyer flasks, and so on. The actuation layer interacts with the experimental layer by filling or emptying a specific volume to/from a syringe, washing a syringe, disposing dirty containers.

The sensing layer consists of a camera below the experimental layer to monitor the experiment, or required sensors depending on the experiment. The sensing layer collects data from the experiment, and provides feedback for the robot to interact with the experiment.

The robot frame is built from Aluminium profiles, and the experimental layer and actuation layer are mounted on it. The layers can be easily moved up or down on the robot frame with a cam lever mechanism. The frame size is 600x400x600 mm. However, it can be extended based on experiment specific requirement.

We have built different kinds of modules to be used on EvoBot's head as can be seen in Figure 2.2.3. Figure 2.2.3 (a) shows two different types of syringe modules on the left. A syringe module can move in the z direction and a stepper motor moves a plunger for aspiration and dispensing. A pump-based dispensing module can be seen on the right in Figure 2.2.3 (a). The pump-based dispensing module

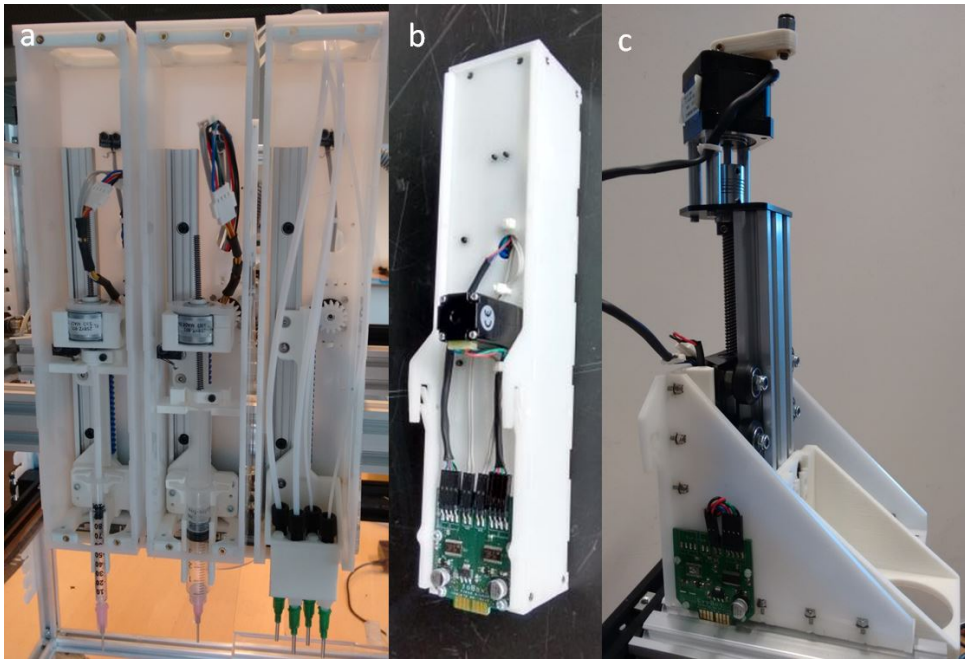
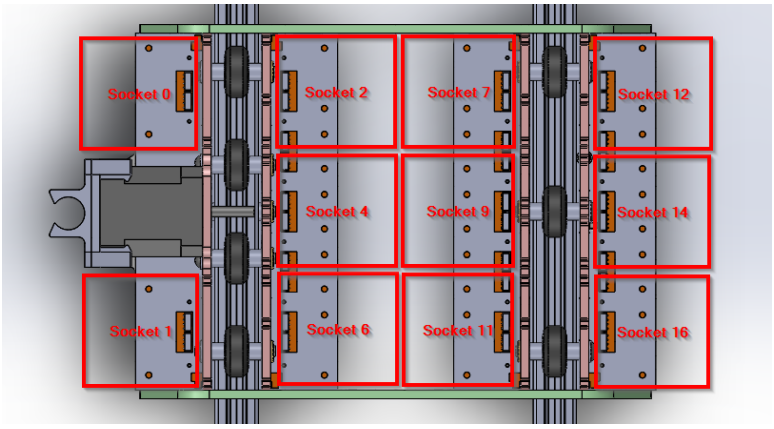


Figure 2.2.3: Modules. (a) two different types of syringe modules on the left and a pump-based dispensing module on the right. (b) the back of a syringe module. (c) heavy payload module to hold an OCT scanner in order to perform OCT scans.

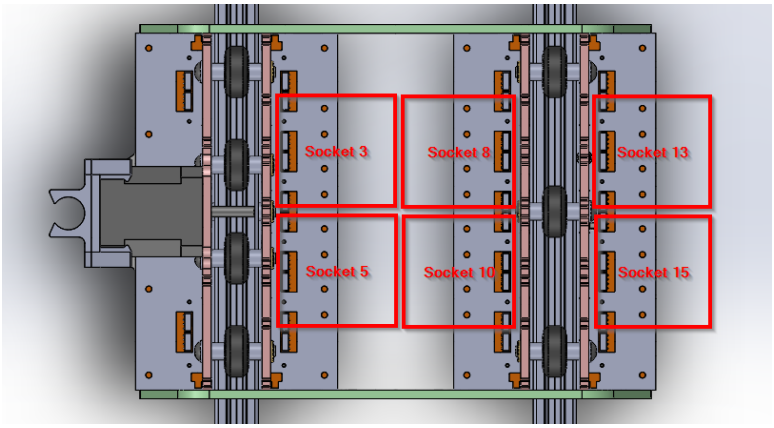
can also move in the z direction, but a pump is used for dispensing. Figure 2.2.3 (b) show the back of a syringe module, and Figure 2.2.3 (c) show a heavy payload module, which can hold an OCT scanner to perform OCT scans.

The robot head can accommodate syringe modules to aspirate or dispense liquid at 17 potential positions, as shown in Figure 2.2.4. Up to 11 syringes can be used simultaneously as the socket positions overlap with adjacent ones, as shown in Figure 2.2.4 (a), and Figure 2.2.4 (b).

Configuring EvoBot for different experiments is easy, as different types of modules can be easily removed or plugged at the appropriate position. The head is responsible for moving the modules in the x-y plane, while the modules have motors to move vertically. Figure 2.2.5 shows two different syringe modules mounted on



(a) 11 possible syringe modules positions on robot head.



(b) 6 more possible middle syringe modules positions on robot head.

Figure 2.2.4: Possible syringe modules positions on robot head.

sockets 12 and 16.

2.2.2 Electronics

EvoBot's design is based on open-source 3d printers. We use an Arduino board and a RAMPS shield, as can be seen in Figure 2.2.6. This electronics design allows us to build on existing software for open-source 3D printers. Figure 2.2.7 shows how the electronics of the robot communicate together. The computer receives

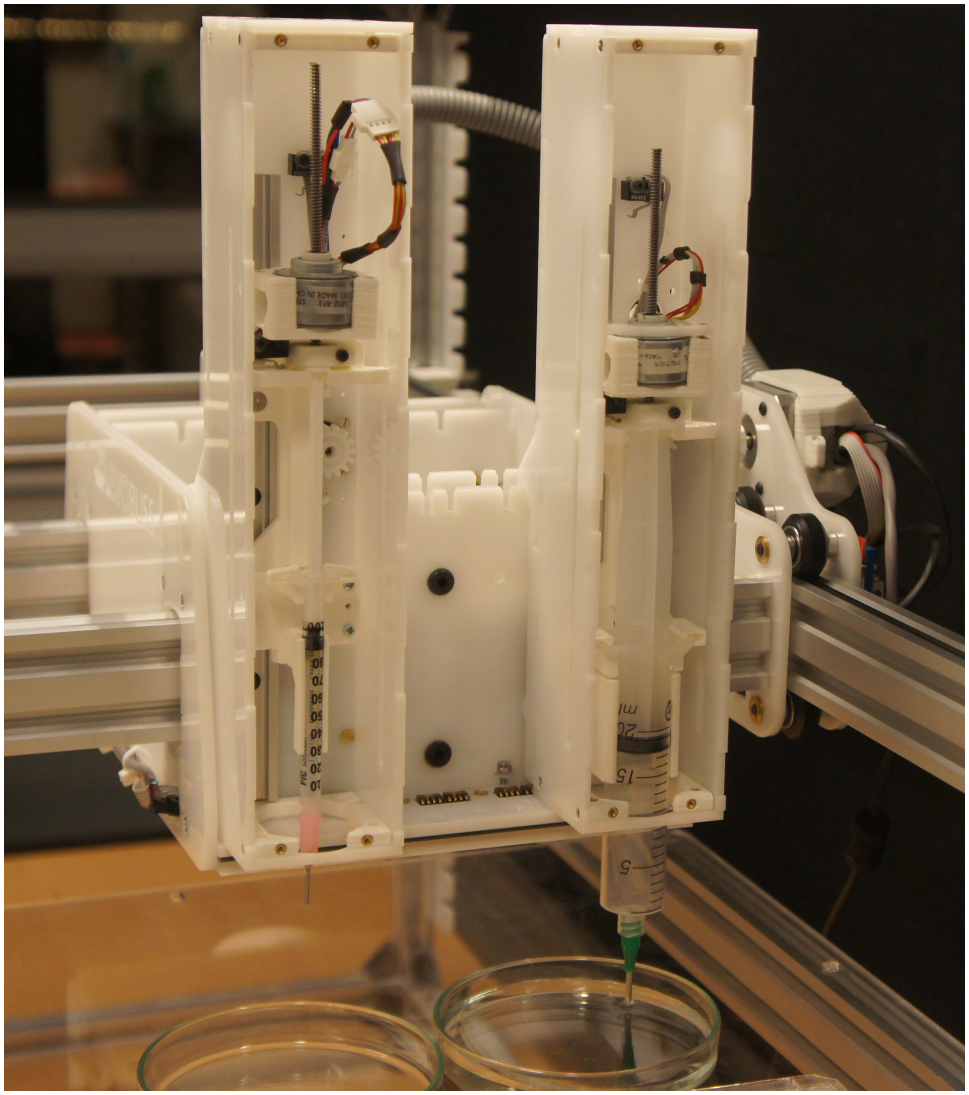


Figure 2.2.5: 1ml (Left), 5ml (right) syringe modules mounted on robot head.

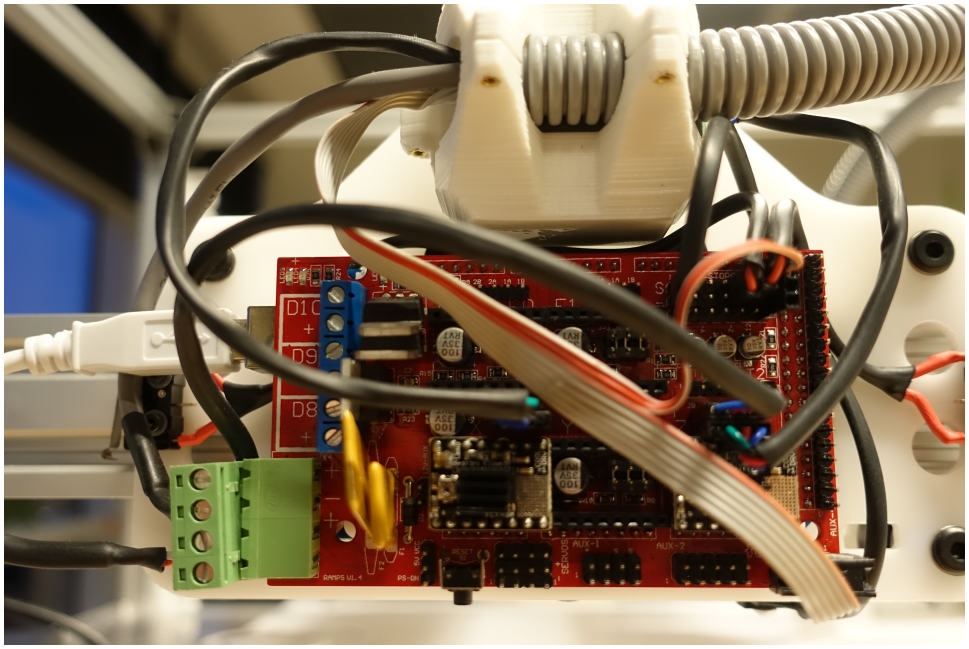


Figure 2.2.6: An Arduino board and a RAMPS shield.

image frames from the camera. On the other hand, the computer communicates with the Arduino through serial communication. The Arduino is connected to the RAMPS shield. The RAMPS shield controls the three stepper motors to move the robot head. The RAMPS shield is also connected to two large PCBs (Printed Circuit Board) on the robot head. On the other hand, each module contains a small PCB (printed circuit board) with the electronics to control the actuators and sensors of the module. When a module is plugged into the robot head, the module PCB makes an electrical connection with the spring connectors on the head PCB. Therefore the module is provided with power and an SPI bus through this spring connector. Owing to this electronics design, the Arduino can detect if a socket is populated, the type of module , and can control the module by sending SPI commands. Furthermore, if new types of modules are designed, they can be easily controlled.

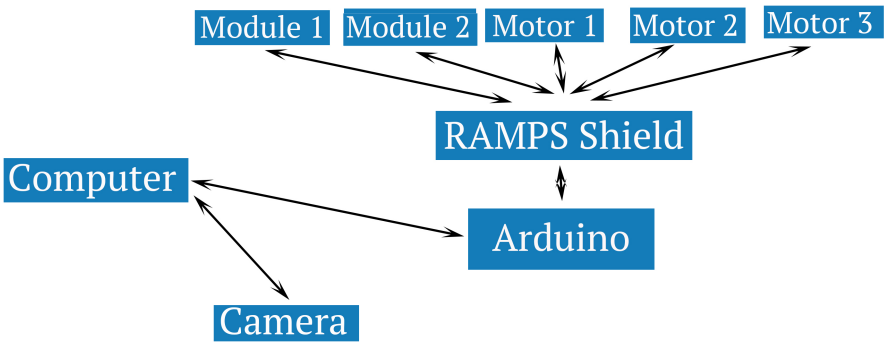


Figure 2.2.7: Communication of robot parts.

2.2.3 Precision

In this section we describe EvoBot's precision for speed, positioning, and liquid handling tasks, providing interested users with insights whether these specifications suit their requirements.

Speed

An important metric for defining the performance of a robot for liquid handling is robot speed. Therefore, we tested the maximum speed of the robot for each axis. In order to test the performance, we increased the speed of the robot on each axis to the point where the robot was missing steps. Therefore the maximum speed of the robot corresponds to the speed the robot is not missing any steps. Table 2.2.1 reports the results of these tests.

Positioning Performance

In order to test the positioning performance, we modified a syring module to hold a test probe. This probe would touch a dial indicator with a 0.01mm accuracy. To perform this test, we first homed the robot, and the dial indicator was placed in contact with the probe and set to zero. Then we moved the robot to 1, 2, 10, 20

Table 2.2.1: Max speed and accelerations of the robot.

	X	Y	Z (Syringe)	Plunger
Max Speed (mm/s)	180	180	235	8
Acceleration (mm/s^2)	3000	3000	235	4

Table 2.2.2: Positioning accuracy for each axis of the robot.

	0mm	1mm	2mm	10mm	20mm
X (mm)	0.04	0.03	-0.04	-0.11	-0.17
Y (mm)	0.01	-0.05	-0.08	-0.14	-0.18
Z (mm)	0.10	0.02	0.09	0.28	0.35

and 0 mm positions respectively, measuring the real position of the probe. We repeated this experiment 30 time to measure the accuracy. Accuracy is calculated by as $A = \bar{q} - q_t$ formula, where q_t is the target position and $\bar{q}$ is the average of n measured coordinates. Table 2.2.2 shows the results.

Liquid Handling Performance

In order to measure the precision of the robot for handling droplets, we performed experiments with various droplet volumes, and needle diameters. To measure the precision, we compared the actual volume of a dispensed distilled water droplet with the intended droplet volume. The actual droplet volume needed to be calculated from the density equation. For this equation, we needed to know the mass of the droplet. We measured the weight (hence the mass) of the droplet with a $\pm$0.003g precision scale. Four experiments were performed with a professional

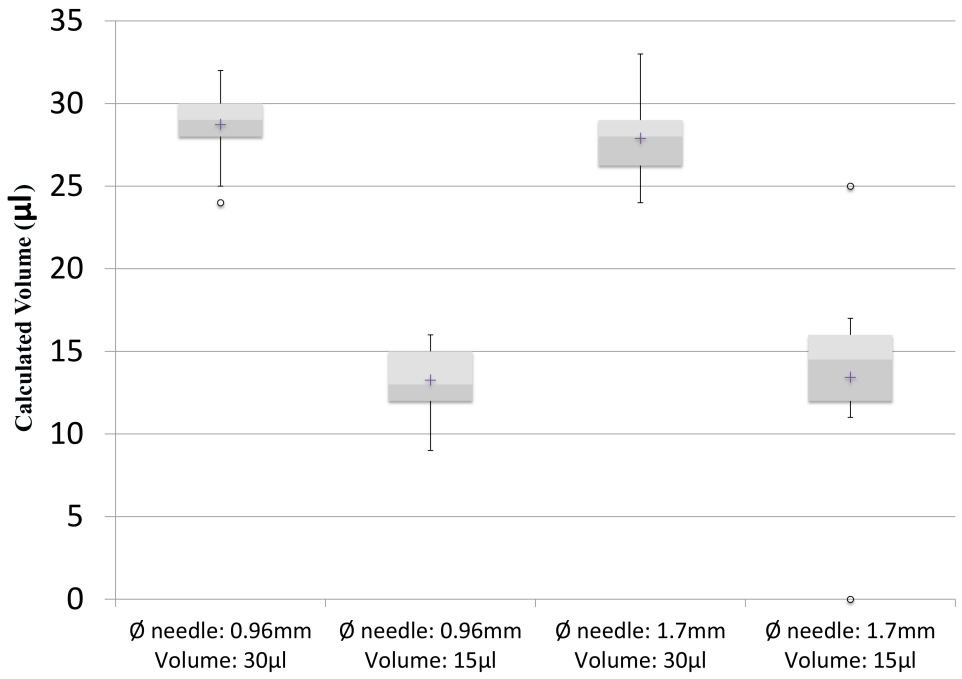


Figure 2.2.8: Boxplot comparing experiments with 100μl syringe. The whiskers represent the lowest and highest datum still within 1.5 IQR (interquartile range) of the lower quartile or of the upper quartile, respectively.

100 μl syringe (Hamilton 710 LT) to handle liquid volumes of 15 μl and 30μl, with needle internal diameters of 0.96mm and 1.7mm. In order to obtain better performance, we performed experiments with an air gap, i.e. aspirating air before water, to minimize the the effect of surface tension. The results of the experiments can be seen in Figure 2.2.8.

2.2.4 Building a robot

EvoBot is based on open-source 3d printer technology. The open-source nature reduces the cost of the robot and makes it possible to design a robot for specific experiments. Interested users can build the robot as it takes advantage of open-source 3d printer community and open-source electronics. They can also build already developed modules as they are available as open source.

2.3 Conclusion

In this chapter, we described the implementation of EvoBot, a liquid handling robot, and how it addresses the requirement for basic liquid handling functionality. EvoBot is designed in 3 layers. An experimental arena, to hold microscope slides, Petri dishes or other reaction vessels, and on top of the experimental layer the robot head, accommodating up to 11 syringe modules to aspirate or dispense liquid. The sensor layer is under the experimental layer where the camera is placed. This sensing layer provides us with the required hardware infrastructure to enable us perform artificial chemical life experiments. In the following chapter, we will explain what software functionality is required to make this platform capable of performing feedback based experiments.

3

Required Software for Artificial Chemical Life Experiments

3.1 Introduction

This chapter is the second step towards building a robotic system capable of performing artificial chemical life experiments. In section 1.7.2, we described that one of the requirements of such a platform is performing feed back based experiments. In section 1.7.3, we explained that another requirement of this platform is interacting with experiments while happening. In chapter 2, we described the hardware of the required infrastructure to address the needs of feedback based experiments. In this chapter, we describe the software functionality required for a robotic platform capable of feedback based experiments in artificial chemical life experiments introduced in section 1.2, and also able to interact with the experiments in real-

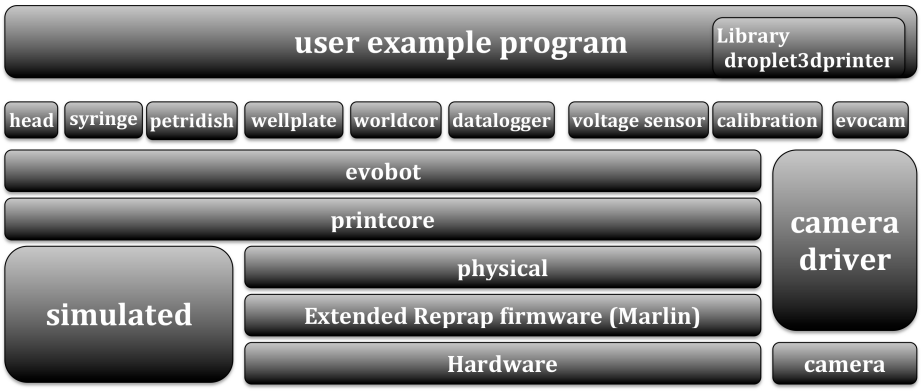


Figure 3.2.1: Software architecture of EvoBot. The API developed for EvoBot provides the basis for performing a large class of experiments.

time. Addressing these software requirements allows us to use a liquid handling robot, introduced in chapter 2, for artificial chemical life experiments.

In this chapter, we describe how the software for this liquid handling robot, EvoBot, is developed to meet the different needs of artificial chemical life experiments. These needs include detecting changes in the experiment condition, based on sensor input, and interacting with the experiment based on the observed changes. We also introduce the software architecture including firmware, vision system, camera calibration, coordinate system transformation, logging experiment data, customization for different robots, graphical user interface for interactive experiments, experiment templates, and the robot's wiki.

3.2 Software Architecture

The software for EvoBot has been developed in Python, and has been tested on OS X, Linux, and Windows. Figure 3.2.1 shows the software architecture of EvoBot. At the top layer the user programs and runs the experiment. Depending on the command, an API module like the syringe module will be responsible for sending the corresponding G-code (or G programming language used in computer-aided

Function
syringe1.fillVolFrom(head, ml=2, container=petridish1)
syringe2.emptyVolTo(head, ml='all', container =petridish2)
syringe1.syringeWash(head, times=3, offset=10, container=petridish1)
syringe2.goToXY(head, point=(100,100), worldcor=worldcorSyringe1)
syringe1.isEmpty()
syringe2.isFull()
syringe1.canDispenseVol(vol=1)
syringe2.canAbsorbVol(vol=3)
head.getX(), head.getY()
syringe2.getXY(head, worldcor=worldcorSyringe1)
wellplate1.fillWell(syringe1,wellName='h12', dispenseVol=2, dispensePos=40,upPos=38)
wellplate2.emptyWell(syringe2,wellName='e', dispenseVol=3,dispensePos=40,upPos=38)
wellplate1.fillAll(syringe1, petridish1, dispenseVol=2, dispensePos=45,upPos=40)
wellplate2.emptyAll(syringe2, petridish2, dispenseVol=3, dispensePos=45,upPos=40)
wellplate1.transferToWellplate(syringe1, wellplate2, absorbVol=2, absorbPos=40,upPos=38)
droplet3dprinter.startPrint(printString= 'R E A L ',printPoint=(130,0),dimensions= (80,320),dispenseVol=0.05 , minDistancemm=3, goalPos=-50, upPos=-48)

Figure 3.2.2: Sample functions provided by EvoBot's API.

manufacturing to control automated machine tools) to the next layer down. The syringe module, the Petri dish, the well plate, and the head have methods for their own behavior or their interaction with other modules. The bottom layer, EvoBot represents the robot as a whole. It is mainly used to initialize the robot, turn it on or off. One layer down, the open-source library "printcore" is responsible for checking errors. The physical layer is responsible for configuring a serial connection on a specific port and baud rate, as well as reading to and writing from the serial port. On the other hand, the simulation module enables the user to verify his program in simulation mode prior to running it on the robot.

The API developed for EvoBot provides the basis for performing a large class of experiments. Figure 3.2.2 shows some of the functions provided by EvoBot's API enabling it to perform various chemical experiments.

3.2.1 Firmware

EvoBot uses an extended version of the Marlin firmware to add support for controlling syringe modules. The robot's firmware resides on an Arduino Mega, and

is the link between software and hardware, interpreting commands from the host computer in G-code and controlling the motion accordingly.

3.2.2 Reactive vision system

EvoBot's vision API is responsible for processing the camera frames from the experimental layer to extract data about the experiment, e.g. droplet behaviors. We use OpenCV3, NumPy, Matplotlib, and Webcolors libraries for image data analysis. In order to recognize droplets in artificial chemical life experiments, and analyze their behavior we use a series of image processing operations, including different filters. We smooth the image to remove noise, and convert the color space of camera frames from BGR to HSV for better accuracy, and robustness to lighting changes. We then threshold the HSV image to extract only the droplets. We apply morphological operators, namely closing, i.e. dilation of image frames followed by erosion, and then opening, i.e. erosion of the resulting frame followed by a dilation, to remove noise, and recognize droplets more accurately. The next step is to find the contours in the binary image to detect droplets. Then the moments are calculated to determine the center of droplets. The center coordinate is used to calculate other droplet information, e.g. speed, direction, or acceleration. We use Hough transform to detect Petri dishes.

In order to measure the accuracy of the vision system, we needed to compare the actual size of a droplet, with the size obtained from the vision system. As it was difficult to measure the actual area of a droplet precisely, we laser cut droplets from Poly methyl methacrylate (PMMA) at different sizes, namely 5 mm^2, 10 mm^2, and 15 mm^2. Then we placed the laser cut droplets on the experimental layer to be detected by the vision, and consequently had their size calculated. The accuracy of the vision system was tested with the droplet placed at different locations on the experimental layer, and when the experimental layer was placed at different distances to the camera. Our experiments determined the accuracy of the vision system to be 4% of droplet area.

It should be noted that the vision system may have constraints in some work-

ing conditions. The vision system of our robot detects droplets based on color segmentation. Therefore a white background should be used on top of the robot's frame, so that external objects will be filtered. There are also certain lighting conditions for the vision system that need to be met. There should be sufficient ambient light to make the droplets detectable. Therefore if the robot needs to be operated under low lighting conditions, e.g. at night, an external source of light should be used.

3.2.3 Camera Calibration

In order to extract relevant experimental data, EvoBot's Vision API needs to calibrate the coordinate systems of the robot and the camera. To this end, we utilize Affine Transformation that represents a relation between two images, and can be used to express rotations, translations, and scale operations. We need five coefficients to calculate the Affine transformation matrix in images obtained from the camera. This is done by asking the user to click on the needle tip at three different positions to provide six equations for calculating the Affine transform coefficients. This step is only done once, when the robot is setup.

Having performed the camera calibration, we need to verify the accuracy of calibration. To this end, after the user has clicked on three predefined positions, i.e. calibration is finished, the phase to verify the accuracy begins. In the verification phase, when a user clicks on an arbitrary point on the image, the syringe needle tip should move to that location. The clicked point on the image is small, and therefore the needle tip moving to the same points provides an acceptable accuracy. Through tests at different positions on the experimental layer, and at different distances of the experimental layer and the camera, the calibration has shown to be reliable. It should be noted that we also implemented the calibration with perspective transformation. However, the performance was not as accurate. Therefore the method used for the vision system of EvoBot is affine transform.

3.2.4 World coordinate system for Multi syringe experiments

The world coordinate module transforms coordinate systems of different modules to each other, enabling moving different modules to the same location. Syringe modules can be placed at 17 different positions on EvoBot's head, and up to 11 syringes can be mounted on EvoBot simultaneously. An object, such as a Petri dish, or a well plate can be defined relative to any of the 17 coordinate systems, and its coordinates will be calculated automatically relative to other syringes, enabling precise access to the same object by any desired syringe module. Even if using multiple syringes, only one of the syringes needs to be calibrated.

3.2.5 Logging experiment data

We have equipped EvoBot with a data logger which is able to log different types of experiment data, including head x and y position, head speed and acceleration, syringe x, y and z position, and plunger vertical position over time. The data logger can also log events, such as aspirating or dispensing with a syringe, washing a syringe, various experiment steps and so on. The user can save the log either in "csv" file-format or a simple text format. The "csv" format enables the users to analyze this data easily, as well as providing compatible data for advanced data mining applications with software such as Weka or python-based Orange. Also sophisticated plots of real time experiment results can be depicted using Matplotlib library. In addition, EvoBot's vision API enables users to record the video of experiments when a desired event occurs.

Taking advantage of the log module in EvoBot API, we obtain useful information weather the experiment was performed successfully, and in accordance with the profile defined in the firmware. Figure 3.2.3 shows the head position for 1-N experiments. 1-N refers to experiments like aspirating liquid from one chemical vessel, and dispensing in different chemical vessels, e.g. filling all wells in a well plate with a specific liquid from a Petri dish. The head position provides an overview of the experiment, and can be used to verify if the experiment has been

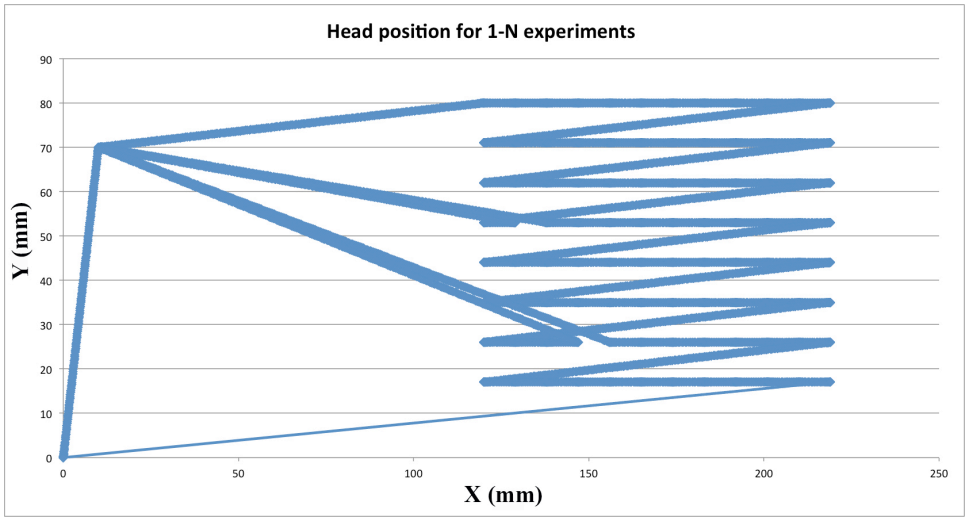


Figure 3.2.3: Head position for experiment verification in 1-N experiments.

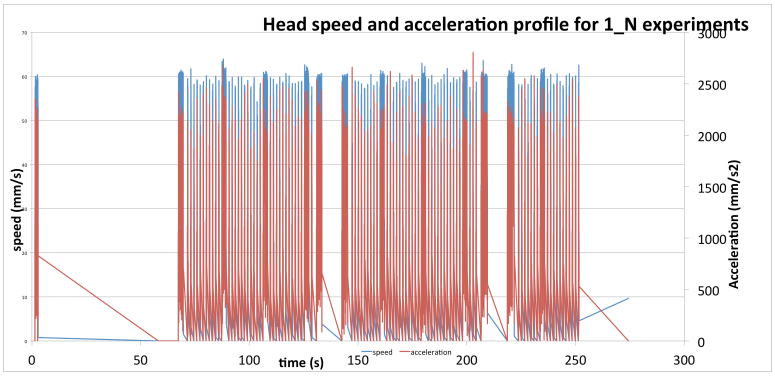
performed correctly. In case of errors it is useful for fast debugging.

It should be noted that the trail is generated by the motion controller. Hence, it is unknown if the robot actually moved this trail, as we don't have any sensors to detect this. Hence, the program can be verified, but not the working of the actual robot.

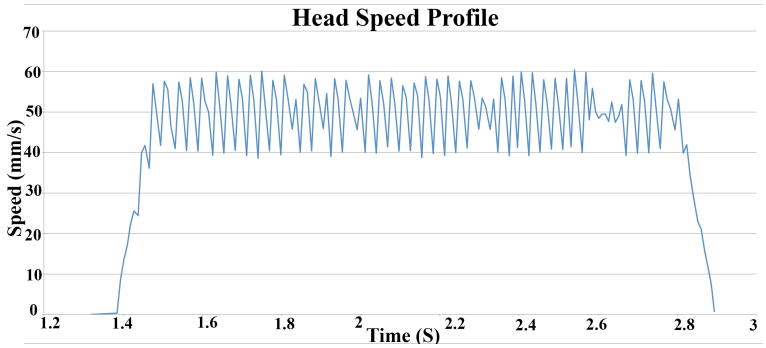
Figure 3.2.4 (a) shows the speed and acceleration for the head during the experiment. An overview of this diagram can help optimising the experiments, as well as highlighting errors in the experiments, and detecting unusual behaviors.

In addition, we can look at this diagram data in more detail to verify if the microscopic behaviors set in firmware are satisfied. Figure 3.2.4 (b) shows the speed profile for a movement over a straight line. The speed profile set in the firmware is accelerating at the start of a movement, keeping the speed constant during the movement, and decelerating at the end of the movement as shown in Figure 3.2.4 (b).

Figure 3.2.5 shows the syringe and plunger speed, and acceleration for 1-N experiments. This data can also be beneficial for verifying the experiment or high-



(a) Head speed and acceleration profile for experiment optimization and verification in 1-N experiments.



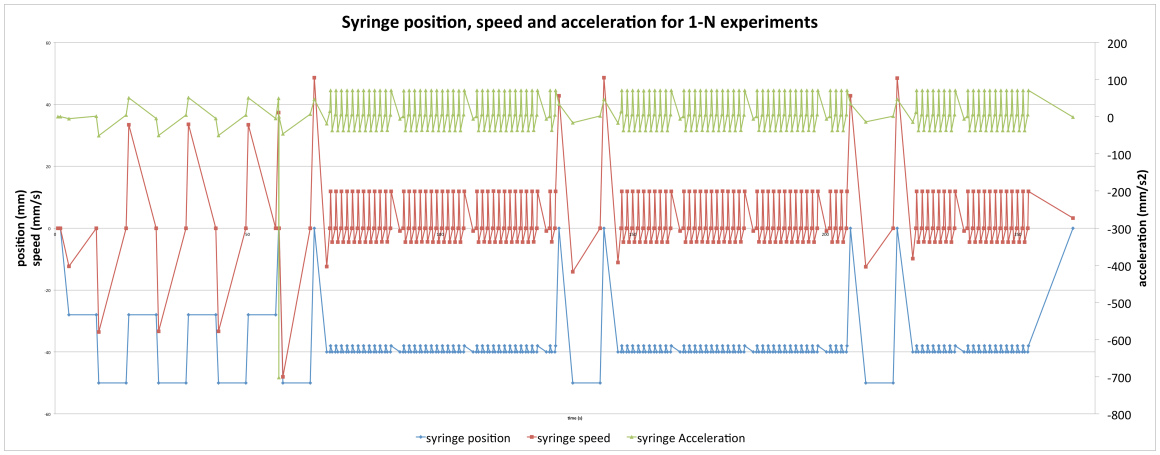
(b) Head speed profile for movement over a straight line as set in firmware.

Figure 3.2.4: Head speed and acceleration profile for experiment verification in 1-N experiments.

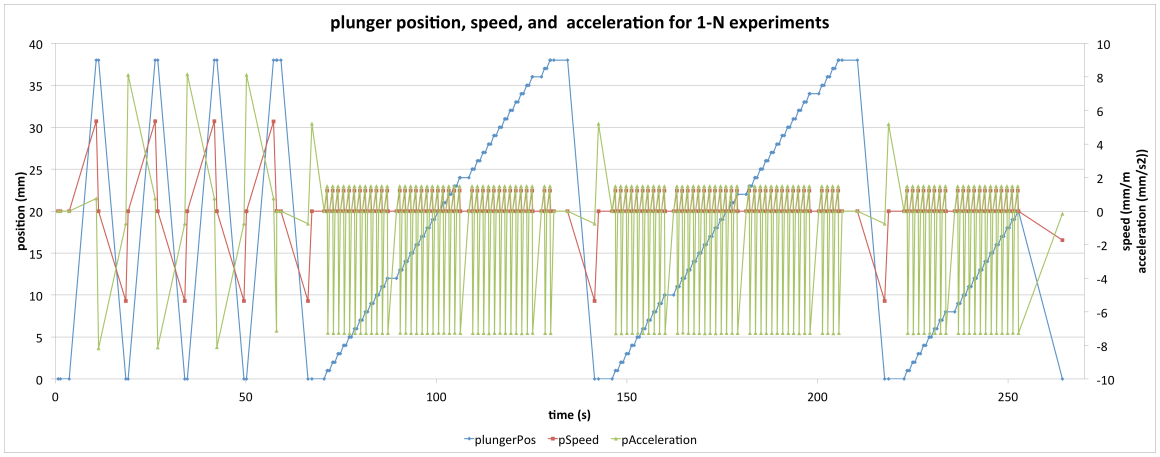
lighting inconsistencies in experiment pattern.

3.2.6 Customizing robots

EvoBot comes with a default configuration file, and each user has his own local configuration on top of that, enabling users to use the same software independent of the specific parameters of the robot they use. We use this configuration structure as we have built seven copies of EvoBot with different dimensions, and modules for different users according to their needs. The software detects the type, serial port and slot number for all modules mounted on the head. In addition, the users use different versions, size, and number of syringe modules based on the types of



(a) Syringe position, speed and acceleration profile for 1-N experiments.



(b) Plunger position, speed and acceleration profile for 1-N experiments.

Figure 3.2.5: Syringe and plunger position, speed and acceleration profile for experiment verification 1-N experiments.

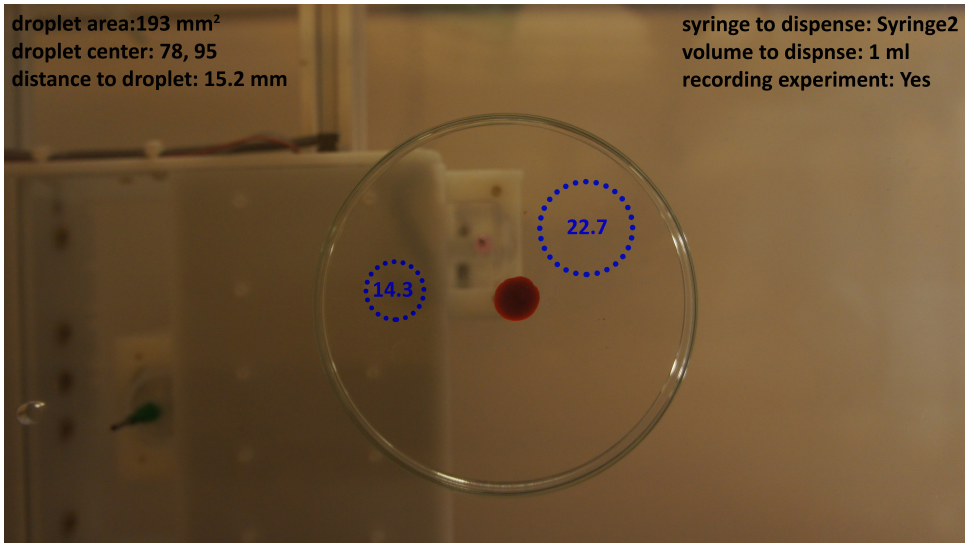


Figure 3.2.6: EvoBot's interactive graphical user interface. The user interface displays experiment information live, and enables users to interact with the experiment real time. The dotted circles show the estimated propagation of injected salt droplets. The elapsed time since dispensing is displayed in the center of the circle..

experiments they perform. Furthermore, EvoBot may be connected to different serial ports on different computers. The camera is also assigned different ids when connected to different ports or computers.

3.2.7 INTERACTIVE GRAPHICAL USER INTERFACE

As can be seen in Figure 3.2.6, the EvoBot software provides an interactive graphical user interface, which enables chemists to interact with the experiment. The users are provided with a live video of the experiment, showing real time data about the experiment including droplet data, such as size or speed, time elapsed since dispense. Based on this data, the chemist can interact with the experiment. The user will select one of the syringes, specify the volume either to aspirate or dispense liquid, and click on the point he/she wants to perform the interaction. Data about the interaction, e.g. time elapsed from interaction, will also be displayed.

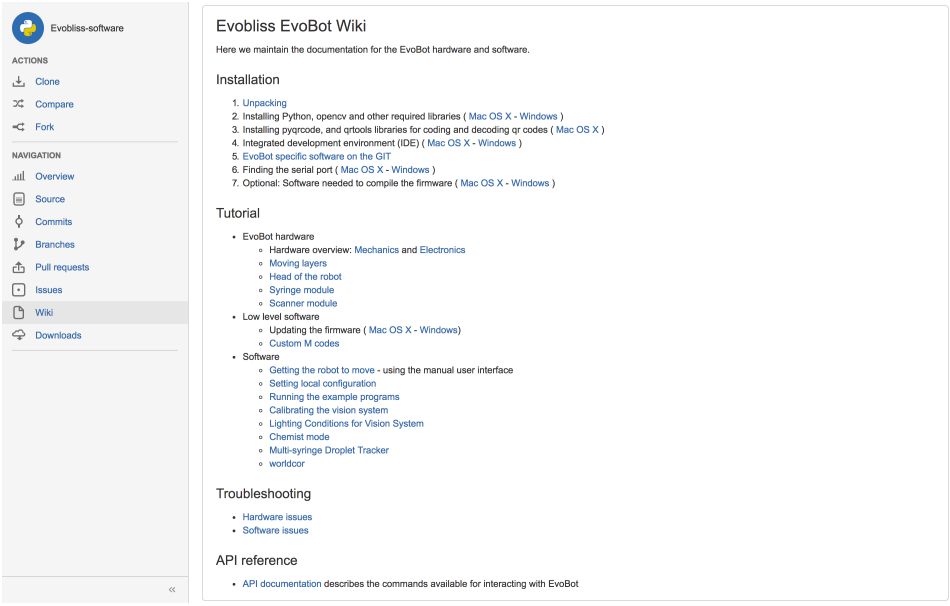


Figure 3.2.7: EvoBot's wiki.

3.2.8 Experiment templates

We have provided numerous examples in the EvoBot's codebase [21], demonstrating the use of the API. Furthermore, since EvoBot is an open source system other users are able to develop software for any potential modules they develop.

3.2.9 EvoBot's Wiki

We maintain the documentation for the EvoBot hardware and software in EvoBot's wiki, as can be seen in Figure 3.2.7. The wiki includes information about installing different required software packages, low level software, e.g. updating the firmware, and custom M codes, setting local configuration, running the example programs, calibrating the vision system, lighting conditions for vision system, multi-syringe droplet tracking, world coordinate system, trouble shooting hardware, and software, and API documentation.

3.3 Conclusion

In chapter 2, we described the required hardware for a robotic system capable of feedback based experiments. In this chapter, we describe how we address software requirements for such a platform. In this chapter, we demonstrate the software implementation in order to make a liquid handling robot capable of artificial life experiments, which require feedback from the experiment. This software implementation provides the data obtained from the vision system as feedback to the experiment. Different software modules enable required interaction with the experiment. The different functionalists this software architecture provides empower us to perform a wide class of artificial chemical life experiments. In the following chapter, we demonstrate a range of possible feedback based experiments owing to the developed hardware, and software. We will discuss how EvoBot has realized experiments that formerly were difficult to perform.

4

Interactive experiments

4.1 Introduction

In chapter 2, and chapter 3, we described the hardware, and software functionality required for a robotic system capable of performing feedback based experiments in the field of artificial chemical life. Having defined the requirements, and implemented the robotic system, it is required to verify the the applicability of developed platform to liquid handling experiments. To this end, the capability to perform both routine liquid handling experiments, and various artificial chemical life experiments should be tested. This chapter addresses the need for verification of the system by applying the robot to non reactive experiments, as well as different artificial chemical life experiments.

We commence this chapter by verifying the general non-reactive liquid handling functionality of the robot by performing the routine liquid handling experiments

with the robot. Then we provide examples of precise droplet placement. These functionlities are often required in liquid handling experiments as well as artificial life experiments. In the next step, we investigate the reactive functionality of the robot. EvoBot has been used for performing numerous reactive liquid handling experiments. We will include three use cases, as droplet behaviors in these experiments and the required type of robot interaction are comprehensive enough to be generalized to various reactive experiments. The use cases are examples of what is possible with EvoBot, and are functionalities requested by our partners in the EVOBLISS EU project to enable them to perform a class of experiments. We compare the way non-automated experiments were performed without the robot for demonstrating the ease, accuracy and precision of results obtained by the robot. Furthermore, easy configurability of the robot for "needed to be modified" experiments is examined. These experiments are quickly designed to investigate the effect of a parameter on the experiment. Based on these pilot experiments, an experiment with desired behavior can be designed. Finally, we will discuss how EvoBot has realized experiments that formerly were difficult to perform.

In this chapter, we describe how we have made a liquid handling robot usable for artificial chemical life experiments by addressing hardware, and software requirements explained in chapters 2 and 3.

4.2 General Liquid Handling Experiments

EvoBot can do general liquid handling experiments as it can perform 1-N, N-1, and N-N experiments, mix the contents of a reaction vessel, and wash reaction vessels. 1-N, as shown in Figure 4.2.1, refers to experiments like aspirating liquid from one chemical vessel, and dispensing in different chemical vessels, e.g. filling all wells in a well plate with a specific liquid from a Petri dish. N-1, as shown in Figure 4.2.2, refers to experiments like emptying wells of a well plate in a garbage collector. N-N, as shown in Figure 4.2.3, refers to experiments like transferring liquid from specific wells in a well plate to the corresponding wells in another well plate. Owing to the software developed, the robot is intelligent enough to fill a syringe with the same

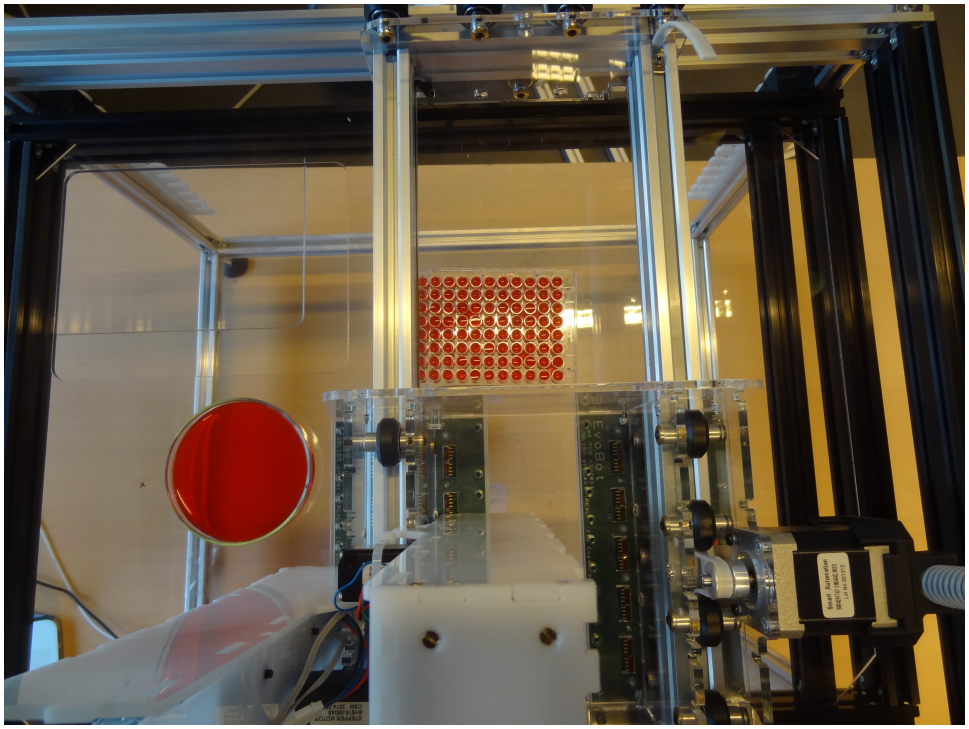


Figure 4.2.1: 1-N experiment. Filling all wells in a well plate with a specific liquid from a Petri dish.

liquid if that syringe runs out of liquid. Due to these functionalities, EvoBot can be used in application like serial dilutions, ELISA, and PCR. A serial dilution is diluting a substance in solution step by step. ELISA (enzyme-linked immunosorbent assay) [110] is a test to identify a substance. PCR (Polymerase chain reaction) [84] is a technique in molecular biology for generating copies of DNA sequence.

It should be noted that the general liquid handling functionality of the robot is not the focus of this chapter, and not as important as the reactive functionality of the robot, but provided to briefly showcase some of the possibilities with the robot.

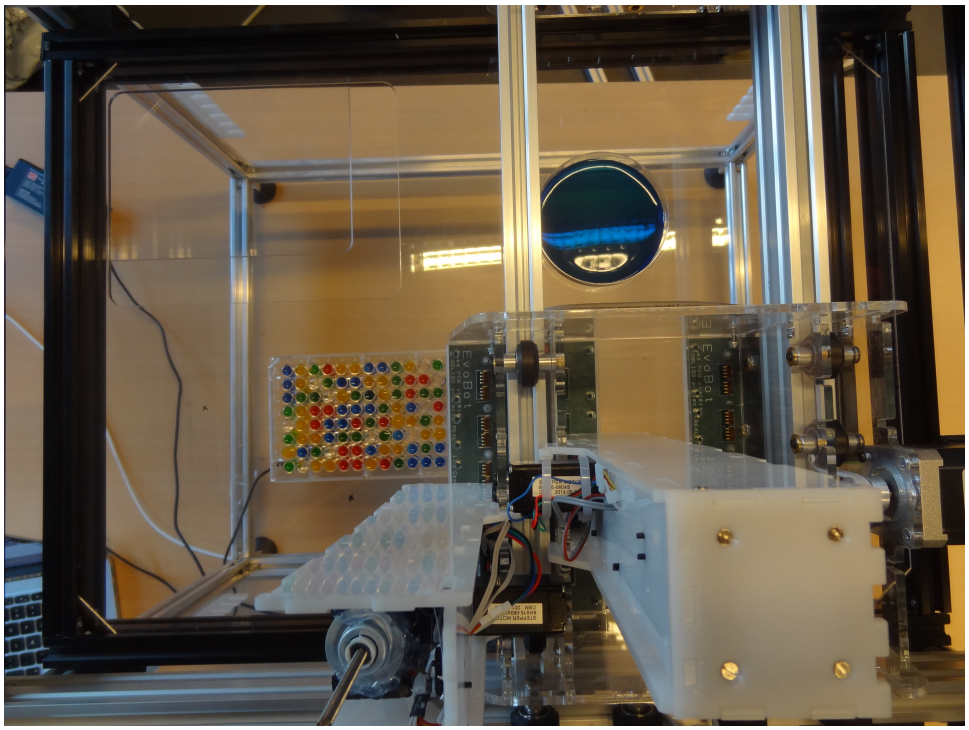


Figure 4.2.2: N-1 experiment. Emptying wells of a well plate in a garbage collector.

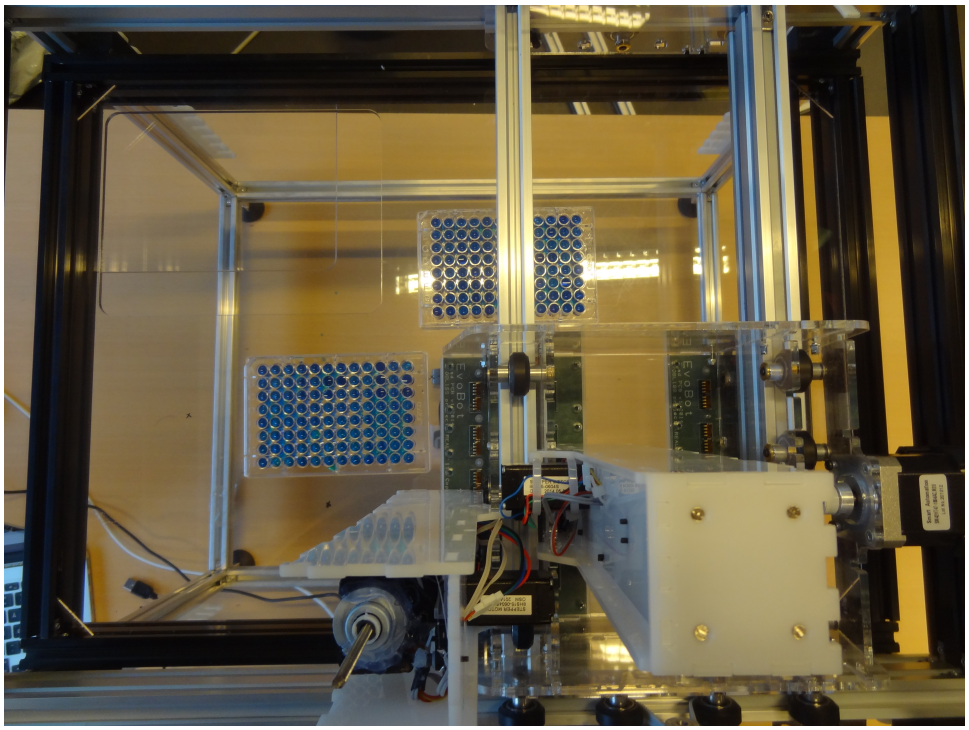


Figure 4.2.3: N-N experiment. Transferring liquid from specific wells in a well plate to the corresponding wells in another well plate.

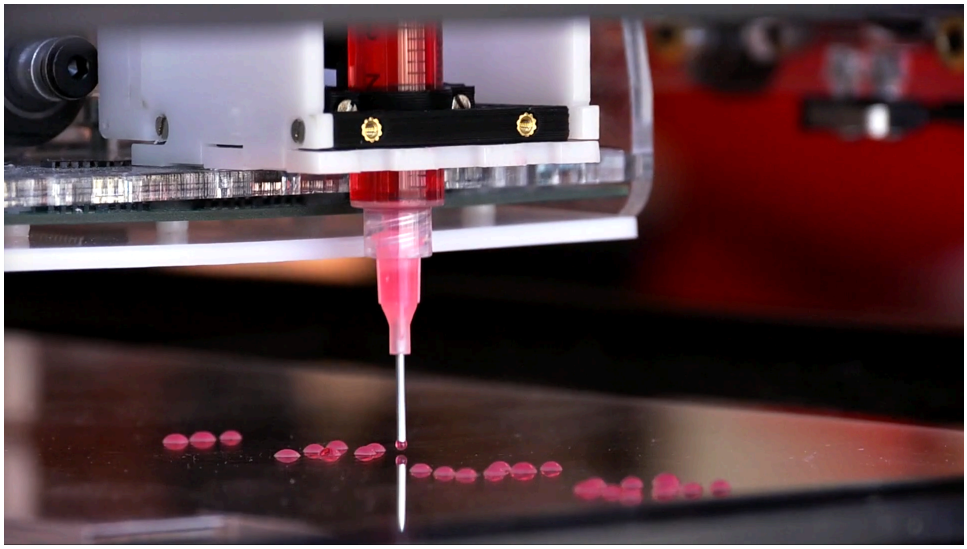


Figure 4.3.1: Droplet placement in two dimensions in controlled patterns (the word "REAL" in this experiment).

4.3 Droplet Placement

EvoBot can be used to place droplets in 2 or 3 dimensions precisely. The droplets can be placed in specific patterns such as circles or rectangles, or in sophisticated geometric shapes. Droplet pattern printing in 3D is a novel opportunity as it is difficult to achieve the necessary precision in the vertical direction if performed manually. Figure 4.3.1 shows an image of droplets, placed in two dimensions in an arbitrary pattern (the word "REAL" in this experiment). Figure 4.3.2 shows droplets placed in 3D patterns. This was obtained by injecting salt solution into silicone. The salt density was calculated in a way so the injected droplets would float in the silicone. Different droplet volumes at different positions, and heights were injected to form a 3D pattern. Then the silicone was cured in an oven for 10 minutes.

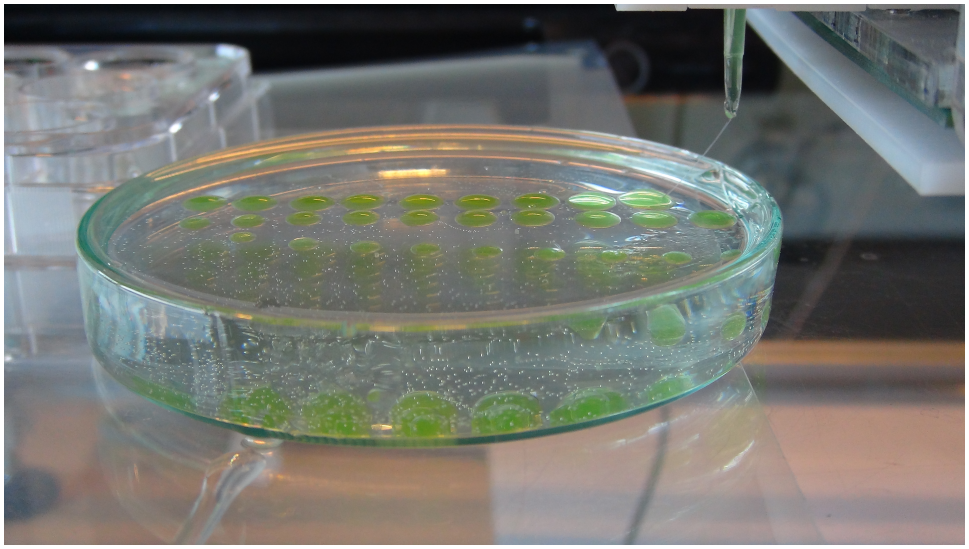


Figure 4.3.2: Droplet placement in 3D patterns. The 3D pattern was obtained by injecting salt solution into silicone, and then curing silicone in an oven.

4.4 Interactive Experiments

As discussed in chapter 1, artificial life experiments require precise timing and relative positioning of reagents with respect to motile droplets. In this type of experiments, the introduction of a reagent into the experiment will make a stationary droplet motile, or affect the motility of a dynamic droplet. Therefore the experiment parameters, such as distance between droplet and reagent, time of adding reagent, and concentration of liquids, affecting droplet behavior are investigated. The experimental parameters either influence individual droplet behavior, such as droplet speed, droplet size, and induction time, or group droplet behavior, such as droplets clustering or declustering, droplets merging or dividing.

The confluence of computer vision and robotic automation, as discussed in chapters 3 and 2, makes automation of these experiments possible. Real-time analysis of experimental images provides data about droplet properties and behavior. The preciseness of robot automation enables significant reduction of noise related to positioning of reagents and control over parameters such as fixed dispensing angel,

dispense time, dispensing force, and dispense distance to liquid surface. It is also possible to make reactive experiments because the data obtained from the computer vision system may trigger the robot to perform a specific action. Hence, the introduction of automation makes it is possible to control experimental parameters precisely and significantly reduce the noise leading to improved statistical significance of results.

In the following sections, we will discuss three use cases of EvoBot for artificial chemical life experiments. These use cases demonstrate how applying EvoBot to artificial chemical life experiments improves precision and repeatability of the experiments. Furthermore we show how experiments that were very difficult or not possible to perform are made possible using EvoBot.

4.4.1 Use case 1: droplet response as a function of distance to a reagent

In this section we automate an experiment whose purpose was to understand the response of a droplet as a function of distance to a reagent [57]. Chemists did this experiment by hand, but given they were not able to place reagents precisely they had to rely on intuition and luck to get a sufficient coverage of distances. Whether they were successful or not could first be verified after the experiments were performed by analyzing the experiment videos. In contrast, by employing computer vision driven automation it is easy to ensure systematic coverage of relevant distances and we were also successful in reducing the noise of the experiments.

Experimental Setup

An example of reactive experiments is the dynamics of chemotactic droplets in salt concentration gradients [57]. To verify the reduction in variability of reactive artificial life experiments, we used EvoBot to duplicate the experiment reported in [57], as shown in Figure 4.4.1. In this experiment, a decanol droplet is added to decanoate spread over a microscope slide. Now adding a droplet of sodium chloride will cause the decanol droplet move. We have used EvoBot to duplicate this

Figure 4.4.1: Bottom camera view of decanol droplet on microscope slide.

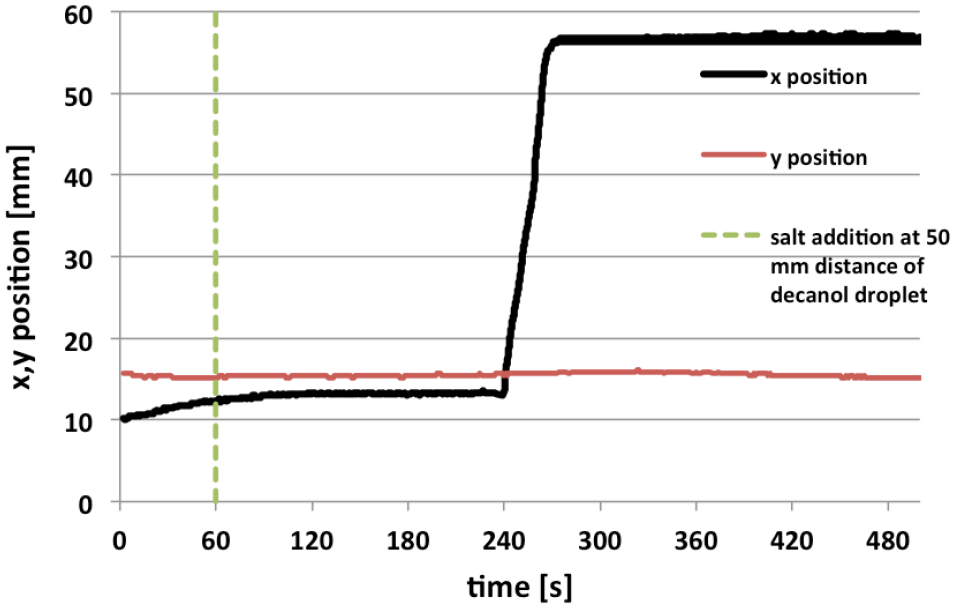
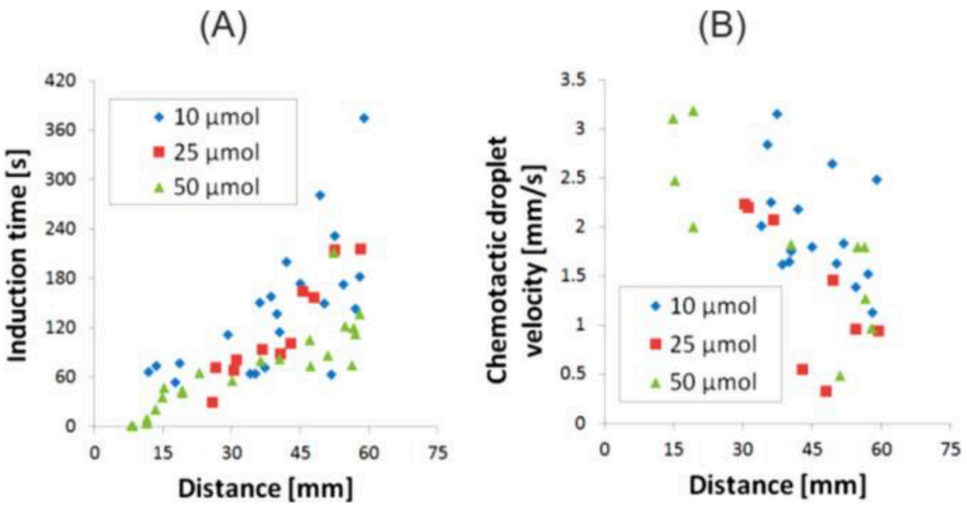


Figure 4.4.2: The x, y position of the droplet as a function of time.

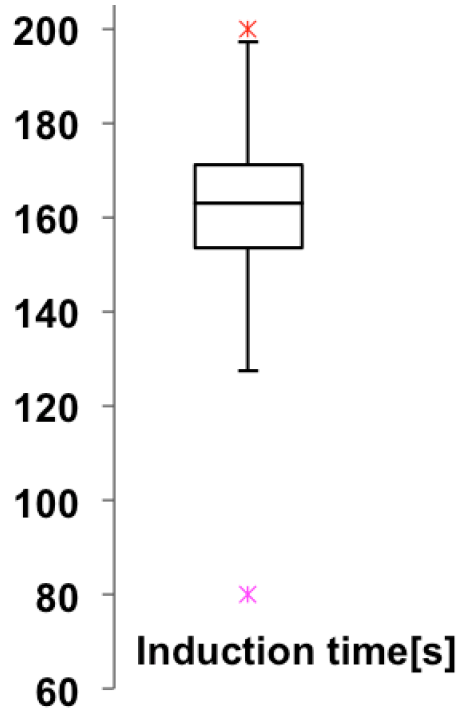
experiment by adding 1 mL of 10 mM sodium decanoate on a 76×26 mm microscope slide. The microscope slide was placed horizontal on the experimental layer of the robot, and the lower left corner of the slide was assumed to be the origin of the coordinate system. In order to have the decanoate spread on the microscope slide, first a 0.5 mL droplet of 10 mM sodium decanoate was placed at point x=10 and y = 6.5 (10 mm to the width, and 6.5 mm to the length of the slide). Then another 0.5 mL droplet of 10 mM sodium decanoate was placed at the center of the microscope slide (x=38 and y = 13). The robot syringe was perpendicular to the microscope slide and the syringe tip was 1 mm above the microscope slide when dispensing the decanoate. The stepper motor of the syringe had a speed of 4 mm per second dispensing droplets. As the next step, a 5μL droplet of decanol mixed with oil red O was dispensed at point x = 10 and y = 13 on the slide. The robot syringe was perpendicular to the microscope slide and the syringe tip was 1 mm above the microscope slide when dispensing decanol. The stepper motor of the syringe had a speed of 4 mm per second dispensing decanol. Another syringe was used to add a 10μL droplet of a 1 M NaCl solution (i.e., 10μmol of NaCl) at 50 mm distance to the decanol droplet. Similarly, the robot syringe was perpendicular to the microscope slide and the syringe tip was 1 mm above the microscope slide when dispensing Sodium Chloride. The stepper motor of the syringe had a speed of 4 mm per second dispensing the droplet. We repeated the experiment 15 times to verify the repeatability of the experiment.

Experiment Results

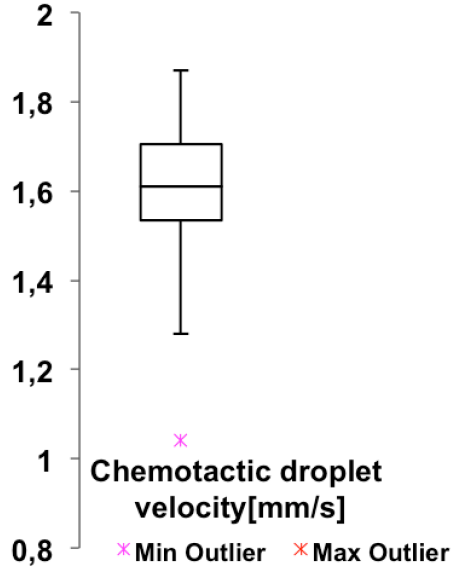
Figure 4.4.2 shows the x and y position of decanol droplet over time. Figure 4.4.3 shows the induction time of decanol droplet, i.e., the delay between NaCl addition and the start of its chemotaxis, and chemotactic droplet speed for the experiment being performed 15 times. Verifying the repeatability of the experiment with fixed parameters, such as distance between decanol and salt droplets, fixed dispensing angle, force, and time, and precise analysis of the result is only possible with automation. The results of our experiments verify the behavior of decanol droplets



(a) Experiment results performed by hand



(b) induction time



(c) chemotactic droplet speed

Figure 4.4.3: Experiment results performed by hand (top), induction time (bottom left), and chemotactic droplet speed (bottom right).

Table 4.4.1: Mean and standard deviation of experiment results

Statistics	Induction Time	Chemotactic Droplet Speed
Mean	159.60	1.28
Standard Deviation	25.89	0.51

in presence of salt concentration gradients observed by [57]. We used coefficient of variation to compare the accuracy of the experiment results when performed by hand and the robot. The coefficient of variation [53] is the ratio of the standard deviation to the mean, and therefore a unitless measure of spread that describes the amount of variability relative to the mean. For instance, the coefficient of variation of induction time the for the results of the experiment when performed by hand was obtained form Figure 4.4.3 (a) for the blue points (10μ) at the distance of 50 mm, and was compared by the coefficient of variation obtained from the experiment performed by the robot. Comparison of the results obtained from pipetting by hand and the robot, shows a reduction in the variability of the results from the robot, in particular a 26% decrease in coefficient of variation for the induction time. Table 4.4.1 summarizes the mean and standard deviation of experiment results performed by the robot.

Automation provides chemists with data which was not possible to be obtained performing experiments by hand. Using EvoBot, chemists are enabled to perform experiments with a precise spatial organization, e.g at fixed intervals of 1 mm, in a timely manner.

Further possibilities

EvoBot can be used to interact with the experiment while it is happening, e.g. adding a second salt droplet when the first droplet has moved a specific distance. In addition, it can also be used to make droplets move in certain trajectories by adding droplets of different concentrations at different positions relative to the decanol droplet.

In addition, it is possible to analyze the effects of salt addition for multiple decanol droplets or multiple salt additions. Furthermore, owing to automation, droplet patterns such as squares or circles can be precisely created.

Conclusion

In this experiment, we described how we applied computer vision and automation to improve an artificial chemical life experiment. For this experiment the use of the robot enabled systematic control of experimental parameters resulting in timely, and precise verification of the experiment results with fixed parameters, and less noisy experimental data. Overall, we conclude that computer vision and robot automation make it possible to perform chemical experiments where precise spatial placement and timing are important.

4.4.2 Use case 2: Sensor input feedback for aspirating droplet

In this experiment, as can be seen in Figure 4.4.4, a moving droplet is aspirated by the syringe module when the droplet speed goes below a specified threshold. These types of experiments are not possible to be performed by hand. However, by employing computer vision driven automation they become feasible.

Experimental Setup

As the first step, the robot dispenses 3 mL of 10 mM sodium decanoate (pH 11) in a 90 mm diameter Petri dish. The robot syringe is perpendicular to the Petri dish surface and the syringe tip was 1 mm above the Petri dish surface when dispensing the decanoate. The stepper motor of the syringe had a speed of 4 mm per second dispensing decanoate. The next step is adding a 40μL droplet of a 1 M NaCl solution (i.e., 10μmol of NaCl) at 10 mm distance from the edge of the Petri dish. Another syringe was used for this purpose. The robot syringe was perpendicular to the Petri dish surface and the syringe tip was 1 mm above the Petri dish surface when dispensing the Sodium Chloride. The stepper motor of the syringe had a

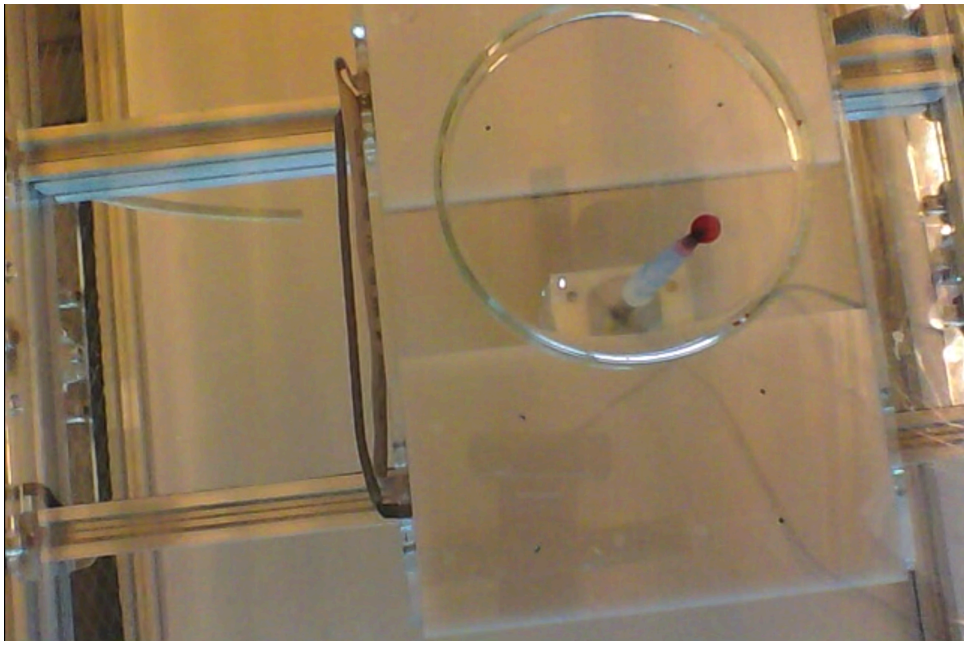


Figure 4.4.4: Aspirating a moving droplet by the syringe module when the droplet speed goes below a specified threshold.

speed of 4 mm per second dispensing the droplet. Finally, another syringe is used to dispense a 20 µL decanol droplet over the sodium decanoate, as can be seen in supplementary video 5 in Appendix A. The robot syringe was perpendicular to the liquid surface and the syringe tip was 1 mm above the liquid surface when dispensing the decanol. The stepper motor of the syringe had a speed of 4 mm per second dispensing the droplet.

Experiment Results

As can be seen in the supplementary video, the decanol droplet moves towards the salt gradient. When the speed goes below a threshold, the syringe tip aspirates the droplet. We repeated the experiment 10 times with the same droplet. In order to do this, the vision API of EvoBot tracks the droplets, and analyzes droplet speed. Figure 4.4.5 shows the droplet speed over time obtained from EvoBot's vision API. As can be seen in the Figure, whenever the droplet speed reaches 1 mm per second,

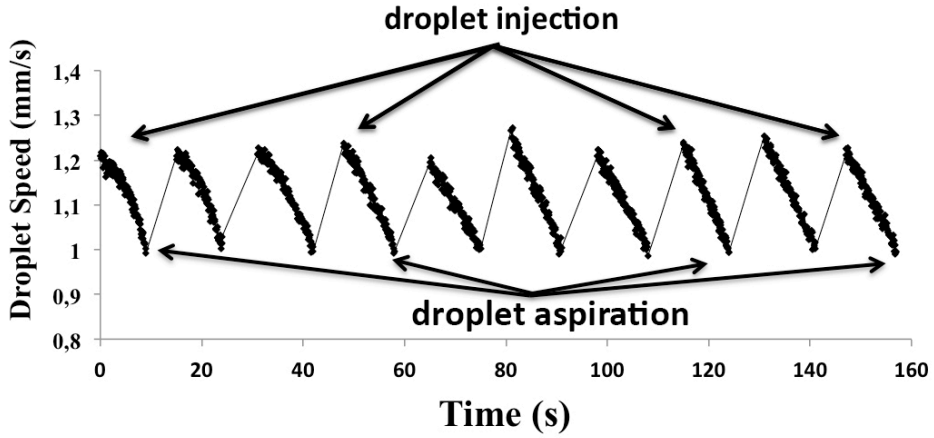


Figure 4.4.5: Droplet speed over time for use case 2. As can be seen in the supplementary video 1 in Appendix A, when droplet speed goes below a threshold, the robot aspirates the droplet. The experiment is repeated 10 times.

the syringe tip moves to the center of the moving droplet, and aspirates the droplet. The syringe tip touches the droplet vertically and in the center.

Conclusion

Analysis of the speed of a droplet composed of different chemicals is of interest to chemists. EvoBot makes interacting with the experiment based on droplet speed possible. This is useful as the composition of an aspirated droplet with certain speed can be analyzed with an external instrument. A similar application is injecting a chemical in a moving droplet when the speed reduces [71]. It is of interest for artificial life scientists to refuel a moving droplet when it stops motion or when the speed is below a certain threshold in order to make the droplet continue movement. Furthermore, collecting data such as droplet position, speed, acceleration, size, analyzing this data, and potentially modeling droplet behavior is crucial for chemists.

4.4.3 Use case 3: Sensor input feedback affecting group droplet behavior (Clustering Experiment)

In this experiment, as can be seen in Figure 4.4.6, multiple decanol droplets are added to a decanoate solution at different points. After a certain time these droplets come together and form a cluster. Now if a droplet of Sodium Chloride is added at a specific distance from the cluster, the experiment will be reversed, meaning the decanol droplets will de-cluster. However this step takes a long time, depending on the set of parameters even as long as a day. Subject to the parameters of the experiment such as the number of decanol droplets, how distant and in what pattern they are placed, pH of Decanoate solution, room temperature, decanol droplets cluster at different times and patterns. In addition to these parameters the location and time of salt injection affects the number of clusters over the course of the time and the total time needed to de-cluster completely.

The need to perform different long running experiments with various parameters makes EvoBot ideal for these experiments. EvoBot's vision API enables it to detect clustering and declustering and count the number of cluster over the course of the experiment. While humans need to waste time monitoring the experiment for declustering to happen, EvoBot can perform many of the experiments in parallel at the same time without the need for presence of a human. In addition, humans are not as accurate as the robot detecting the clustering, not precise enough to inject salt at a specific point relative to the cluster at a certain time, and it is too tedious and inaccurate to count the clusters over the course of a long experiment.

Supplementary video 6 in Appendix A demonstrates the feasibility of these types of experiments with the robot. First the robot adds 0.5 mL of 10 mM sodium decanoate (pH 11) on one end of a 76 × 26 mm microscope slide, and then adds another 0.5 mL of sodium decanoate in the center of the microscope slide. Therefore the sodium decanoate is distributed all over the slide. Then the robot adds a row of five 5µL decanol droplets, and a row of four 5µL droplets of decanol with a distance of 10 mm between droplets. Figure 4.4.7 shows the number of clusters over time for this experiment obtained from EvoBot's vision API. As can be seen

Figure 4.4.6: Multiple decanol droplets added to decanoate solution at different points forming a cluster.

in the video, the robot detects the desired behavior, as soon as the formation of the cluster, and injects a 10µL droplet of a 1 M NaCl solution (i.e., 10µmol of NaCl) at 10 mm distance to the cluster center. It should be noted that the data from this experiment is inconclusive, and provided as a showcase of possibilities of reactive experiments.

4.5 Conclusion

In this chapter, we demonstrated that by addressing hardware and software requirements for a liquid handling robot discussed in chapters 2 and 3, we were able to use the developed robotic platform for performing diverse experiments, including artificial chemical life experiments. As explained in chapters 2 and 3, EvoBot is developed to meet the image processing and robotic automation requirements to

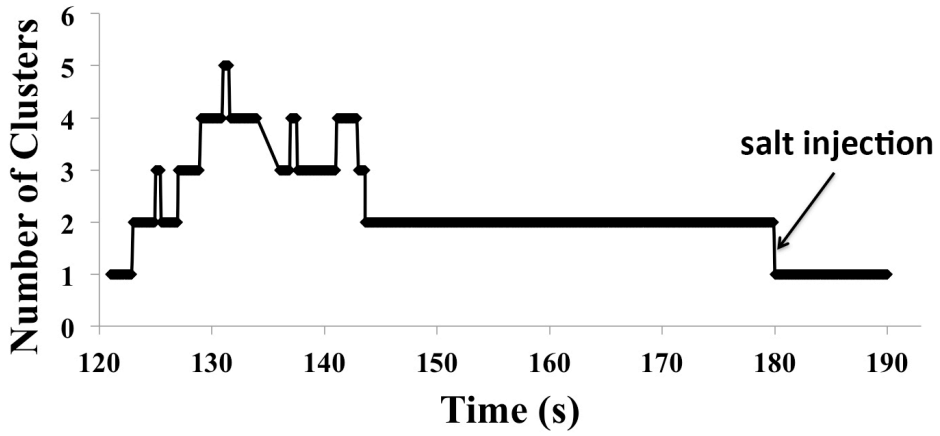


Figure 4.4.7: Number of clusters over time in declustering experiment. The number of clusters is obtained from the vision API of EvoBot. As can be seen in the supplementary video 2 in Appendix A, when droplets form a cluster, a salt solution is aspirated and dispensed at the specified distance to the cluster.

perform automated artificial chemical life experiments. The sensor layer under the experimental layer collects various droplet data collected from the image analysis of the camera, and is used as feedback to make artificial life experiments possible. The camera provides data about droplet properties, such as position, speed, area, and color, or about change in droplet behavior, such as droplets merging or dividing, clustering or declustering. Based on this data, EvoBot will interact with the experiment.

We commenced this chapter by showing our robotic system can perform routine liquid handling experiments. Another application of EvoBot described was 2D or 3D positioning of droplets at precise positions or in specific patterns forming complicated geometric shapes. We then demonstrated that EvoBot can be used to automate a variety of artificial life experiments. As a use case, we utilized our robotic system to investigate the response of a droplet as a function of distance to a reagent. The use of the robot enabled systematic control of experimental parameters resulting in less noisy experimental data. We described that computer vision and robot automation make it possible to perform chemical experiments

where precise spatial placement and timing are important. In addition, we used EvoBot to track motile droplets, and interact with the experiment, e.g aspirating a droplet when its speed goes below a threshold. We also used EvoBot to detect when a collection of droplets behaves in a certain way, e.g droplets clustering, and accordingly interact with the experiment, e.g inject a reagent at a certain distance from the droplet cluster.

In summary, the confluence of software required for artificial life experiments, and precise automation makes artificial chemical life experiments more precise and repeatable. Furthermore, new possibilities are provided for researches to perform experiments which were very difficult for infeasible to perform.

5

Robotic Platform with Integrated Controller Hardware

5.1 Introduction

So far we have described a robotic system capable of performing artificial chemical life experiments, and verified its applicability to the experiments. As discussed in section 1.7.5, one of the requirements of this robotic system is ease of software management, and fast setup of the system, without needing to install different software required for artificial chemical life experiments. In section 1.7.4, we explained another requirement for this robotic system is low price, so the robotic platform is affordable even for small laboratories. The robotic system we described in chapters 2.1, and 3 uses a dedicated external computer for running the required software, which constitutes a large portion of the price of the robotic system. In this

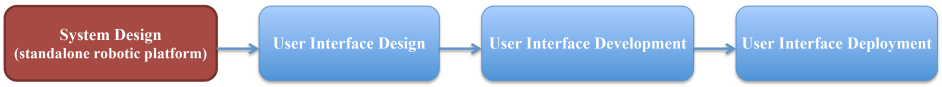


Figure 5.1.1: User interface procedure.

chapter, we address these two requirements, namely easy software management, and affordability.

We need to make our robotic platform usable on different computers without requiring to install all the software, and libraries. This is because EvoBot's application programming interface (API) uses numerous software packages, like OpenCV and PyQt. Installing these libraries and managing their dependencies on an external computer proved to be time consuming and difficult for non-expert users. In order to solve this issue, we need to decouple control software from user software. Therefore we have to replace the dedicated external computer with an integrated controller hardware. There are different single board computers available to use as an integrated controller hardware. In this chapter, we discuss our choice of single board computer. We explain how this choice enables us to meet the requirement of an affordable robot, and how easy it is to duplicate software for new platforms by mounting an image of the operating system with all required packages on a microSD card. We then describe the hardware and software implementation so we can run the experiments on this robotic platform with integrated controller hardware. Finally, we evaluate the performance of this standalone robotic platform with integrated controller hardware.

This chapter also comprises our first step towards developing a user interface for liquid handling robots. In this chapter, we discuss the implementation of a platform which provides the infrastructure for developing the user interface. As can be seen in Figure 5.1.1, in this chapter we focus on the system design. In the following chapters we will continue this route by designing the user interface, developing the user interface, and then deploying the user interface.

The approach we have taken for detaching user software from control software can be applied to similar robotic systems. Our choice of a cheap single board computer can also be employed in many robotic applications to make them affordable. Given that implementing hardware, and software for these applications can be time consuming due to some technical details. We describe the implementation of our platform to serve as a guideline for interested readers.

5.2 Requirements

In order to choose an integrated controller hardware, we need to consider the requirement of our robotic system for affordability. On the other hand, the hardware needs to have sufficient processing power to perform non-reactive liquid handling experiments, as well as feedback based experiments which require processing of image frames. We use Geekbench [24], a benchmarking tool to measure processor and memory performance of different devices. GeekBench runs a series of tests on the platform, and evaluates the performance by assigning a score to the platform. The scores are calculated against a baseline score of 4000, which represents the performance of an Intel Core i7-6600U @ 2.60 GHz. GeekBench is available on most platforms, e.g. Windows, Mac, Linux, Android, BlackBerry, iPhone, and produces consistent results across them.

Geekbench uses a number of different tests, or workloads, to measure CPU performance. The workloads are divided into four different sections, including crypto, integer, floating point, and memory. These workloads measure the performance of cryptography instructions, integer, and floating point operations, and memory, respectively. For instance, the floating point workload includes image processing operation tests such as blurring images, sharpening images, and dot products.

We had successfully used a MacBook Pro with a 2.5 GHz Intel Core i7 processor, and 16 GB 1600 MHz DDR3 Memory to perform various experiments with the robot. The Geekbench score for this computer was 3862. This provides an approximate metric in deciding whether an integrated controller hardware would

be a suitable choice. For instance, an integrated controller hardware with a Geekbench score ten order of magnitude smaller than the Geekbench score of the original computer is not a good candidate.

Finally, the form-factor for integrated controller hardware should also be taken into consideration. A choice of a not too large integrated controller hardware provides us with more flexibility.

5.3 Choice of Integrated Controller Hardware

In order to develop a robotic platform with integrated controller hardware, one option is to use a mini computer. Intel NUC Rock Canyon NUC5I5RYK, as can be seen in Figure 5.3.1a, is an example of a powerful mini computer. Intel NUC's processor is an Intel Core i5-5250U (1.6 GHz up to 2.7 GHz Turbo, Dual Core). This choice addresses our requirement for an integrated controller hardware, although priced at $400, it does not meet the affordability requirement. The not inexpensive price of mini computers drives us towards the choice of single board computers.

There are numerous single board computers (SBCs) designed with different specifications appropriate for different applications. Priced at just $9, CHIP, as can be seen in Figure 5.3.1b, is the cheapest SBC on the market. CHIP has built-in Wi-Fi, Bluetooth, 4 GB flash storage, 1GHz ARM R8 CPU, and 500 M RAM. However, CHIP's low processing potential, does not suit our application, as it has a Geekbench score of 413 [25] (10 order of magnitude smaller than the original computer we used to perform the experiments with). The BeagleBone™ Black Wireless, as can be seen in Figure 5.3.1c, is another SBC with 1GHz ARM Cortex-A8 Processor, and 512MB DDR3 RAM. However, it does not have an Ethernet port, and priced at $70, there are newer SBCs with higher processing power at a lower price in comparison (Geekbench score 248 [23]). Udoo x86 Ultra, as can be seen in Figure 5.3.1d, is another SBC comparable in power to that of a typical budget PC. The embedded Arduino 101 board, 8 GB of RAM, and a 2.6 GHz quad-core Intel CPU make this SBC a versatile tool. However, Udoo x86 has no in-

tegrated hardware to connect to Wi-Fi or Bluetooth networks, and priced at $259, it's expensive in comparison with SBCs. [10] provides the specifications for popular single board computers in 2016.

Raspberry Pi 3 is another single board computer priced at $35, and having a Geekbench score of 2128. This single board computer is our choice of the integrated controller hardware. The following section describes our motivation for this choice.

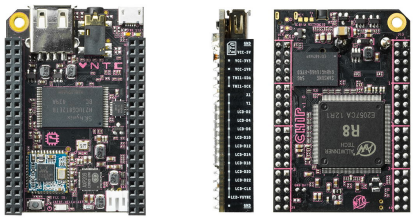
Motivation for the Choice of a Raspberry Pi 3

We have chosen a Raspberry Pi 3, a single board computer, in lieu of the external computer used in the robotic system introduced in chapter 2 as the integrated controller hardware for various reasons. One motivation for this choice is the reasonable price of the Raspberry Pi helps us satisfy the requirement discussed in section 1.7.4 for affordability. Furthermore, the processing power of the Raspberry 3 (Geekbench score 2128 [26]) is relatively close to the original computer we have used for experiments (Geekbench score 3862). Therefore it can be a potential candidate. In addition, the performance to price ratio of the Raspberry Pi 3 surpasses its counterparts. The Raspberry Pi 3 model B costs $35 due to mass production price reduction. In February 2016, the Raspberry Pi Foundation announced that they had sold eight million devices, making it the best-selling UK personal computer.

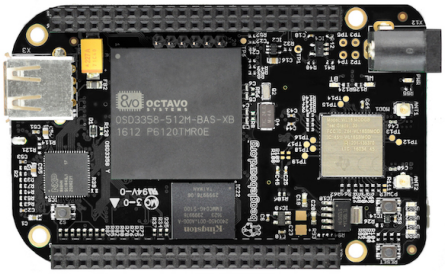
Moreover, the Raspberry Pi performs better than its average performance for the algorithms we use in our application programming interface. Figure 5.3.2 shows the floating point performance of Raspberry Pi 3. Specifically for tasks performed in our image processing algorithm, it outperforms the average performance. For instance, the score for dot product (multi-core scalar) is 4181, for sharpening images (multicore scalar) is 9460 , and for blurring images (multi-core scalar) is 12783, which is higher than the average Raspberry Pi performance score 2128.

Another motivation for this choice is the available Raspberry Pi resources, and

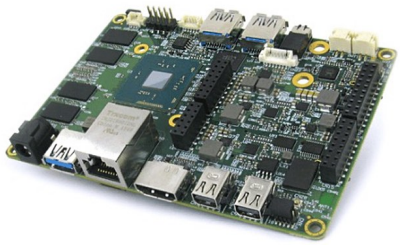
(a) Intel NUC ROCK CANYON Mini Computer



(b) CHIP Single Board Computer.



(c) BeagleBone™ Black Wireless Single Board Computer



(d) Udoo x86 Ultra Single Board Computer

Figure 5.3.1: Mini computer and single board computers for integrated controller hardware.

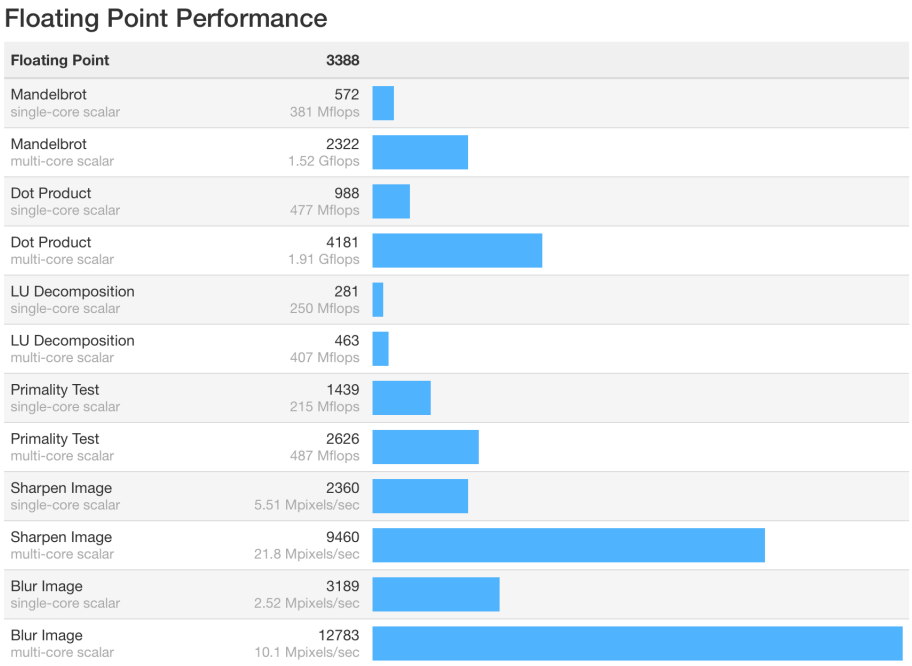
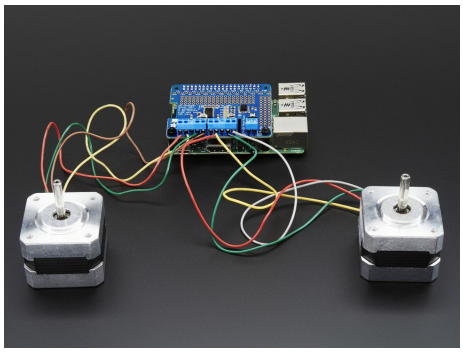


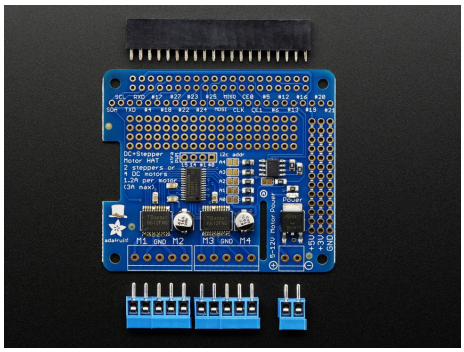
Figure 5.3.2: Raspberry Pi 3 floating point performance.

the active Raspberry Pi community. Having the largest and most active community among SBCs, the amount of guides, tutorials and software available for the Raspberry Pi is unmatched by any other competitor. There is close to no chance of a routine problem not already have been covered by the community. Furthermore, the native Raspberry Pi module V2 camera, and the Picamera library provide valuable advantages for reactive experiments as will be discussed in the following sections. In addition, replicating disk image for a new robot is as easy as copy pasting using a Raspberry Pi. Moreover, as will be explained in chapter 8, the availability of the Ethernet port, and WiFi module allows us to preload the network SSID and secret key so that the robots can introduce themselves to the web server automatically.

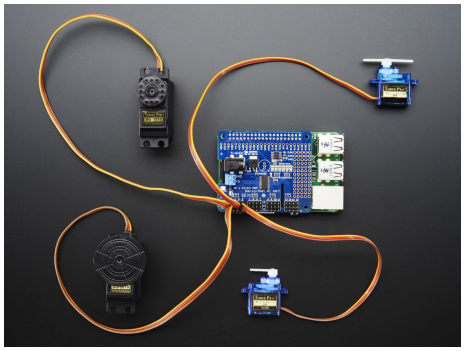
Another motivation for this choice is that Raspberry pi 3 comes with HATs, i.e. add-on boards attached on top of the Raspberry Pi, for different applications. The functionality provided by these HATs, for instance motor or sensor support,



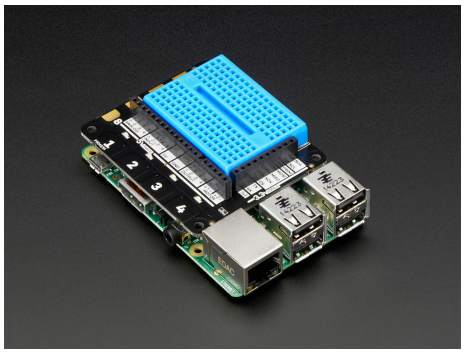
(a) Raspberry Pi 3 Stepper motor HAT



(b) Raspberry Pi HATs can be stacked up using a stacking header.



(c) Raspberry Pi 3 Servo motor HAT



(d) Raspberry Pi 3 Pimoroni HAT

Figure 5.3.3: Raspberry Pi HATs for different applications.

can be potentially integrated into our robotic platform. Adafruit Stepper & DC Motor HAT for Raspberry Pi 3, as can be seen in Figure 5.3.3a, is an example that can drive up to 4 DC or 2 Stepper motors with full PWM speed control . As the Raspberry Pi does not have many PWM pins, this board uses a fully-dedicated PWM driver chip onboard to both control motor direction and speed. This chip handles all the motor and speed controls over I^2C, and up to 32 Motor HATs can be stacked for controlling up to 64 stepper motors or 128 DC motors if soldered with a stacking header, as can be seen in Figure 5.3.3b. Adafruit 16-Channel PWM / Servo HAT for Raspberry Pi , as can be seen in Figure 5.3.3c, is another HAT for controlling DC Servo Motors. These motors need specific and repetitive timing pulses to set the position. This board can drive up to 16 servos or PWM outputs over I^2C with 2 pins. Pimoroni Explorer HAT Pro, as can be seen in Figure 5.3.3d, is another example for reading analog sensors, interfacing with 5V systems, and touch interfaces.

5.4 Hardware Components

This section describes our choice of hardware parts in order to develop a standalone robotic platform with integrated controller hardware, including a Raspberry Pi 3 Model B, Raspberry Pi camera module V2, flex cable for Raspberry Pi Camera, and 15.6" HDMI interface LCD with capacitive touchscreen. Then we describe integrated controller hardware setup for this standalone robotic platform.

5.4.1 Raspberry Pi 3 Model B

The Raspberry Pi 3 controls the robot, and processes data obtained from sensing modules. The Raspberry Pi is a series of credit card-sized single-board computers developed in the United Kingdom by the Raspberry Pi Foundation. Several generations of Raspberry Pis have been released, the first generation (Pi 1) being in February 2012, and the latest Raspberry Pi 3 model B in February 2016.

The Raspberry Pi 3, as can be seen in Figure 5.4.1a, comes with:

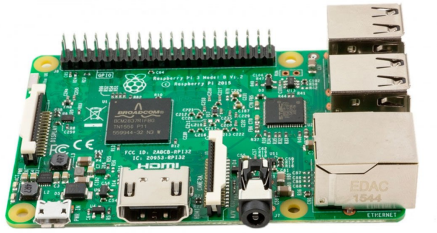
- A 1.2GHz 64-bit quad-core ARMv8 CPU
- 802.11n Wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)
- 1GB RAM
- 40 GPIO pins
- Ethernet port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Push-pull Micro SD card slot
- VideoCore IV 3D graphics core

5.4.2 The Raspberry Pi camera module v2

The V2 camera, as can be seen in Figure 5.4.1c, has a Sony IMX219 8-megapixel sensor, and can be used to take high-definition video, as well as still photographs. We connect the camera to the Raspberry pi Camera Interface (CSI). We use this camera for reactive experiments, as well as streaming video over the network, when required.

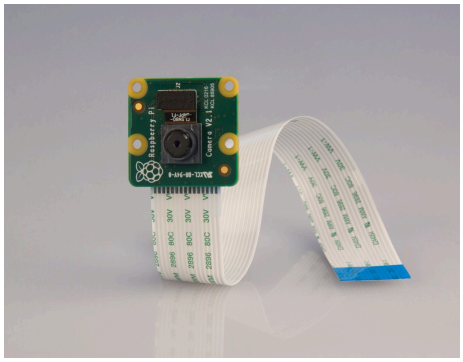
5.4.3 Flex Cable for Raspberry Pi Camera

This cable, as can be seen in Figure 5.4.1d, lets us swap out the stock 150mm long flex cable from a Raspberry Pi Camera for a different size. We need to open the connector on the Raspberry Pi and slip this one in. This is useful for accessing all the space on the experimental layer of the robot.



(a) Raspberry Pi 3 Model B

(b) Capacitive Touchscreen



(c) Raspberry Pi camera module v2

(d) Flex Cable for Pi Camera

Figure 5.4.1: Hardware for platform with integrated controller.

5.4.4 15.6" HDMI Interface LCD with Capacitive Touchscreen

An LVDS cable connects the 15.6" 1366 x 768 pixel glossy LCD to the HDMI-LVDS converter board that has all required voltages for LCD, and can power Raspberry Pi through a USB connector. An HDMI extender connects the convertor board to the raspberry pi through the HDMI interface. The LCD can be seen in Figure 5.4.1b.

5.4.5 Integrated Controller Hardware Setup

The Arduino and the Raspberry Pi 3 transfer data through serial communications with a standard A-B USB Cable, as can be seen in Figure 5.4.2. The Raspberry Pi is

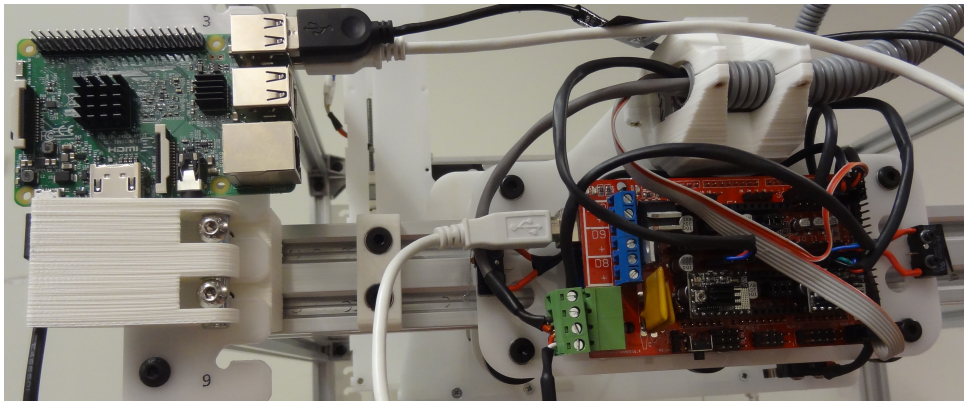


Figure 5.4.2: Implementation of platform with integrated controller hardware.

connected to the touch screen through the HDMI interface. A $5V, 3A$ power adaptor provides power to the HDMI-LVDS converter board that powers the Raspberry Pi through the USB connector. A mouse or a keyboard can be added to the Raspberry Pi via USB port. The Raspberry Pi includes a wireless network adaptor as well as an Ethernet port to connect to the Internet. Figure 5.4.3 shows the setup.

5.5 Software Implementation

Raspbian, a Debian-based Operating System, is the Raspberry Pi Foundation's official supported Operating System. Raspberry Pi Foundation promotes Python as the main programming language, although C, C++, PHP, Java, Perl, Ruby, and many more are supported. We mounted an image of the Raspbian operating system on an MicroSD 16GB Class 10 Memory Card using an SD/MicroSD adaptor, and inserted the MicroSD into the Raspberry Pi.

We use the Advanced Package Tool, or APT, which is a free software user interface that works with core libraries to handle the installation and removal of software on the Debian-based Linux distributions. To install the required libraries for EvoBot's API, we also use pip which is a package management system used to in-

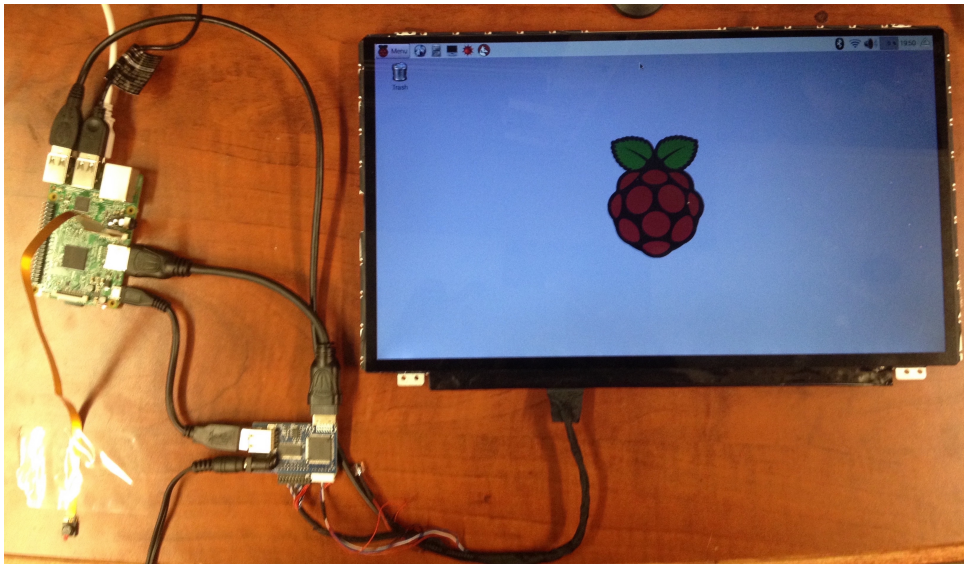


Figure 5.4.3: Setup.

stall and manage software packages written in Python. Python 2.7, and Python 3 come preinstalled on Raspbian. We also installed NumPy, SciPy, matplotlib as required for the Evobot's API. We installed pySerial to manage the serial communication between the Raspberry Pi and the Arduino.. Installing OpenCV 3.1.0 for image processing was more straightforward than installing on a Mac, but compiling OpenCV for Python takes about 75 mins on a Raspberry Pi. We also installed Flask, which is a Python web framework on Raspberry pi.

We used Cron to automate managing specific tasks. Cron is a unix utility that allows tasks to be automatically run in the background at regular intervals by the Cron daemon. Owing to cron, required tasks can be scheduled to run at reboot or at specific time intervals.

5.6 Performance

We used the robotic platform with integrated controller hardware to perform formerly performed experiments with EvoBot and evaluate the performance of the new platform. We needed to evaluate the performance of EvoBot for non-reactive

experiments, as well as reactive experiments where feedback is provided by analyzing camera image frames.

In case of non-reactive experiments, we managed to run the code on EvoBot's repository without any modification. This is because we have installed the Raspbian Jessie with Pixel to provide a GNOME (GNU Network Object Model Environment) for users. Therefore Raspberry Pi can be used as a Linux computer by connecting a touch screen, and keyboard, as Python and the required libraries had been successfully installed.

In case of reactive experiments requiring a camera, the Raspberry Pi camera from Raspberry Foundation, and the PiCamera library for Raspberry Pi can be used for image processing. However, we could also use the same camera we had used with EvoBot, with the Raspberry Pi, without the need to change code.

We first evaluated the performance of experiments when EvoBot's webcam is connected to the Raspberry Pi. This option didn't require any major code change. We compared the performance of EvoBot running with a Mac computer, and the robotic platform with integrated Raspberry Pi 3 controller running reactive experiments requiring OpenCV. There are various reactive experiments performed with EvoBot, which have their own computational requirements. We first compared the performance of EvoBot, and the platform with integrated controller hardware for basic image processing tasks. The Mac was a MacBook Pro with a 2,5 GHz Intel Core i7 processor, and 16 GB 1600 MHz DDR3 Memory. The Raspberry Pi managed to have the same performance as the computer, and frame-processing rate was limited by the camera frame rate (around 18 frames per second). We then compared the frame processing for an extreme case of multi droplet tracking which requires image processing, processing experiment data, logging them in detail, writing experiment data to disk, displaying live experiment video, and recording experiment video. EvoBot performed this experiment at about 11 frames per second. In this extreme case the Raspberry Pi performed the experiment at 3 frames per second. However, the performance of the Raspberry pi could be increased if we didn't display the live experiment data, or ease the processing requirements.

We then needed to evaluate the performance of experiments with the Raspberry

Pi Module 2 camera provided from Raspberry Pi foundation. The camera provided from Raspberry Pi foundation, along with the optimized PiCamera library for Raspberry Pi performs very well. However, we could not use the EvoCam API without any modification as it was. Therefore, we modified EvoCam to support PiCamera and OpenCV working together. Consequently, we were able to use EvoCam with the webcam, or the Raspberry Pi camera, or both of them at the same time.

Having the new EvoCam library with PiCamare support, we evaluated the performance of Raspberry Pi, and the pi camera. For basic image processing experiments, we had the same frame-processing rate for the case of the webcam connected to the computer, and the Raspberry pi connected to the Raspberry camera. For the extreme processing case assessed earlier, we had a great increase in performance processing at 8 frames per second (in comparison with 11 frames per second with the computer and EvoBot's webcam).

We then tried overclocking the Raspberry Pi. We needed to use heat sinks on the Raspberry, as can be seen in Figure 5.6.1, to avoid damaging the chips. We had to increase the processor, gpu, and sdram frequency for better performance. The details for overclocking the Raspberry Pi can be found in Appendix C

It should be noted that changing the "up_threshold" on the Raspberry Pi would cause it not to increase the speed until CPU utilization gets above a certain threshold, for instance %75. Using these settings we managed to process frames at 10. frames per second for the extreme case.

For common image processing tasks, mean and standard deviation of frame processing rate per second is m=18.6, s=0.9 for 2,5 GHz Core-i7 Mac Book Pro, and m=17.8, s=1.1 for Pi. For experiments with extreme processing requirements, we overclock the Pi by increasing GPU, and CPU frequencies when CPU utilization exceeds a certain threshold (75%). The above-mentioned computer processes frames at m=11.3, s=0.8, and Pi at m=10.3, s=0.9 (91 performance).

In Summary, evaluation of the robotic platform with integrated controller hardware demonstrates it is possible to perform routine liquid handling experiments

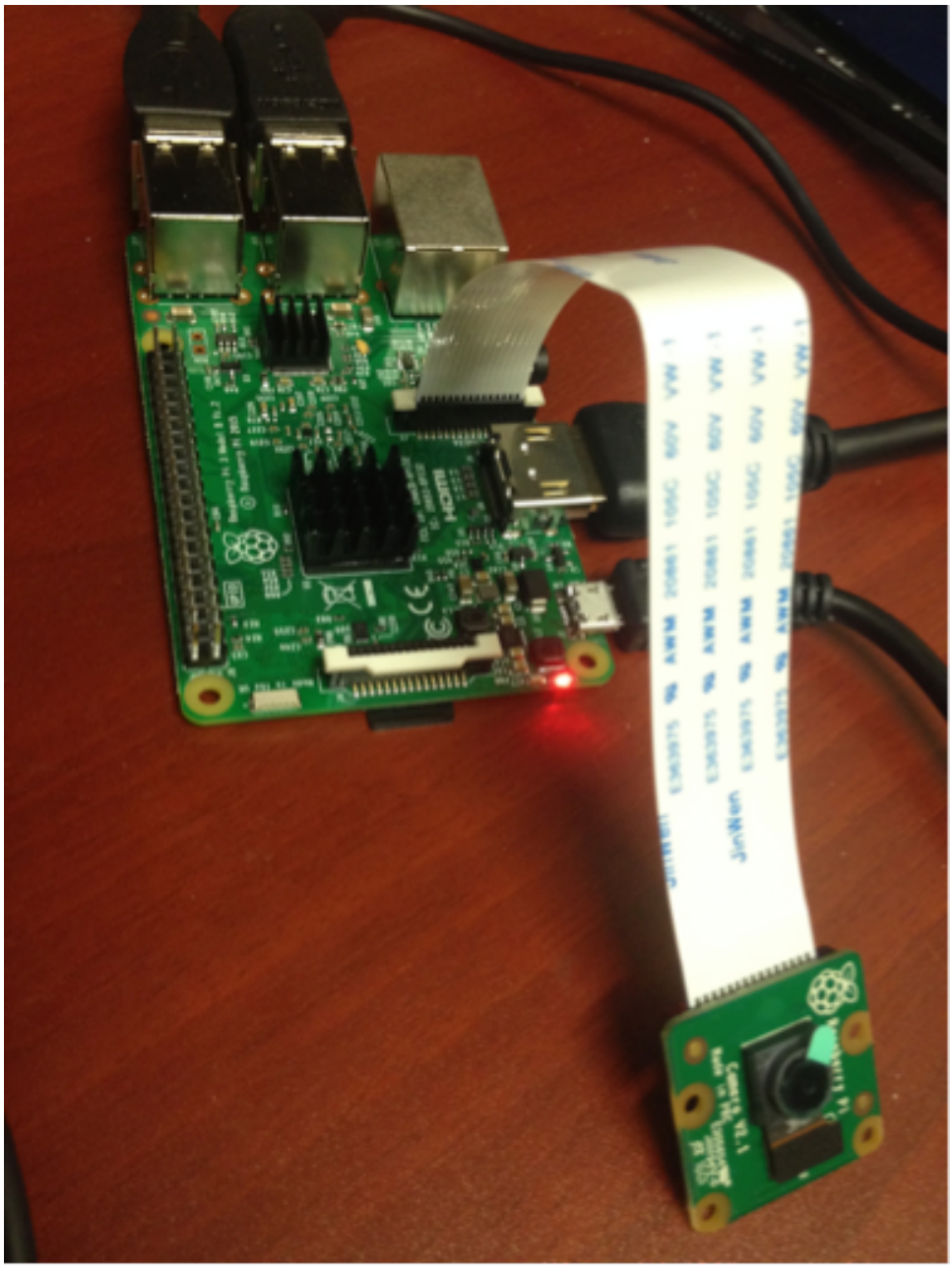


Figure 5.6.1: Using heat sinks to overclock the Raspberry Pi.

and it has sufficient computational power to perform feedback-based experiments, owing to tight integration of hardware and software. We use Raspberry Pi Foundation's native camera-module v2. On software side, we boost performance by building our image-processing API on top of optimized Raspberry Pi Camera library and GPU accelerated OpenCV. Finally overclocking the Raspberry Pi can result in processing performance close to EvoBot running with the computer.

5.7 Conclusion

In conclusion, we introduce a robotic platform with integrated controller hardware, resulting in an architecture where user software for programing experiments is decoupled from control software. We discussed possible approaches to develop such a platform, and design consideration we have taken into account for our choice of the implementation. We showed that the resulting platform eases software management as installing, and managing software libraries required for feedback based experiments on different hardware, and operating systems was difficult. Furthermore it is affordable owing to the low cost of the Raspberry Pi. This robotic platform with integrated controller hardware is able to perform liquid handling experiments, including feedback based experiments in artificial chemical life. Using the native Raspberry Pi camera, developing image processing Application programing interface (API) for Raspberry Pi, and overclocking the Raspberry Pi 3 adaptively, we gained satisfactory performance.

User interface is like a joke.
if you have to explain it to someone, it's not a good one!

...

6
User interface design

6.1 Introduction

In the preceding chapters, we have described a robotic system capable of performing artificial chemical life experiments, and have verified its applicability to these experiments. We also decoupled user software, and control software by integrating controller hardware into the robotic platform for easy software management.

As we explained in section 1.7.6, another requirement for our robotic platform is an intuitive user interface enabling users to use the functionality of our robotic platform effectively. Most of the users of our robotic platform for performing artificial chemical life experiments are chemists, biologists, etc. As these users are not proficient in programming, they would prefer a graphical user interface rather than writing code in order to be able to perform the experiments. Therefore making the features of our robotic platform usable for these users is critical.

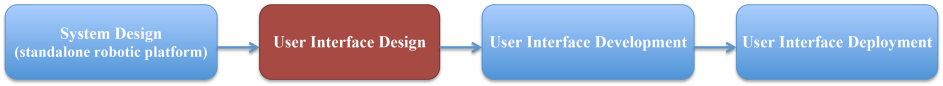


Figure 6.1.1: User interface procedure.

Design of this user interface is important as it ensures a good experience for the users. The user interface is the gateway to the system. Even if all the other parts of the complicated robot system are very well built, but the user interface is not easy to use, all the other effort is wasted. A good user interface has a very important role in making the users willing to use a product.

User interface design is also an important part of the software development chain. Usability bug fix costs increase as the bugs propagate through requirement analysis, design, coding, testing, and production. Research has shown it costs five times more to fix a problem in the design phase than in the requirement phase. It costs ten times more if the same issue propagates to the coding phase. If the usability bug gets to deployment it can cost from 100 times more to fix [3] [2] [48]. Furthermore, a good user interface reduces costs, and saves users' time by eliminating the need for providing unnecessary documentation for the system.

This chapter also comprises our second step towards developing a user interface for liquid handling robots. In chapter 5, we discussed the implementation of a platform which provides the infrastructure for developing the user interface. This infrastructure includes both hardware and software. As can be seen in Figure 6.1.1, in this chapter we focus on the the user interface design and evaluation. In the following chapters we will continue this route by designing the software architecture required for developing this user interface, and deploying the user interface on the cloud.

In order to design an intuitive user interface for our robotic system we need to go through several tasks. We need to observe users and based on the observations analyze the requirement for the user interface. Furthermore we need to satisfy the principles of user interface design. Having observed users and met the principles

of user interface design, we need to come up with a mock up of the user interface that can be tested. We need to perform a heuristic evaluation of this interface by experts. Moreover, this interface should be evaluated by real users of this robotic system. In this chapter, we will go through these steps.

6.2 User Requirement Analysis

Analyzing the requirements of users for an intuitive user interface was our first task. This section describes our effort to this end. In chapter 3, we described the development of software that would allow control of our robotic platform using Python code. However, given the preferences of our users for a graphical user interface rather than a programming interface, we describe the development of a simplified user interface that will eliminate the need for programming skills when operating our robotic platform. In this more demanding context, users would not be expected to have any substantial training and the interface should expose the key functionality in a simple and easy to use way.

The user interface of the robot was developed using the User-Centred Design (UCD) framework [93]. As a first step in the user centered design, we carried out field visits to interview and observe users and learn to understand their needs. The field visits were carried out in chemistry and biology laboratories at University of Toronto, Canada. Users carried out real tasks, and their performance and reactions were observed, and recorded. We analyzed the goals users were trying to achieve, how users performed tasks without the interface, the parts they liked or disliked about it, the difficulties they experienced along the way, and the workarounds they used.

Our field visits with users showed us how tasks should be divided into logical steps that make sense to users. For instance, we learned that some experiments may require the syringe tip to be touching the liquid in vessels for better dispensing results. Due to the properties of surface tension, some liquids may need to be blown out while dispensing. We also learned that, when working with micro well plates, the edge wells should be excluded in some experiments, as their evaporation

pattern is different from that of the inner wells. Another lesson was that using air gaps while absorbing may be required (depending on the experiment) for better liquid handling performance.

6.3 Design Principles

As the next task, we go through user interface design principles. In this section, we enumerate the design principles we have taken into account for the design of our user interface.

6.3.1 users are not design experts

In order to come up with the best design of the user interface, it is crucial not to ask the actual users of the robotic system what kind of a user interface they would prefer for the system. This is because they are not user interface design experts, and they often get confused encountered with this question. This is referred to as the first rule of usability [86]. When we were conducting the interviews with users this point was well taken into account. We didn't ask users what kind of user interface they would like, but rather how they would do the task. We interviewed them for what the common tasks were, and what tasks were less frequent. We tried to develop the subsequent design based upon an explicit understanding of users, tasks and environments with respect to the domain of liquid handling experiments.

6.3.2 No Panacea

While we interviewed users we understood there are diverse types of experiments users do. This was because they would come from different fields, for example chemistry or biology. Furthermore, different chemistry or biology researchers focused on different types of experiments. In other words, there were different personas. Personas are fictional characters created to represent the different user types that might use a site, brand, or product in a similar way [79]. We used the observations from user interviews to create personas.

Trying to design a user interface that satisfies the needs of every potential user is a wrong practice [83]. A real world example is trying to design a car which is sport, SUV, and pick up. Therefore we didn't aim to design an interface that can please every possible user. Our designed interface was centered around the primary persona, the most influential persona among the cast [50]. The primary persona represents the core users, or uses core features, and is the main focus of the design [44] [49]. Our primary persona were chemists doing common chemistry experiments in a chemical laboratory.

6.3.3 Functionality vs Usability

When designing a user interface, there is a compromise between functionality and usability. The more features included in the user interface, the more functionality the user interface will have. However, it comes with the price of reducing the usability. A good example is iTunes, which has good usability. Although, potentially much more features could have integrated into this product, the most crucial ones have been included into the design as a trade off between functionality, and usability. This example demonstrates an important user interface design principle, the necessity of excluding features for the sake of improving usability [92]. Furthermore, according to Hick's law, "The time taken to make a decision increases as the number of choices is expanded." [102].

As computer scientists, we were first tempted to provide as many features as possible, and include every detail in the user interface. We covered even less probable scenarios, and all corner cases. However, this had made the design less usable. Therefore we took this principle into account, and considered the trade off between functionality and usability. Therefore, our final user interface is designed to be usable as well as having only the core functionality required.

6.3.4 Progressive disclosure

Progressive disclosure is a way to satisfy both of the conflicting requirements for power, and simplicity in a user interface. Progressive disclosure is accomplished

by initially, showing users only a few of the most important options, then offering a larger set of specialized options upon request. Therefore the secondary features are only disclosed if a user asks for them, meaning that most users can proceed with their tasks without worrying about this added complexity [89].

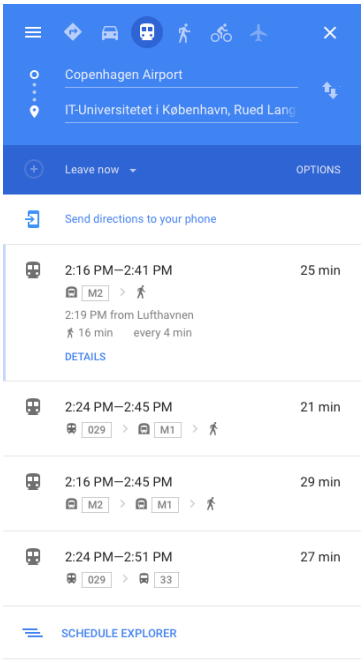
Progressive disclosure makes a complicated user interface simple and powerful at the same time. Google maps is a good example [91]. It has complicated functionality and at the same time provides a simple straight forward interface as can be seen in Figure 6.3.1 (a) . At the same time there are numerous features available based on demand. If required, the user can click on "leave now" to have access to more options like "depart at", "or arrive by". Furthermore, clicking on the "OPTIONS", the user can choose to avoid tolls, etc, or display distance in miles, etc, as can be seen in Figure 6.3.1 (b). These features don't confuse users by complicating the user interface, when the users do not actually need them.

In addition, contrary to common belief, the number of clicks to reach a specific target is not as decisive as believed. If each click is an unambiguous choice, clicking to reach the target would be seamless, and the number of clicks are not as significant [75].

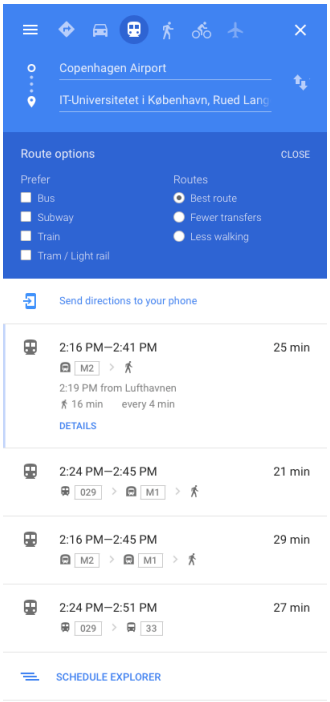
We have considered this design principle in our user interface design. Therefore details in the interface are not presented to the user unless the user asks for them. Consequently, unnecessary complexity is avoided. An example in our user interface design is disclosing several check boxes only when clicking on a button.

6.3.5 Iterative design

Iterative design is a method for developing user interfaces by refining them iteratively over several versions [87]. Iterative design, or incremental design, is more suited for user interface design than waterfall design. In waterfall design, production is the ultimate goal, and change in specifications is very difficult or nearly impossible as waterfall design aims for functionality and stability. On the other hand, in iterative design, users are the main goal of the design, and changes in specifications are inevitable. Iterative design aims for usability and usefulness [104] [97]



(a) Simple Interface



(b) Interface Details

Figure 6.3.1: Progressive disclosure in Google Maps

[78] [101].

The design of our user interface was based on iterative design. We tested each version of the design with users, and tried to make the iterations as much as possible.

6.3.6 Validated learning

Validated learning is the process in which learning is achieved by trying out an initial idea and then measuring it to validate the effect. Each test of an idea is a single iteration in a larger process of many iterations whereby something is learnt and then applied to succeeding tests [94]. In other words, validated learning is a form of iterative design where the design team test design hypotheses with users [37].

We implemented validated design in our design of the user interface. This originated from the fact that there is no intelligent designer that can sit alone and design the best user interface, but rather actual users provide the best feedback on the user interface. If experimental validation of a design fails, the design is not acceptable. It does not make any difference how beautiful a design looks, or how intelligent the user interface designer is [77].

6.3.7 Testing early

In most projects, the first system build is barely usable. The only question is whether to plan in advance to build a throwaway or promise to deliver the throwaway to customers [52]. Furthermore, Testing one user early in the project is better than testing 50 near the end [64].

A good example for this principle is the design of MacBook. It has been a process where they discovered the product through constantly creating new iterations. Apple has made hundred prototypes of the product, in comparison with few prototypes common in other companies. The team has discovered the design through an exhaustive process of building numerous prototypes [74].

The preceding discussions motivated us for testing our design of the user interface in the early stages with the users.

6.4 Low Fidelity Prototype

Next step was to build a testable mock up of the user interface. To this end, a focus group was carried out with three experts in human factors and user interface design. This was because we were following User-Centered Design (UCD), and UCD is a method that relies on interactive design, prototyping and testing after the requirements analysis. The goal was to determine the overall strategy for constructing the user interface for dilution experiments, a frequent task in a number of scientific areas that might use our liquid handling platform. The experts recommended a strategy where the user first configures the robot and then programs the experiment, using a menu selection approach.

There were several tools for wireframing this design. Axure RP Pro is a wireframing, rapid prototyping, documentation and specification software tool aimed at web and desktop applications [43] . Sketch3 is a vector graphics editor for Apple's macOS . Invisionapp is a prototyping, collaboration, and workflow platform [29]. Using Invision, prototypes can be developed online, without installing any software. Then they can be sent to desktop or mobile devices, e.g. Android or iOS, to be installed and evaluated on the device. Balsamiq Mockups is a graphical user interface mockup and website wireframe builder application. It allows the designer to arrange pre-built widgets. The application is developed as a desktop version, and also a plug-in for Google Drive . Balsamiq makes fast Iterations possible without much effort, and makes collaboration smooth.

We chose Balsamiq for wireframing owing to its simplicity,and fast implementation time. The prototype of the user interface consists of two pages. Figure 6.4.1 and Figure 6.4.2 show two screenshots for the resulting Balsamiq prototype developed for setting up and programming experiments with our robotic platform. Figure 6.4.1 shows the layout page, used for setting up and configuring the experiment. On this page, the user chooses the vessels, and assigns them a position based

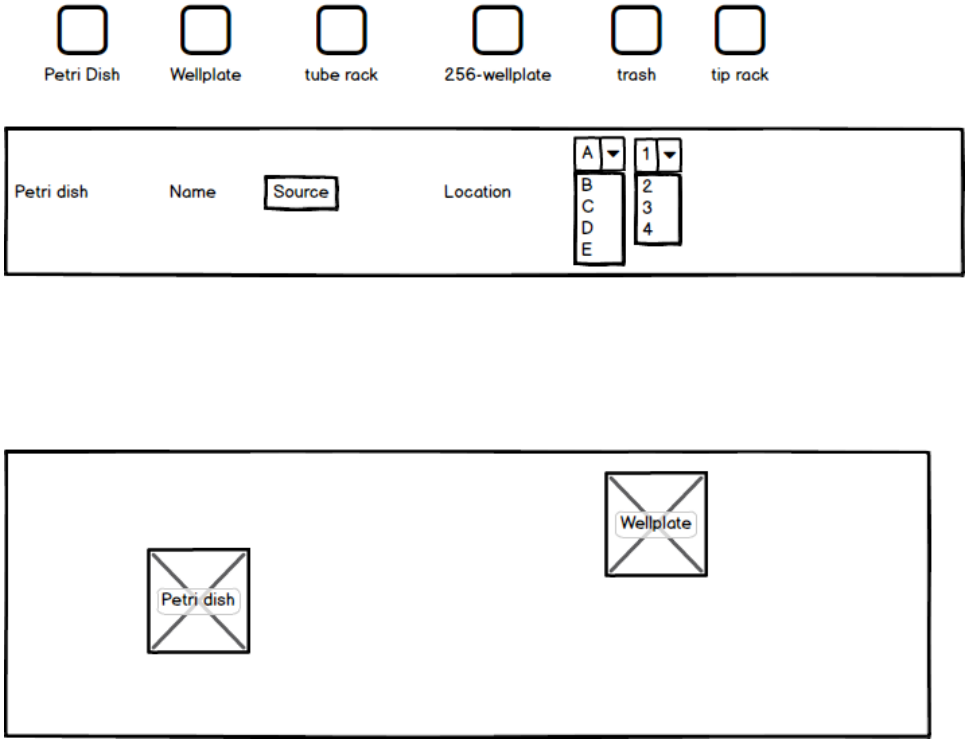


Figure 6.4.1: Layout mock up.

on his/her observation of the experimental layer. There exists also a text box in which the user enters an arbitrary, easy to remember name for vessels, so they can be referred to in the next step. Having defined the layout, the user proceeds to the next page.

Figure 6.4.2 shows the screen for programming the experiment. The user defines the desired experimental protocol on this page. At the top of the page, common tasks in a chemical experiment can be seen. The user chooses a task among these, and based on the choice a box is presented on the screen. He will then set the required parameters in that box based on the choice of the task. Having added the required steps, the user can either run the experiment, or save it by clicking the corresponding buttons. Finally, at the bottom of the page, the experiment log is displayed, providing useful feedback for the users.

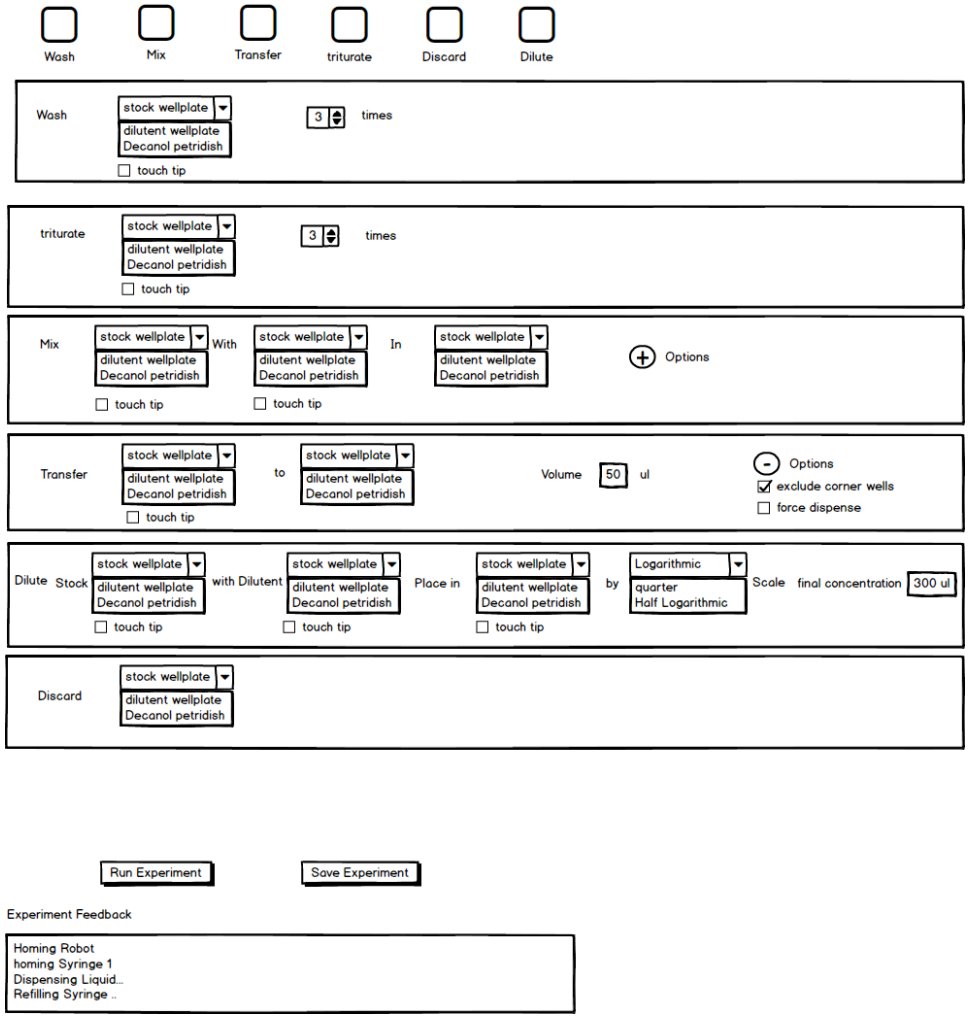


Figure 6.4.2: Experiment setup mock up.

6.5 Heuristic Evaluation

Next task we had to go through was expert evaluation of our graphical user interface. To this end, once the low fidelity prototype was constructed, it was tested with Heuristic evaluation, a method of discount usability engineering that allows efficient formative evaluation of user interface designs based on the assessments of expert reviewers, guided by a set of heuristics [90]. Clarkson and Arkin [59] examined a number of lists of heuristics that were subsequently developed, and then integrated them into a new list in the special context of human-robot interaction:

1. Sufficient information design.
2. Visibility of system status.
3. Appropriate information presentation.
4. Use natural cues
5. Synthesis of system and interface
6. Help users recognize, diagnose, and recover from errors
7. Flexibility of interaction architecture
8. Aesthetic and minimalist design

An expert evaluator used the Clarkson and Arkin list to identify usability problems in the low fidelity prototype of our robotic platform's user interface. Each problem identified was labelled with a severity based on the original ratings provided by Nielsen [88] which vary from 1-4, where 1 is a cosmetic problem only, and 4 is "usability catastrophe"

Table 6.5.1 shows the eight high severity problems that were identified. Two of them were found in the configuration component of the interface and six of them were found in the experiment programming interface. Seven (1-7) of the 8 heuristics listed by Clarkson and Arkin were involved in defining the problems (numbers provided in third column of Table 6.5.1).

Table 6.5.1: Heuristic Evaluation

Interface	Evaluator Observation	Heuristics	Severity
Set up environment	The meaning of the "location" input field is unclear.	3, 6	4
	The experimental layer representation does not clearly reflect the real-world.	3, 4, 5	4
Program Experiment	Terms and phrases do not reflect users' language.	4	4
	Units are manually input from a keyboard.	3, 6	4
	Units are hardcoded into the interface.	3, 4, 6 ,7	4
	Required input parameters are missing.	1, 3	4
	Tasks cannot be removed or reordered.	7	4
	The "run experiment" function is unclear.	2, 6	4

6.5.1 Addressing expert evaluation

Usability problems identified by expert evaluators in the low fidelity prototype of our robotic platforms's user interface were addressed. Figure 6.5.1 shows the resulting screen for setup and configuration, and Figure 6.5.2 shows the resulting screen for programming the experiment.

6.6 User Interface Evaluation

Evaluation of our user interface design by real users of the system was the final task that we had to perform. For this purpose, we evaluated our user interface with 15 users with diverse expertise in chemistry, biology, artificial chemical life, etc, from different universities, including University of Toronto, Canada, Mcgill University, Canada, University of Trento, Italy, University of Western England, UK, Karlsruhe institute of technology, Germany. The user interface was constantly improved by feedback from users. One example of feedback was to correct the use of technical terms. For instance, we replaced absorb with aspirate as they refer to different tasks in laboratories, as absorbing is a physico-chemical effect. The term triturate was also replaced by pipet up and down, as not all users were familiar with this expression. Dilution was changed to serial dilution, and the parameters for the task were modified to reflect how chemists perform the task in practice. We used the System Usability Scale (SUS) to evaluate our user interface. SUS is a highly robust and versatile tool for usability testing [109]. In order to calculate the SUS score, users answer questions on a questionnaire as can be seen in Figure 6.6.1.

The SUS score is calculated from the responses on the questionnaire [51]. The responses have values from 1 to 5 (from left to right on 6.6.1). To calculate the final score, we add the value of each of the odd-numbered questions minus 1. For instance, if the user has chosen the second choice for question 3, the score 2-1=1 will be calculated for this question. For even- numbered questions we subtract the

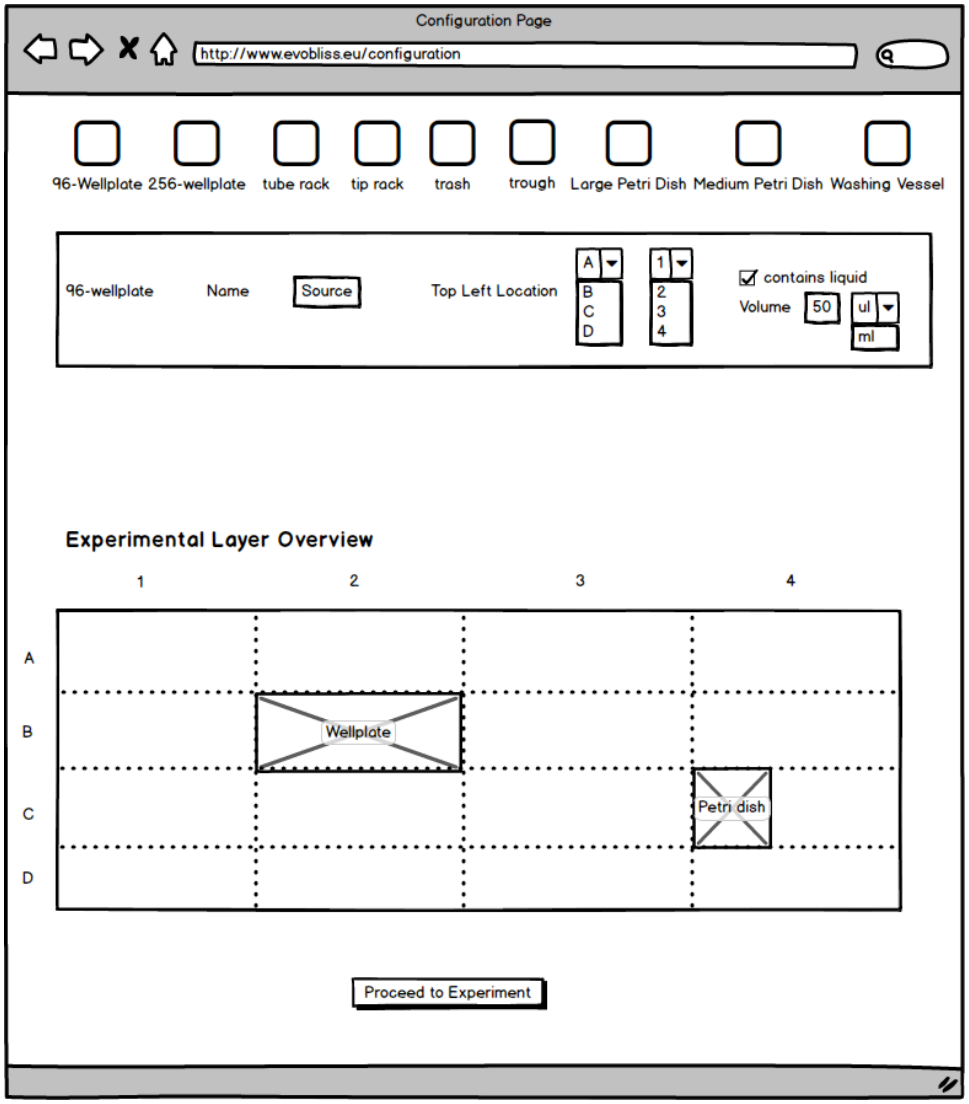


Figure 6.5.1: Iterated version of layout mock up.

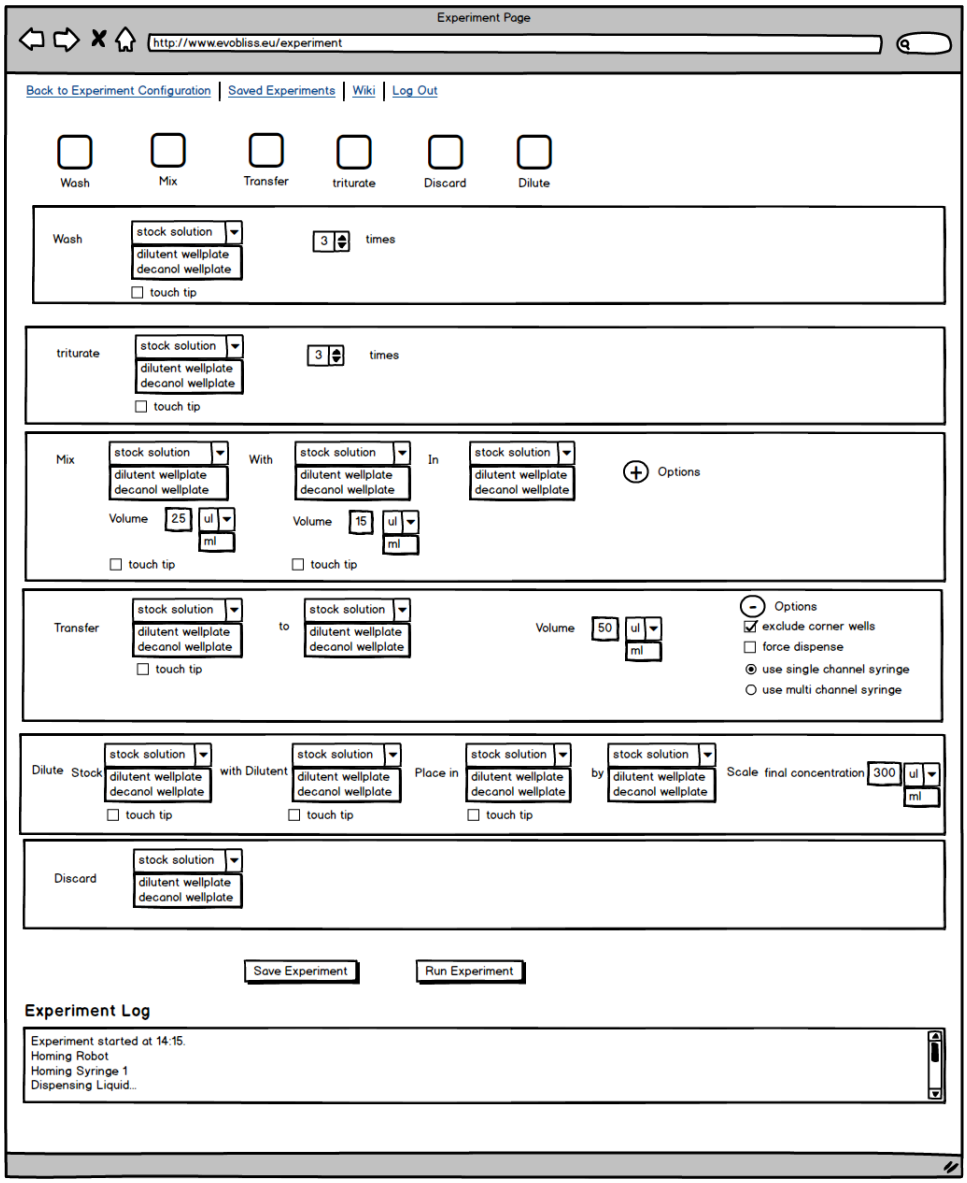


Figure 6.5.2: Iterated version of experiment setup mock up.

Participant ID: _____ Site: _____ Date: ___/___/___

System Usability Scale

Instructions: For each of the following statements, mark one box that best describes your reactions to the user interface *today*.

		Strongly Disagree				Strongly Agree
1.	I think that I would like to use this user interface frequently.	☐	☐	☐	☐	☐
2.	I found this user interface unnecessarily complex.	☐	☐	☐	☐	☐
3.	I thought this user interface was easy to use.	☐	☐	☐	☐	☐
4.	I think that I would need assistance to be able to use this user interface.	☐	☐	☐	☐	☐
5.	I found the various functions in this user interface were well integrated.	☐	☐	☐	☐	☐
6.	I thought there was too much inconsistency in this user interface.	☐	☐	☐	☐	☐
7.	I would imagine that most people would learn to use this user interface very quickly.	☐	☐	☐	☐	☐
8.	I found this user interface very cumbersome/ awkward to use.	☐	☐	☐	☐	☐
9.	I felt very confident using this user interface.	☐	☐	☐	☐	☐
10.	I needed to learn a lot of things before I could get going with this user interface.	☐	☐	☐	☐	☐

Please provide any comments about this user interface:

Figure 6.6.1: SUS score questionnaire.

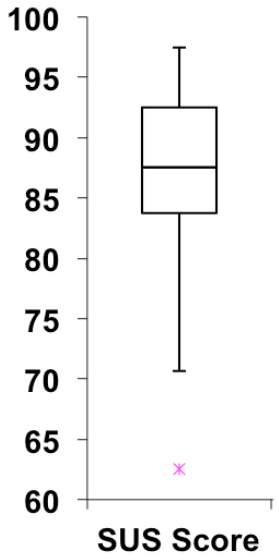


Figure 6.6.2: SUS score boxplot.

value from 5. For example, if the user has chosen the forth choice for question 6, the score 5-4=1 will be calculated for this question. Therefore the scores for each question are from 0 to 4. We add up each score. Then we multiply the total by 2.5. This converts the range of possible values from 0 to 100 instead of from 0 to 40.

We got an average score of 87 in our usability testing as can be seen in Figure 6.6.2.. Another interesting observation was that the users with experience using a liquid handling robot interface gave a higher usability score to the user interface.

6.7 Conclusion

In the preceding chapters, we described a robotic system capable of performing artificial chemical life experiments. Then we verified its applicability to these experiments. We also decoupled user software, and control software by integrating controller hardware into the robotic platform for easy software management. In this chapter, we addressed another requirement for our robotic platform for an in-

tuitive user interface enabling users to use the functionality of our robotic platform effectively.

In order to design an intuitive user interface for our robotic system, we had to go through several tasks. To this end, we observed users and performed user requirement analysis based on user studies. We also took into account important design principles considerations in the user interface design of our robotic platform. We highlighted the strategies that allowed us to develop the user interface for control of our robotic system, as well as the UCD process that we followed to design the interface. We developed a mock up of the user interface based on user observations and user interface design principles. Using a special version of heuristic evaluation designed for robot interfaces we were able to identify key usability problems in the prototype and address them. Finally, we evaluated this user interface by real users of the robotic system.

This intuitive user interface enables our users, who are less inclined to programming the experiments, use the functionality of our robotic platform. Furthermore, by providing a good user experience for the users, they are more willing to use the robotic system. Moreover, providing a good user interface for our platform means less documentation for users, and consequently more efficient use of their time. This also helps us further meet our robotic platform's requirement for affordability.

In this chapter we focused on the the user interface design and evaluation. In the following chapters, we will continue our path towards developing a user interface for liquid handling robots by designing the software architecture required for developing this user interface, and deploying the user interface on the cloud.

7

Software Architecture for an Open Source Multi-platform User Interface for Remote Real Time control of a Robotic System

7.1 Introduction

Up to this point, we have described a robotic system capable of performing artificial chemical life experiments, and have verified its applicability to these experiments. We also decoupled user software, and control software by integrating controller hardware into the robotic platform for easy software management. We then designed an intuitive user interface for this platform described in chapter 6.

As we explained in section 1.8.1, another requirement for our robotic system is design of a software architecture needed for an open source multi platform user

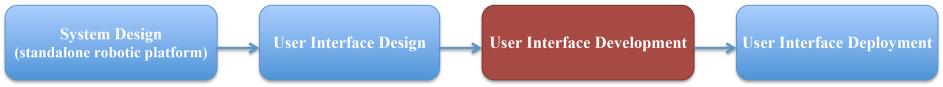


Figure 7.1.1: User interface procedure.

interface to enable remote real time control of such a robotic system. Portability of our robotic system is increased taking advantage of a user interface which can be accessed on different operating systems, or various devices. Furthermore, remote real time control of the robotic system improves accessibility, productivity, and efficiency as users can access the robot at a location at their comfort, and receive notifications regarding experiment progress, or potential warnings, and errors. Moreover, open source software, in contrast to proprietary software for other liquid handling robots, provides a high degree of customization, bug fixes are faster, and is free of cost.

This chapter also comprises our third step towards developing a user interface for liquid handling robots. In chapter 5 and 6, we discussed the implementation of a platform which provides the infrastructure for developing the user interface, and design of the user interface, respectively. As can be seen in Figure 7.1.1, in this chapter we focus on the software architecture required for developing this user interface. In the following chapters we will continue this route by deploying the user interface on the cloud.

This chapter aims at developing an open source multi platform user interface enabling remote real time control of a robotic system. However, in order to develop such a user interface, we have implemented different software architectures, encountered shortcomings, and continued our path for design of a software architecture addressing the deficiencies. Developing a user interface for similar robots would involve similar challenges. For instance, we discuss challenges for remote control, and explain that trivial solutions based on search results are not only the best answer to the problem, but hinder users from innovative approaches. Therefore we present the problems we have run into, how we have solved them, and

lessons learnt. We believe the software architecture design presented in this chapter can be applied for remote real time control of similar robotic systems, and can help users to implement this approach in various applications. Furthermore, a review of different types of software architectures for user interfaces along with their upsides and downsides can provide a good metric for users to choose which software architecture suits them in different scenarios.

In this chapter, we describe different software architectures for user interfaces for our robot, including a Python based user interface, an iOS-based user interface, a LAMP stack web interface, and finally a MEAN stack user interface. We describe the implementation, and discuss advantages, and disadvantages of each user interface. We describe that developing a native user interface in Python has less complexity as our Application programming interface is in Python. However, it has limitations inherent to Python for user interfaces, and it cannot control the robotic platform remotely. We then investigate an iPad user interface for our robot. This user interface enables remote control of our robot, and improves accessibility. However, it cannot be used over Internet from a different local area network, and it cannot be accessed on multiple platforms, i.e. operating systems, or devices. We then investigate a LAMP stack web interface. This interface can control the robot remotely. Finally, we compare these user interfaces for different applications.

The MEAN software stack introduced in this chapter is a new technology used in many start ups for pushing innovative technologies. We believe applying this technology to robots not only contributes to our platform, but will also be advantageous to other robotic systems. This technology can be applied to not only internet of things projects, as simple as controlling appliances, or reading sensors over the internet, but also to applications for complicated robotic systems. We explain how this technology enables fast affordable real time control, enhances code reusability, and testability owing to the front-end, and improves flexibility to store data as we want owing to NoSQL databases. We also explain how we can take advantage of modern web advances, for example classes and modules but run them on old browsers

7.2 Requirements

The software architecture needed for a user interface enabling remote real time control of a robotic system must meet several requirements. The following is the list of the requirements that we will address in this chapter.

1. A native software architecture, in regards to our application programming interface (API), for our graphical user interface application is advantageous, although not a strict requirement. A native software architecture facilitates the communication of different software components between the graphical user interface and the API.

2. The software architecture needs to provide support for remote control of our liquid handling robot.

3. The software architecture should enable web based control of the robotic system. We differentiate between remote control, and web based control in the sense that a user interface may be controlled remotely, and unattended on the same local area network, but not from internet on a different local area network. A software architecture enabling web based control is more powerful and pervasive.

4. The software application should also be cross-platform.

5. We need a software solution for our application that works on different devices, e.g. computer, tablet.

6. Activeness of the user community, good documentation, availability of resources, software packages, and libraries should be taken into account.

7. Easy integration of modern technologies into the software architecture, scalability, code reusability, and maintainability, and easy testability are other important factors. Taking advantage of modern technologies enables us to control our robot real time.

8. Simplicity of software implementation is important.

9. The learning curve should be taken into consideration.

7.3 Python User Interface

Our robotic system comes with a Python based user interface as can be seen in Figure 7.3.1. The users can see information regarding the experiment they are performing, such as droplet size, and droplet center. The users can interact with the experiment based on the displayed information. For instance, if they left click on a point on the image, they will see the distance to the droplet center. This is useful, as users need to know at what distance they need to dispense the salt solution in artificial chemical life experiments. If the users left click on their desired point, the amount of liquid they have assigned will be dispensed or aspirated. The users can choose the amount to dispense with a track bar on top of the display. They can also start or stop recording the experiment when they want.

Being cross-platform, simple syntax, and existence of numerous libraries in various fields of science makes Python a popular programing language among the scientific community. However, when it comes to user interfaces Python is not as powerful as its counterparts. There are several choices for developing a user interface in Python including TkInter, pyqt, and kivy. Tkinter is a Gui (Graphical user interface) Programming toolkit for Python [36]. However, it is basic in comparison to its peers in other programing languages, for instance Autolayout for Swift, or JavaFx for Java, or Layout Editor for Android development. PyQt is a Python binding of the cross-platform GUI toolkit Qt . However, installing the right version of Qt and PyQt can be tedious, and they often break with operating system updates. Furthermore, the license is not GPL. This is in contrast with our open source philosophy for our robotic platform. Kivy is an open source Python library for rapid development of applications that make use of innovative user interfaces . However, to use Kivy all packages need to be be installed on the Kivy core, and not Python, which means reinstalling every required package for Kivy.

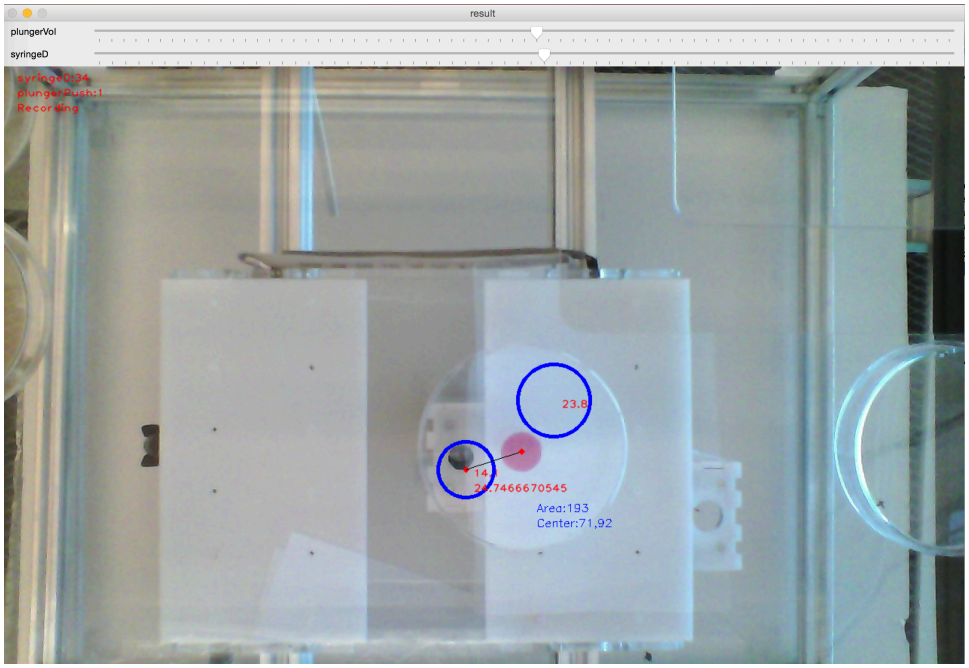


Figure 7.3.1: Python based user interface.

Despite the simplicity of developing a native user interface in Python, it is a common practice for full stack applications in Python, to develop the front end in another language due to the aforementioned shortcomings. Therefore, the user interface is developed in another language, e.g. HTML5, CSS, JavaScript, and a Python based server plays as the back-end to talk with the API (Application programming interface) in Python.

Therefore we use a non Python based user interface for our robotic system, as will be explained in the following sections. However, for simple user interfaces use of a Python framework is suggested as it results in much more simplicity.

7.4 iOS-based User interface

In this section we describe the software architecture for an iOS-based user interface to control our robot, and discuss its advantages and disadvantages.

7.4.1 Description

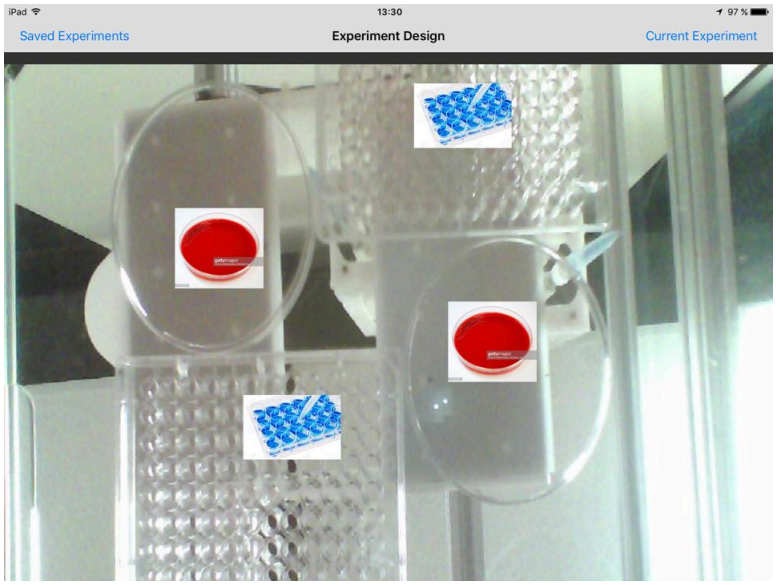
The iPad user interface displays the experimental layer live as the experiment is happening as can be seen in Figure 7.4.1(a). Therefore the users can see the different vessels available to perform their desired experiment. For instance, if the users starts a drag gesture from a Petri dish and ends it on a well plate, a pop over will appear as can be seen in Figure 7.4.1 (b). Then the user will choose experiment parameters, such as amount of liquid in ml or ul, and the syringe to use for transferring liquid. The user has the option to add this step to the experiment, or cancel it.

Having added different experiment steps, the user can reorder steps, edit or delete the steps as can be seen in Figure 7.4.2. The users can then either run the experiment or save the experiment for later use. If the user saves the experiment it will be available later in saved experiments as can be seen in Figure 7.4.1(a).

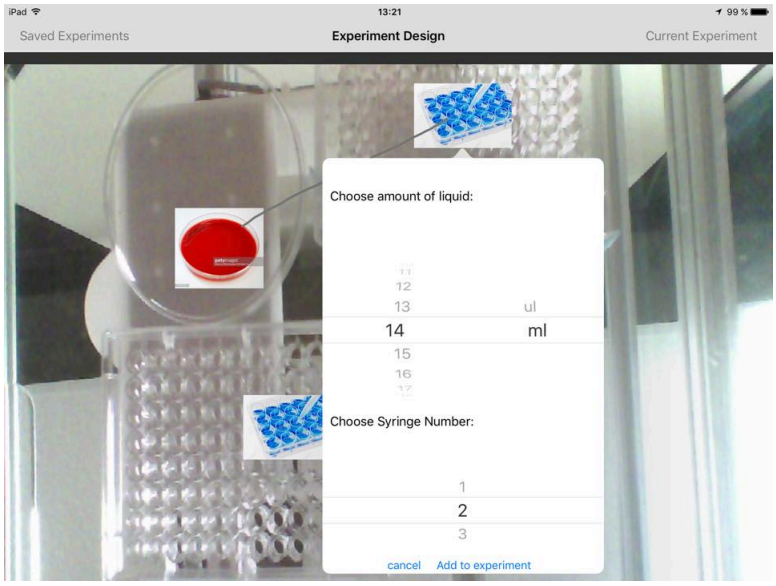
7.4.2 Implementation

The user interface was developed in Swift, a language developed by Apple Inc. for iOS, macOS, and Linux. Swift, the successor of Objective-C, is designed to be concise, and more resilient to erroneous code. We use the Model–View–Controller (MVC) design pattern for implementing our user interface. Therefore, our software is divided into three parts. The view is the presentation of different components on the screen. Our controller defines how our views interact with each other, and is the logic of our application. We use core data as the model for our software architecture for persisting data. We use a protocol-based programming approach to make the communication of different objects possible.

The back-end of our application is implemented using a Python based Server, called BaseHTTPserver. The server is responsible for responding to communication from the client side, namely our iPad user interface. Based on the request from the user interface, the server will communicate with the robot control application programming interface so that the experiment will be run.



(a) User interface showing experimental layer live.



(b) Pop over to choose experiment parameters.

Figure 7.4.1: iPad user interface.

In our Python application programming interface, we have developed a module to translate the coordinates of the touch point on the iPad screen to the corre-

Figure 7.4.2: ipad interface.

sponding coordinate in the coordinate system of the robot. This is done by translating the coordinate system of the iPad screen to the coordinate system of camera, and then translating the coordinate system of the camera from pixels to the coordinate system of the robot in millimeters.

7.4.3 ADVANTAGES

The advantage of an iPad user interface is the capability to control the robot remotely. Therefore the users can run the experiment unattended, allowing them to perform multiple tasks and organize their time. The other advantage is that the robot can move in an arbitrary path drawn by the drag gesture. Therefore, if the user observes an obstacle, the robot can avoid the obstacle if the user defines the path around the obstacle. The other application can be in 3D printing an arbitrary drawn object, as EvoBot comes with a 3D printing module.

7.4.4 DISADVANTAGES

Using the iPad interface has two main disadvantages. The first downside of this software architecture is that only users on the same local area network can access the robot. Therefore, the robot cannot be controlled from user's home over the internet or user's office with a different network. Another issue is that the IP address of the server on our robotic platform has to be entered manually on the iPad user interface. We cannot set a static IP address on the server, as this address may be reserved on the local area network our robotic system is being used, therefore causing conflicts.

7.5 LAMP Stack Web interface

To address the challenges discussed in section 7.4.4, we use LAMP, an archetypal model of web service solution stacks . We used the four open source components in LAMP stack, namely Linux, Apache, MySQL, PHP, to develop our user interface. As can be seen in Figure 7.5.1 [76], Apache HTTP Server, and PHP pro-

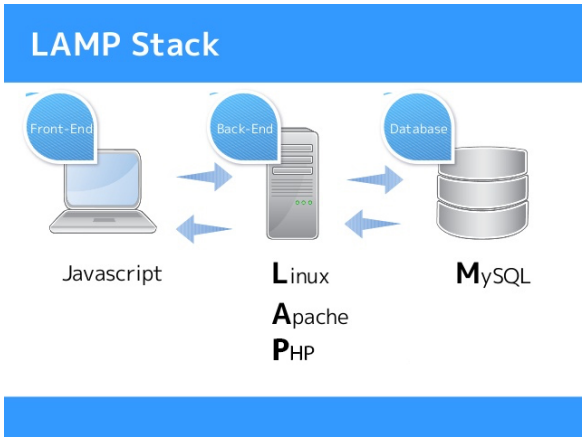


Figure 7.5.1: LAMP stack architecture [76].

gramming language form the back-end technology to serve the user interface. We use MySQL relational database management system (RDBMS) as the database to save experiments.

7.5.1 Implementation

We use Representational state transfer (REST) or RESTful web services to implement our user interface. RESTful APIs support CRUD operations, namely creating data in the database, Reading from the database, Updating the database, and Deleting entries from the database. Therefore the user interface sends an HTTP GET or POST request to the server, and receives a response in JSON, providing a feedback about the request. We used Python-based Flask microframework for the back-end. The frontend was developed in HTML5, CSS, and Javascript.

We use port forwarding to communicate with the Raspberry Pi behind a local area network. Port forwarding is forwarding the incoming traffic to a network to a specific port. Therefore we were able to access the robot even behind a local area network.

7.5.2 ADVANTAGES

The advantage of the web based user interface is that it is accessible on any device connected to internet, therefore the robot can be used remotely.

7.5.3 DISADVANTAGES

We used port forwarding to communicate with our robot. This is not always possible, as when using the robot on a private network, the network does not have a static IP. Therefore we cannot communicate with the robot. In addition, on many public networks, like a LAN network in a University users don't have administrative privileges on the router to enable port forwarding. More importantly, port forwarding is not a good practice due to security issues. It is a bad practice to leave a port always open, and exposes the robot to security vulnerabilities.

7.5.4 Reverse tunneling

Due to the challenges encountered with port forwarding, we used the reverse tunneling concept to communicate with the robot. Reverse tunneling, or back door is used when incoming connections are blocked. Therefore, instead of accepting incoming connections, an outbound connection is opened on the client to a server. Then the user interface connects to the server. Consequently, as the client on the Raspberry Pi is already connected to the server, the server can communicate with the client on the Raspberry Pi.

We managed to implement this software architecture on our local development server. However, when we tried to host it on a hosting service provider, we were not successful. The reason was that installing system level software, e.g. new libraries required for reverse tunneling, on external hosts is often not possible. We needed to get a dedicated server to overcome this limitation, which is less affordable. This was against our philosophy of an affordable robotic platform. This was the reason to look for other solutions to implement reverse tunneling which will be discussed in the following section.

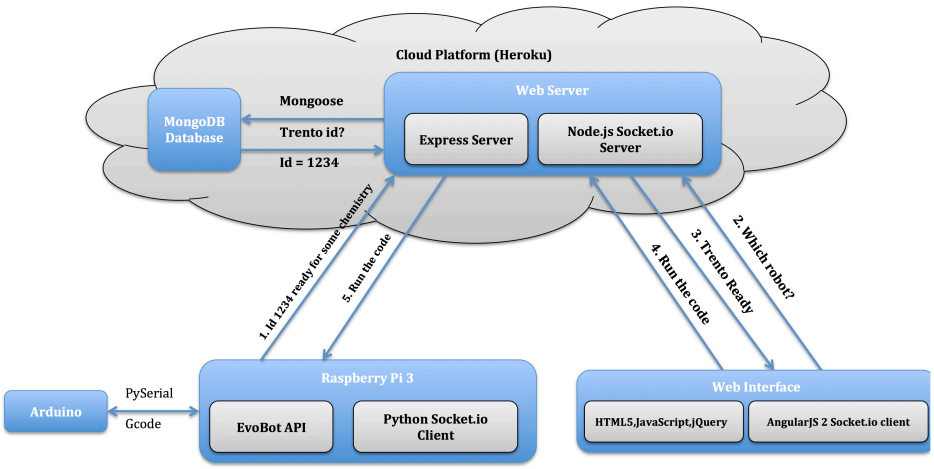


Figure 7.6.1: Software architecture.

7.6 MEAN Stack Web Interface

Due to the challenges mentioned in section 7.5.3, we use MEAN stack as our software architecture. The MEAN stack consists of MongoDB, Express.js, AngularJS, and Node.js. Using the MEAN Stack, it is possible to develop both server side and client side applications in Javascript. Appendix D describes the building blocks of our application, including package management, database architecture, back-end, front-end, unit testing, and styling.

7.6.1 Software architecture

The various software components work together using events, as can be seen in Figure 7.6.1. When the robot is turned on, the Raspberry Pi 3 sends an event to the web socket server using a Python socket.io client, stating that the robot is avail-

able. The Node web socket server keeps this connection alive, waiting for events that need to be responded to by the robot. When the user logs in to the web interface page, based on the credentials provided the robot id is inquired from the database using RESTful APIs. For instance, if the University of Trento user logs in, the assigned id for the user in the database, e.g. id=1234 is inquired from the database. Therefore, the server knows which Raspberry Pi to control among all connected Raspberry Pis. On the web interface page, there is also socket.io client under the hood. When the run experiment button is clicked an event is sent to the Node socket.io server stating that the robot needs to perform a specific action. The socket server then tells the Raspberry Pi 3 that the robot needs to do an experiment, as the connection has been kept alive. Going the other way, when there is a new notification from the robot regarding the experiment, such as completing a task or finishing the experiment, the Python socket.io client will send a notification event to the Node socket.io server. This event is followed by an event sent from the Node server to the web socket.io client, as the second connection from the web socket.io client to the web server has also been kept alive. This results in the notification being updated on the coding web interface. It should be noted that the id used in this example is simplified. However, ids in the database look like $b8:27:eb:22:22:a0$, a sample Raspberry Pi MAC address.

7.6.2 Implementation

In the following sections, we will introduce the final implementation of our MEAN stack user interface taking advantage of the components, and software architecture described in the preceding sections. We provide two different interfaces, namely a coding interface for programing experiments in Python, and a protocol-based interface, to define protocols and run them on the robotic platform without the need to program in Python.

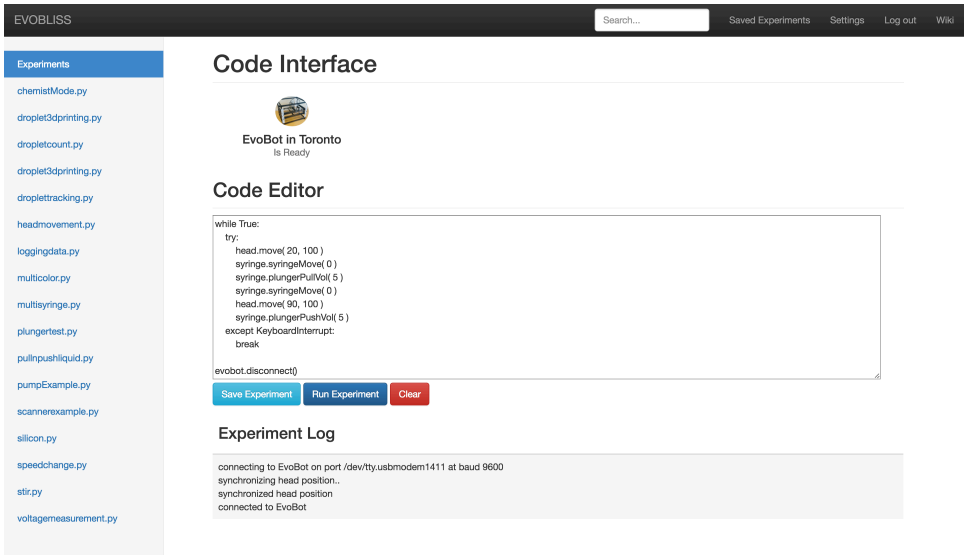


Figure 7.6.2: Code Interface.

CODING USER INTERFACE

The coding web interface, as can be seen in Figure 7.6.2 is a webpage, which allows users to write Python code that will then be run on EvoBot. The users can write Python code in the code editor, as if they were going to run it on a desktop computer. They can either run the experiment on the robot or save it for later use.

The project navigator on the left, as can be seen in Figure 7.6.2, provides access to built-in examples provided in EvoBot's repository. The users can browse files using the menu at the left, and can select them to use as is, or to modify them based on their needs.

The experiment log at the bottom of the screen provides live feedback from the robot while the experiment is happening. It informs users of the start of experiments, the experiment's progress step by step, and will let them know when the experiments finish. The robot will also warn users if there are any issues during the experiments, such as if the user has forgotten to mount a syringe, or power on the robot, or if a vessel is running out of liquid. The status indicator at the top of the screen informs users if the robot is ready to be used or not.

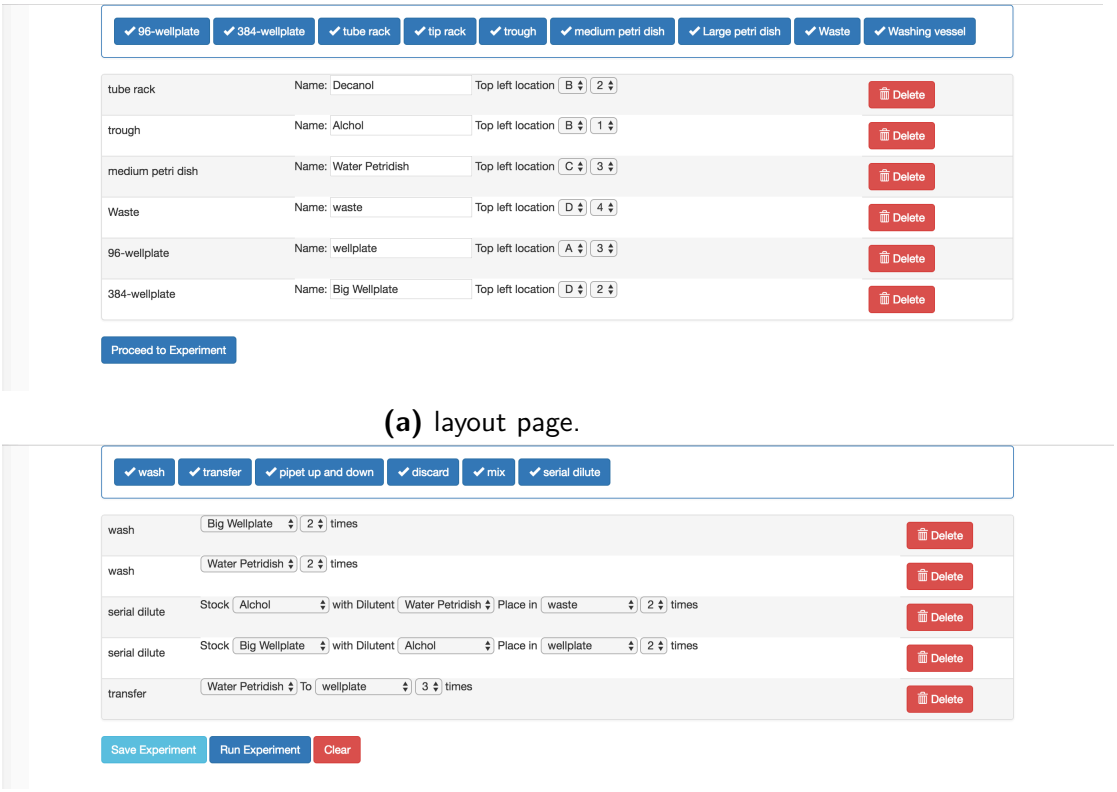
We described the design of the protocol-based user interface in chapter 6. The implementation of the protocol based user interface can be seen in Figure 7.6.3. The user first configures the robot in the layout web page, as can be seen in Figure 7.6.3 (a). At this step, based on what observed on the experimental layer of the robot, the user selects the appropriate reaction vessels and defines the corresponding location on the experimental layer. The users also defines names for vessels for easy access in the next step. Having defined the layout of the experiments the user proceeds to defining the experiment.

As the next step, the user defines the experiment protocol on the protocol web page, as can be seen in Figure 7.6.3 (b). At this stage, the user selects the desired steps in experiment protocol from available operations. For instance, the user can choose wash, transfer, pipet up and down, discard, mix or serial dilute, etc, as the steps of the experiment protocol. Based on the choice, relevant parameters will be inquired from the user. For example, if "serial dilute" is selected, stock vessel, dilutent vessel, and the vessel to place the chemical, as well as the number of times will be inquired. This is different for the choice of a "transfer", where the origin and destination vessel, and the number of times will be asked. Having defined different steps in a protocol, and having set the required parameters, the user can delete or reorder steps. Having finalized the experiment the user can either run or save the experiment for later use.

The users are also provided with an experiment log at the bottom of the screen, displaying live feedback from the robotic platform, including start of the experiment protocol, the step by step protocol progress, the end of the experiment protocol, and warnings, or errors in the experiment protocol.

7.6.3 ADVANTAGES OF WEB USER INTERFACE

The software architecture of the MEAN stack-based user interface addresses our requirement list. This web user interface is usable on multiple platforms, and owing



(a) layout page.

(b) Protocol page.

Figure 7.6.3: protocol-based interface.

to using a Raspberry Pi as integrated controller hardware, software setup is facilitated, and there is an enhancement in affordability. The web user interface also enables user to access EvoBot remotely, unattended, and in an adaptive manner. The experiment can be run over the internet, and the user is provide with live experiment feedback, and is notified when the experiment is finished, when a certain milestone is reached, or if there are errors or warnings.

7.7 Comparison of User Interfaces

In this chapter, we described different software architectures for graphical user interfaces along with their advantages, disadvantages, and the the scenarios they

could be potentially used. In this section, we provide a comparison of the software architectures.

Our choice of software architecture should meet the list of requirements we have described for our user interface. Figure 7.7.1 summarizes the comparison of different software architectures for user interfaces. The figure headings are numbered according to the list of our requirements. Using a native Python user interface results in a less complicated software architecture, as our application programing interface is in Python. However, it does not provide remote control of the robotic platform, and for more complicated user interfaces, like our case, it is not the optimal choice due to limitations of available frameworks. An iPad user interface, on the other hand, provides remote access to our robotic platform. This interface provides live video of the experiment, which can help having a better perception of the experiment conditions. Furthermore, a potential application of this interface can be 3D printing arbitrary shapes drawn by hand. However, this iOS-based user interface is not cross-platform, and cannot be used from the internet or on a different local area network.

The web user interfaces developed in MEAN, and LAMP stack enable remote control of our robotic platform, as well as over the internet. The MEAN stack was a more appropriate choice for our software architecture, regrading the complication, scalability, code reusibility, and the need for integration of modern software technologies.

Considering to choose MEAN or LAMP for an application, there are points to take into account. MEAN Stack is an innovative, new and cutting-edge technology package. It is a frequent choice of new startups willing to push the boundaries in web app development. Significant features of this stack include, being a single language from top to bottom, flexibility in deployment platform, and enhanced run speed, and also in data retrieval, and better scalability. In comparison, LAMP is more traditional. It requires more time and server knowledge to learn the MEAN software stack.

MEAN stack advantages derive from its components. Node.js makes it possi-

Software Architecure for User Interface	1. Native	2. Remote Control	3. Web Control	4. Cross Platform	5. Multi Device	6. User Community	7. Integrating Modern Technology Effort	8. Complication	9. Learning Curve
Python User Interface	x			x		Medium	Medium	Low	Very Shallow
iPad User Interface		x				Large	Medium	Medium	Shallow
LAMP Stack Web Interface		x	x	x	x	Large	Meduim	Very High	Steep
MEAN Stack Web Interface		x	x	x	x	Very Large	Easy	High	Very Steep

Figure 7.7.1: Comparison of different software architectures for user interfaces.

ble to write server side applications using JavaScript, and is faster and more scalable than other server side technologies including LAMP, because of its non-blocking architecture. AngularJS is an open source client-side JavaScript framework developed and maintained by Google. It has solved many development issues regarding reusable, maintainable and testable client-side code. MongoDB is a document-oriented database that allows storing documents in JSON (JavaScript Object Notation), a format that JavaScript natively understands. As both the server-side code, and client-side code are in JavaScript, the required code to serialize and deserialize data is simple, and concise. Non-relational Databases are much faster, and provide an easier method of scaling with higher traffic.

7.8 Conclusion

In this chapter, we addressed another requirement for our robotic system which is an open source multi platform user interface enabling remote real time control of this robotic system. We described different software architectures for graphical user interfaces, along with their advantages, disadvantages, implementation, and finally compared them. Having addressed different challenges, we came up with a software architecture for a user interface which enables users to access our robotic system remotely, unattended, and in an adaptive manner. Owing to this software architecture, the experiment can be run over the internet, and the user is provided with live experiment feedback, and is notified when the experiment is finished, when a certain milestone is reached, or if there are errors or warnings. Remote control of the robotic system enhances effective use of time, and usability. Furthermore, the open source software for the user interface allows users to customize the experiment protocols based on their needs.

Up to this point, we have developed a web user interface which is usable on multiple platforms, i.e. users can access the user interface on any device, such as a tablet, a desktop, a mobile phone, or different operating systems, such as OSX, Windows, or Linux. In addition, as described in chapter 5.1, users don't have to deal with the cumbersome task of installing the numerous packages for the robotic

platform as it is taken care of on the Raspberry Pi 3. Owing to this reasonably priced integrated controller hardware, the price of our platform decreases drastically as a dedicated desktop device is no longer required. In the following chapter, we will accomplish last part of our mission by addressing resource sharing, and deploying the user interface we have developed.

I would rather walk with a friend in the dark,
than alone in the light.

Helen Keller

8

Cloud-based software architecture for a distributed liquid handling system

8.1 Introduction

So far, we have described a robotic system capable of performing artificial chemical life experiments in chapters 2, and 3, and have verified its applicability to these experiments in chapter 5. In chapters 5, 6, and 7.1, we described integrating the controller hardware into the robotic platform, designing the user interface, and the software architecture to implement this design. The resulting web interface consists of two pages. First, the user configures the robot on the layout web page, as demonstrated in Figure 7.6.3 (a) in chapter 7. As the next step, the user defines the experiment protocol on the protocol web page, as demonstrated in Figure 7.6.3 (b). We described the advantages of this web user interface including the

fact that it is usable on multiple platforms, software setup is facilitated, and there is an increase in affordability. The users can access the user interface on any device, such as a tablet, a desktop, a mobile phone, or different operating systems, such as OSX, Windows, or Linux. In addition, users don't have to deal with the cumbersome task of installing the numerous packages for the robot as it is taken care of on the Raspberry Pi 3. Last but not least, the price of our robotic system decreases drastically as a dedicated desktop device is no longer required.

As we explained in section 1.7.8, another requirement for our robotic system is resource sharing and reusability of experiment protocols, the ability to work on the robotic system collaboratively, and parallelizing experiments on different robotic systems. Sharing resources allows users to benefit from experiment protocol templates provided for common experiments, as well as taking advantage of protocol examples developed by other users. Users can also develop their own sample protocols, and share it with their teammates or the rest of the community. This is specifically helpful as our liquid handling robot can be used for numerous applications by different users, therefore taking advantage of sample experiment protocols can save a lot of time for the user community. Collaboration on robotic platforms provides novel opportunities for researchers. Providing the users with the capability of working on experiments collaboratively means multiple users can work on the same experiment simultaneously. On the user interface, they would see the changes other users are making to the experiment protocol real time. They can modify the same experiment as a team, or receive notifications regarding experiment progress. Moreover, users can continue to work on the same experiment on another machine. On the other hand, parallelizing experiments improves efficiency, specifically for artificial chemical life experiments, as several long lasting experiments are performed on multiple platforms.

In this chapter, we describe a cloud based software architecture for deploying our user interface. A cloud based implementation is a paradigm shift from single user single platform concept to single user multi platform, multi user single platform, and multi user multi platform approaches. A single user multi platform paradigm, i.e. a user being able to control several robotic systems at the same time,

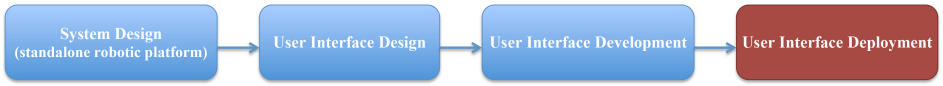


Figure 8.1.1: User interface procedure.

and run the same code on multiple robots, allows for a high degree of parallelism. A multi user single platform, i.e. several users can work on the same robot simultaneously, provides a great potential for collaboration on the robotic platform. A multi user multi platform approach, i.e. several users, e.g. a team, being able to work on multiple robots, enhances resource sharing, and reusability of experiment protocols.

This chapter also comprises our final step towards developing a user interface for liquid handling robots. In chapter 5 and 6, we discussed the implementation of a platform which provides the infrastructure for developing the user interface, and design of the user interface, respectively. In chapter 7.1, we described the software architecture required for developing this user interface. As can be seen in Figure 8.1.1, in this chapter we will deploy this user interface on the cloud.

Deploying application on the cloud is a new trend, providing new opportunities including real-time collaboration, example templates, file synchronization, version control, and more resources in terms of storage or computation. Large companies are moving in this direction fast, providing cloud computing solutions. Amazon EC2 (Elastic Compute Cloud), Google Cloud Platform, Apple iCloud, Rackspace Cloud, Microsofts Azure, Intel Hybrid Cloud Computing are few examples. There are diverse successful services based on cloud computing, and new cloud-based technologies are emerging. ShareLaTeX , and Overleaf are online LaTeX editors. They don't require installation of LaTeX and packages, and enable real-time collaboration, version control, and provide numerous LaTeX templates. Dropbox simplifies creating, sharing and collaborating by offering cloud storage, and file synchronization. iCloud ensures having the latest versions of documents, photos, and notes, on different devices and platforms, e.g MacOS, and iOS. The Construct

Simulators allows using a web browser to simulate robots, e.g. run Gazebo or Webots simulation environment with full ROS support, without any installation required, and with high processing power [18].

In spite of the great potential of cloud computing, this paradigm has not been applied to a real distributed liquid handling robotic system. We believe cloud solutions provide new opportunities for distributed robotic applications. Specifically, when computation requirements outweigh communication requirements, use of cloud computation provides many advantages as the computation can be outsourced to an external powerful resource, and only the computation processing results need to be communicated. Furthermore, as cloud computing is a modern technology, in this chapter we gather and organize resources, and introduce technologies, therefore same software architecture, and deployment techniques will be reproducible by other users to be applied to different applications for robotic systems.

8.2 Requirements

In order to deploy our application we have to meet several requirements. The following is the list of the requirements that we will address in this chapter.

1. The first requirement is to choose the right technology for deploying our application.

2. Once we have chosen the technology, we need to choose an engine to deploy our application.

3. We need to choose a registry to store, maintain, and distribute our application, and manage changes.

4. As our software architecture introduced in chapter 7 requires multiple components, we need to build a flexible, independently deployable software system that is easily scalable.

5. As our application consists of multiple components, making modifications on these components and deploying them to test the changes can be time consuming and tedious. Furthermore, we need to address the dependency between components. For instance components depending on each other, should start in the correct order.

6. As we add features to our multi component application, we need to make sure providing a new feature, does not break the old functionality of our application. Therefore another requirement is to guarantee our application can always be deployed on the production server.

7. As our application consists of different services, it is critical to avoid integration problems, as making modifications to a component and not merging them immediately to the code base may make different components not in sync.

8. Easy scalability of services is another requirement for our application. Based on the need, some of the services may need to scale up at different levels. We need to manage this scaling easily.

9. Finally, we need to deploy our application on a cloud platform.

8.3 Technology Choice for Deploying our Application

In this section, we address the 1st point on the requirement list which is choosing the right technology for deploying our application. We describe the traditional server concept, and compare it with hypervisor-based virtualization, and container-based virtualization.

8.3.1 The Traditional Server Concept

Figure 8.3.1 shows the traditional server model . Before the advent of virtualization, big server racks were used. As can be seen in the Figure 8.3.1, underneath

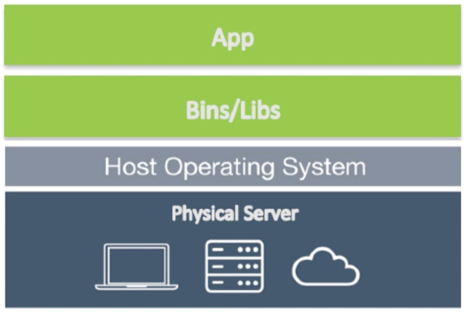


Figure 8.3.1: Before virtualization.

lies the physical server on which the desired operating system is installed. The applications are run on top of the operating system. In this approach, each physical machine would only run one application.

There are several downsides with this model [72]. The first issue is the huge cost associated with this approach as physical machines need to be purchased in order to deploy each application. These commercial servers can be very expensive. We might end up only using a fraction of CPU, or memory of the machine. The rest of the resources are wasted. However, the hardware has to be bought in advance. Another downside is slow deployment time. The process of purchasing and configuring new physical servers can take a long time, especially for big organizations. The other drawback is hard migration of applications to new servers from a different vendor. For instance, migrating from Dell servers to IBM servers can be time consuming, and requires a significant amount of configuration change, and manual intervention [55].

8.3.2 HYPERVISOR-BASED VIRTUALIZATION

Figure 8.3.2 shows the traditional model of virtualization, also called hypervisor-based virtualization technology . Underneath lies the physical server, on which the desired operating system is installed. On top of the operating system a hypervisor layer is introduced which allows installation of multiple virtual machines on a single physical machine. Each virtual machine can have a different operating system,

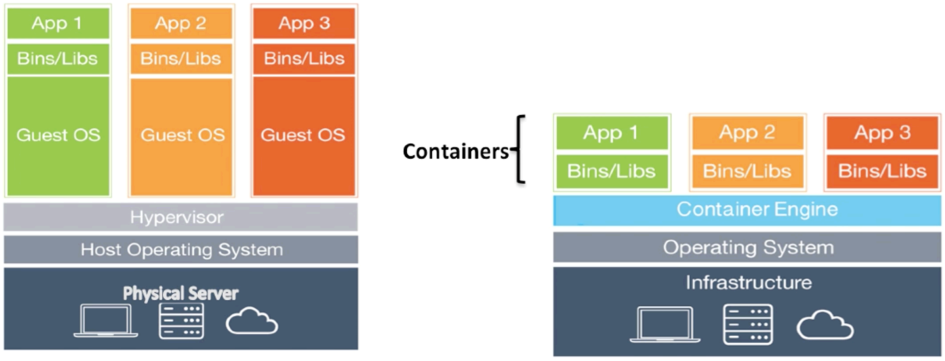


Figure 8.3.2: Hypervisor-based virtualization on the left, and Container-based virtualization on the right.

for instance Ubuntu on one, and CentOS on the other. Owing to this model, we can run multiple operating systems on the same physical machine, and each operating system can run a different application. VMware , and VirtualBox are among the popular providers of hypervisors.

Using hypervisor-based virtualization technologies has several advantages [80]. Firstly, it is more cost effective. Each physical machine is divided into multiple virtual machines, and each one uses its own CPU, memory, and storage resources. Users pay only for the compute power, storage, and other resources they use. This is a cheap solution for users, and they don't have any upfront commitment. Secondly, scaling is easy with virtual machines deployed in the cloud environment. If new instances of an application are required, this can be easily achieved by deploying virtual machines in the cloud, in contrast to ordering and configuring physical servers. With this approach, the time required to scale applications is reduced drastically to minutes. This results in great increase in agility for organizations.

However, there are still drawbacks with hypervisor-based virtualization technology [54]. The first limitation originates from kernel resource duplication. Each virtual machine needs an operating system installed. This entire operating system has its own memory management, device drivers, daemons, etc. Even though, there can be different kernels, still a lot of the core functionality of the operating

system, e.g. Linux, is replicated [45]. Therefore this approach is not efficient as an entire operating system is required to run the application. Secondly, application portability is not guaranteed as running virtual machines on different types of hypervisors is still work in progress.

8.3.3 CONTAINER-BASED VIRTUALIZATION

The final approach is container-based virtualization technology as demonstrated in figure 8.3.2 on the right. Underneath lies the server, which can be either a physical machine or a virtual machine. The operating system is installed on the server. On top of the operating system, a container engine is installed which allows running multiple guest instances. Each guest instance is called a container. In each container, the application, and all the libraries required for that application are installed.

The key difference between container-based virtualization model and hypervisor-based virtualization model is the replication of the kernels [98]. In the traditional model each application is running its own copy of the kernel, and the virtualization happens at the hardware level. In container-based virtualization model there exists only one kernel, which supplies different binaries at run time to the applications running in isolated containers. So the containers will share the base run time kernel, which is the container engine. In this model, the virtualization happens at the operating system level. Containers share the hosts operating system. Therefore this approach is more efficient, and lightweight, as the operating system is not duplicated.

One advantage of containers is run time isolation [108]. Most applications depend on third party libraries. For instance, if we need to run two different versions of Python applications (Python 3, and Python 2.7), running them on the same virtual machine without introducing any conflicts may be challenging. By leveraging containers, we can easily isolate the two run time environments. Therefore if application A (Python Socketio client) requires Python 3, we can install Python 3 in container A, and run the application in container A. On the other hand, if applica-

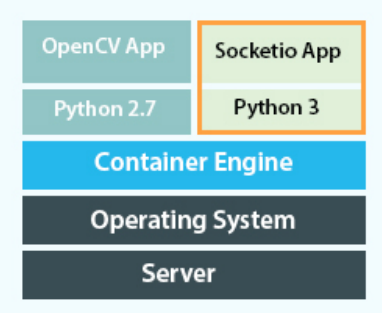


Figure 8.3.3: Isolation.

tion B (OPENCV 3) requires Python 2.7, we can install Python 2.7 in container B, and run the application in container B. In this way, we have two containers on the same machine running two different applications, with different Python versions as can be seen in Figure 8.3.3.

Container-based virtualization has advantages over hypervisor-based virtualization [107]. Container-based virtualization does not create an entire operating system. Instead, only the required components are packed up inside the container with the application. So containers consume less CPU, RAM, and storage space than virtual machines. That means we can have more containers running on one physical machines than virtual machines. Secondly, the deployment speed is faster as containers house the minimal requirement for running the application which can boot up as fast as a process. A container can be several times faster to boot than a virtual machine. Thirdly, portability is guaranteed with containers, as containers are independent self-sufficient application bundles. They can run across machines without compatibility issues.

8.4 Implementation Metrics

In this section, we explain how we address the different requirements for deploying our application discussed earlier. We demonstrate the choice of a container engine for implementing our application. We commence by describing the docker con-

cept, and the choice of a public registry for our images. We then explicate why we use a microservice architecture, and how we automate the docker work flow. We then demonstrate the necessity of running unit tests inside docker containers, and how we use continuous integration in our work flow. We then show how our application can be scaled up for any number of users, taking advantage of docker swarm. We conclude this section by comparing different cloud platforms we used for deploying our application.

8.4.1 Choice of Deployment Engine

In this section, we address the 2nd point on the requirement list which is the choice of an engine to deploy our application. There exist different container engines for this purpose, including Core OS' rkt, Cloud Foundry's Garden, Kubernetes, and docker engine. For this purpose, we use docker engine, an implementation of a container based engine, to deploy our application on the cloud [85]. Docker engine is based on two concepts, namely images, and containers. Images are templates that can only be read. Containers are run time objects built from images and can be written to. In computer science terminology, if we think of images as classes, containers are instances of the class.

The reason for the choice of docker engine is rapid application deployment, version control and component reuse, portability across machines, and minimal overhead of containers.

Figure shows the layout of our image stack, and the corresponding container 8.4.1 . As can be seen, the image stack is made up of layers. The base image can be seen at the bottom, on which other layers required for the application are built. The layers incorporate the differences in the file system. Ubunto forms the base for the container's file system, we then add emacs, and Node image layers on top of the base layer. We spin up containers, thin lightweight layers on top that we can modify, from our image stack. Inside the container, we update the operating system, install required packages, and dependencies, and run required commands. The container encapsulates the environment, and can be ported for reuse [47]. It

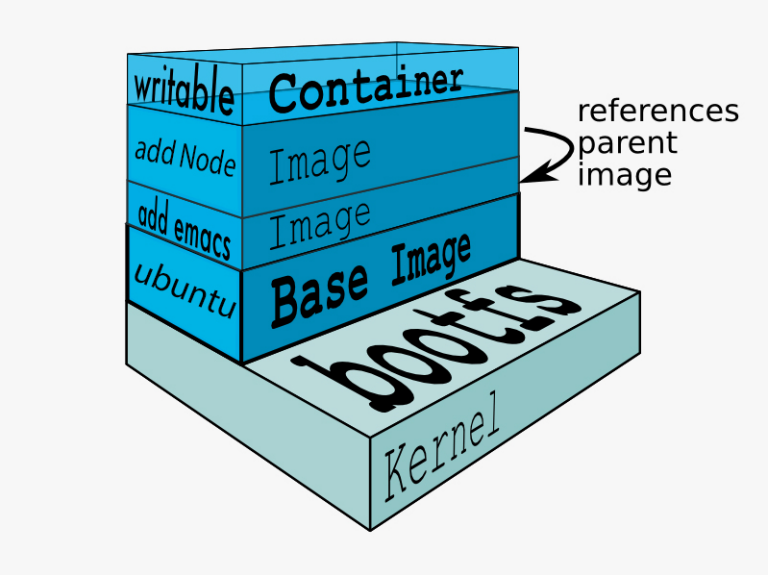


Figure 8.4.1: Docker image layers.

should be noted that multiple containers can be spun up from the image stack.

The major difference between container and an image is the top writable layer. Therefore multiple containers can share access to the same underlying image, and have their own data state. Docker pulls the images layer by layer.

It should be noted that choice of docker is one possible implementation of our application. However, the business logic of our application, regardless of choice of implementation can be generalized to similar applications. It means analogous applications can implement microservices in a vendor agnostic way.

8.4.2 Public Registry Choice

This section addresses the 3rd point on the requirement list which is the need to choose a registry to store, maintain, distribute our application, and manage changes. There exist different hosted registry options for this purpose, including Quay.io, Artifactory, Google Container Registry, and Docker's public registry. For this purpose, we use Docker's public registry, called docker hub , to store our im-

ages, as well as pulling images for production when they have passed unit tests. Docker hub is a registry, similar to NPM registry, containing many images in repositories from either official image providers or the open source user community. For instance, we use the official Node image. Therefore, instead of installing Node, we just need to download it.

The reason for this choice is ease of container management and deployment without the need for building complex pipelines in Docker hub. Furthermore, other registries are aimed for larger enterprises providing high availability.

Docker repository is a collection of different docker images with the same name, that have different tags. Each tag usually represents a different version of the image. As images can become large in size, images are designed to be composed of layers of other images, allowing a minimum of data to be sent when transferring images over the network [103].

8.4.3 Microservices

In order to address the 4th point on the requirement list which is building a flexible, independently deployable software system that is easily scalable, we use a microservice architecture for deploying our application. The microservice architecture allows different services to run independently, and communicate with each other when necessary [62]. The communication between the services is on their own local network, so they don't expose any ports to the outside world, which is a great security advantage. In addition, when using microservice architecture, adding new services or scaling up applications is easy, as we don't need to change the software architecture of the whole application, but only add the new service.

Our microservice architecture consists of three services, namely, mongoservice to manage our database, socketservice to communicate the events on the user interface with the web server, and dockerapp service to provide RESTful APIs, and a socketio web server. To deploy the microservice architecture, we need to link the containers running the mongodb, socketio, and application services, so they can discover each other, and communicate over a secure tunnel. Linking containers

can be done explicitly at container build time. However, we used docker's capability to intelligently infer, and establish service links by following service name consistency [100]. For instance as can be seen in Figure **??**, all services have the same network name "mynet", and we define this network as a bridge network in network declaration.

8.4.4 Automating the work flow

In order to meet the requirement for the 5th point on the requirement list which is easy testing of application components modification in deploy environment, and also satisfying component dependency requirements, we automate our workflow. Our application is not a single component container, and as described in section 8.4.3 it consists of mongoservice, socketservice, and dockerapp services. Therefore, we automate the work flow by taking advantage of docker compose, a tool to setup docker environments for docker orchestration [95]. Therefore we can automatically build images, start all the services, and create the links among services.

8.4.5 Unit tests inside containers

In order to meet the 6th point on the requirement list which is maintaining the core functionality when adding new functionality, we perform unit tests to guarantee our application can be deployed on the production server. The unit tests verify if the basic functionality of our application is not broken. It is specially important for production as developers might not have the exact environment for developing the application, MacOS or Windows for example . Therefore the application may work for one developer, but not the other. As a consequence, we take advantage of docker containers to perform the unit tests. To perform tests, a clean container is created swiftly in which unit tests are run. This ensures the reliability of the application. The same container will be used for production. This guarantees the same image can run on the production server.

Running unit tests in the docker comes with advantages and disadvantages [96]. The upside is that the developer doesn't need to wait for the validation of unit tests

every time he makes changes as the tests are run in docker containers in a parallel process. The downside is the image size grows as unit tests are included in the image. However, this is not an important issue, as our unit tests are not large. Furthermore, it should be taken into account that unit tests are only as good as the design of the tests.

8.4.6 Continuous integration

In point 7 of the requirement list, we described that another requirement for our application is to avoid integration problems. To address this requirement, we use continuous integration, a software engineering practice to test changes to the code base immediately [63]. Continuous integration aims to provide fast feedback in order to prevent introducing flaws into the repository [82]. This is specially useful as image repositories contain larger files than code repositories. Therefore it is helpful to test the code before pushing the image to the repository.

The work flow of our application starts by pushing code to a repository, e.g. Github, or Bitbucket. We use circleci as our continuous integration server. The continuous integration server monitors the repositories we define, in our case our github repository. As a change is made in our codebase, the continuous integration server figures it out, and triggers a build of the image. Therefore a clean image is built upon each commit, incorporating the code pushed to the repository, along with unit tests. Then the unit tests we have defined are run automatically. If they are not successful we receive an email, so that we can revert the commit. On the other hand, if the tests are run successfully, the image is pushed to the docker hub. The image is tagged with the commit hash value, so that images can be easily mapped to their corresponding commit. This image on the docker hub will be pulled to run on the production server, e.g. Digitalocean.

8.4.7 Scaling Up the Application

In point 8 of the requirement list, we described the requirement for easily managing the scaling up of our application. In order to address this requirement, we

use docker swarm for docker orchestration in our application. Docker swarm is based on the idea of dividing docker engines into clusters, and assigning a swarm manager for each cluster [73]. Then the swarm manager, in lieu of each individual docker engine, is responsible for scheduling containers into all docker engines in the cluster. The swarm manager knows the status of all nodes in the cluster based on a discovery method. We define the scheduling strategy on the swarm manager so that it defines the order of docker containers being run. When we decide to run new containers, the docker manager distributes the containers among docker machines, therefore the workload is divided by the nodes in the swarm. This is useful for scaling our application.

8.4.8 Deploying on Cloud Platforms

This section addresses the 9th point on the requirement list which is deploying our application on a cloud platform. For this purpose, we compare deploying applications on two different types of cloud platforms, namely Digital Ocean, and Heroku.

Deploying on Digital Ocean

We deployed our application on Digital Ocean , a cloud computing platform. Deploying our application on other cloud platforms, such as Amazon EC2 (Elastic Compute Cloud), Google Cloud Platform, Rackspace Cloud, Microsofts Azure is also possible, and similar. We have chosen Digital Ocean because of the good pricing to address our affordability requirement described in section 1.7.4. In Digital Ocean terminology, each docker machine is referred to as a droplet.

Deploying on Heroku

Our application can also be deployed on Heroku , a cloud platform which is different form others. Deploying applications on Heroku is possible using Heroku toolbelt. Although deploying applications is easier with Heroku toolbelt, other

platforms are more suitable for applications that need to be scaled up due to pricing considerations.

8.4.9 Example Scenario

A useful application of the cloud implementation can be a multi user multi platform scenario. The users of our robotic system are distributed in seven different universities across the world. Each university owns a robot. Furthermore, in each university several users may use the same robot. On the other hand, due to shared research interests in different universities, some common experiments are performed in different labs. In a multi user multi platform scenario, the users in University of Trento for instance set up a team profile. Thereafter, when each user of the University of Trento accesses the cloud based user interface he/she is regarded as a member of that team. The user will set up the experiment on the protocol based web user interface. When setting the experiment up, each user can see the modifications the other users are making to the experiment real time, and he/she can also modify the experiment himself/herself. Therefore multiple users can work on the experiment collaboratively, and simultaneously. Having set the experiment up, the users will choose which robots to run the experiment on. Therefore each of the users in the University of Trento's team for example can choose to run the same experiment on different robots, e.g. the robots of University of Trento, University of Prague, and IT University of Copenhagen, etc. All the users in a team will receive notifications regarding the progress of an experiment. Each user is also capable of including or excluding robots that the experiment will be run on.

8.5 Conclusion

In this chapter, we addressed another requirement for our robotic system which is resource sharing and reusability of experiment protocols, the ability to work on the robotic system collaboratively, and parallelizing experiments on different robotic systems. We demonstrated a cloud based software architecture for deploying our

user interface. A cloud based implementation is a paradigm shift from single user single platform concept to single user multi platform, multi user single platform, and multi user multi platform approaches. This chapter also comprised our final step towards developing a user interface for liquid handling robots. We demonstrated that deploying an application on the cloud is a new trend, and provides a lot of new opportunities. However, this paradigm has not been applied to real distributed robotic systems.

We described that in order to deploy our application we had to meet several requirements. We had to choose the right technology for deploying our application. Container-based virtualization was the answer to this question, providing run time isolation, portability, and consuming less resources. We also needed to choose an engine to deploy our application. We used docker engine, an implementation of a container based engine, to deploy our application on the cloud. We also needed to choose a registry to store, maintain, and distribute our application, and manage changes. To address this requirement, we used docker hub to store our images, as well as pulling images for production when they have passed unit tests. Furthermore, as our software architecture introduced requires multiple components, we need to build a flexible, independently deployable software system that is easily scalable. In order to address this requirement, we use a microservice architecture for deploying our application, which allows different services to run independently, and communicate with each other when necessary. Moreover, as our application consists of multiple components, making modifications on theses components and deploying them to test the changes can be time consuming and tedious. Furthermore, we need to address the dependency between components. In order to meet these requirements, we automate our workflow, by taking advantage of docker compose, a tool to setup docker environments for docker orchestration

Another requirement is to guarantee our application can always be deployed on the production server. To address this requirement, we used unit testing inside containers. The unit tests verify if the basic functionality of our application is not broken. Moreover, as our application consists of different services, it is critical

to avoid integration problems. To address this requirement, we used continuous integration, a software engineering practice to test changes to the code base immediately. Yet, easily scalability of services was another requirement for our application. In order to address this requirement, we used docker swarm for docker orchestration in our application. Docker swarm is based on the idea of dividing docker engines into clusters, and assigning a swarm manager for each cluster. Finally, we needed to deploy our application on a cloud platform. We chose Digital Ocean, a cloud computing platform to deploy our application on the cloud.

9

Future Work & Conclusion

9.1 Future Work

9.1.1 Robot-facilitated evolution in artificial chemical life research

One future direction for further research is robot-facilitated evolution in artificial chemical life research. As the reagents, and parameters affecting artificial chemical life experiments are varied, machine learning techniques can be applied to explore this space. For instance, if the speed of a droplet composed of several chemicals needs to be maximized, evolutionary algorithms can be used to define the proportion of different chemicals in the droplet. As the robot is capable of processing droplet behaviors, in this case droplet speed, the fitness function can be evaluated. Therefore, the compound formula for the desired behavior can be optimized. Researchers in University of Glasgow have started working in this direction [67], and

potentially there is much more possibilities to explore in this regard.

9.1.2 Experiments with different classes of chemicals

Another line of work can be to use the developed robotic platform for new classes of chemicals. Most of the experiments we have performed with the robotic platform have been performed with low viscosity chemicals like water. Investigating requirements for other classes of chemicals, for instance high viscosity chemicals, and exploring possibilities that performing these experiments provide can be another direction for research. For instance, researches at University of West of England are exploring the possibilities of 3D printing microbial fuel cells, a technology that works by converting biomass into electricity with the aid of microbes, with such a robotic platform.

9.1.3 Dissemination of our work, and Promoting open source hardware and software

Another future work can be introducing the robotic platform we have built to the relevant community, and promoting open source hardware and software. Coordination of a summer school for students and researchers in relevant disciplines to artificial chemical life research, and robotics can be useful to disseminate the results of our work. This summer school can aim at students from different backgrounds such as artificial life, chemistry, microbiology, artificial intelligence, robotics, and mechatronics. Such a summer school can open cross-disciplinary discussions about the potential of artificial chemical life, and participants can discover what new possibilities such a robotic platform provides for their research in areas such as dynamic chemistry or microbial systems. Also publishing parts of this thesis that have not been published yet, and participation in scientific venues can help further disseminate the scientific results of our work.

9.2 Conclusion

In this thesis, we introduced artificial chemical life experiments where the behavior of a motile droplet with respect to relative positioning of reagents is of interest. These droplet experiments are long-lasting, low-throughput experiments in comparison to ordinary liquid handling experiments that are high throughput and don't have to be under constant observation. These experiments require precise timing and relative positioning of reagents with respect to motile droplets. Furthermore, experiment parameters, such as distance between droplet and reagent, time of adding reagent, and concentration of liquids, affecting droplet behavior play an important role in these experiments.

Lab automation has extensively helped chemists perform repetitive chemical experiments. Owing to lab automation, the behavior of the experiment can be verified over many runs and sufficient data can be collected to build a model or attain statistically significant results. Therefore liquid handling technologies alleviate the hardships of dealing with the dull, time consuming task of repeating experiments. It also prevents the numerous types of errors humans may introduce into the experiment. Furthermore, lab automation is beneficial when performing experiments with hazardous chemicals.

The characteristics of artificial chemical life experiments, and the potential advantages automation could bring to these experiments were our motivation to develop a robotic platform capable of performing theses experiments. Artificial life experiments are particularly well-suited for automation because they often stretch over long periods of time, possibly hours, and often require that the human takes action in response to observed events such as change in droplet velocity, size, count, shape, or clustering/ declustering of multiple droplets.

The basic required functionality for our solution was routine liquid handling functionality, i.e aspirating a specified amount of liquid from one reaction vessel, e.g. a Petri dish, or well plate, and dispensing the desired amount of liquid in another reaction vessel. Moreover, in artificial chemical life experiments, based on the observed behavior we must interact with the experiment. Therefore another

requirement for our platform was to be able to perform feedback based experiments. In addition, another requirement for users performing artificial chemical life experiments was to be able to analyze experiment data online, and use this feedback to interact with the experiment real time. Moreover, having developed such a robotic platform, we needed to verify that our solution is able to perform diverse liquid handling experiments. Furthermore, the robotic platform needed to be affordable so it can be used in small size laboratories.

Addressing the requirements, we described the robotic system we have developed in order to automate a large class of droplet experiments in artificial chemical. To enable performing these feedback-based experiments, we have integrated computer vision into the design of a liquid handling robot, and developed the required software functionality. Computer vision enables the robotic system to detect relevant changes, either in individual droplet behavior, e.g. change in droplet area, position, speed, direction, acceleration, color, shape, number of droplets, or group droplet behavior, e.g. droplets clustering or declustering. Having detected the specified behavior change, precise automation enables the robot to interact with the experiment, e.g. dispense or aspirate a chemical at a specific point relative to a droplet center. Moreover, the platform we have developed can record accurate experiment data online, therefore enabling users to interact with the experiment in real time. We also demonstrated the robotic platform can be employed in various artificial chemical life experiments. Furthermore, being based on open-source 3D printer technology, the cost of the platform is reduced significantly.

Other requirements for our robotic platform were related to usability. Short setup time, and ease of software management for the robotic system was a requirement, as installing, and managing software libraries required for feedback based experiments on different hardware, and operating systems would be difficult. Another requirement was an intuitive user interface enabling users to utilize the functionality of our robotic platform. Remote real time control through an open source multi platform user interface was another requirement for our robotic platform in order to make it more portable on different operating systems, and devices. Finally, the ability to share the resources among different users was another require-

ment for our robotic platform in order to enhance code reusability.

To address these requirements, we detached user software for programing experiments from control software. Therefore we developed a standalone robotic platform with integrated controller hardware in order to ease package management. Consequently, regardless of the choice of the system users want to use, they don't need to install any libraries. This platform is also more affordable owing to elimination of the need for an external computer. In addition, we came up with a user interface design for liquid handling robots that is intuitive, as well as being easily extensible for new experiment protocols in biological, and chemical laboratories. We also developed an open source multi platform user interface enabling remote real time control of this robotic system empowering users to access our robotic system unattended. Owing to this user interface, the experiment can be run over the internet, and the user is provided with live experiment feedback, and is notified when the experiment is finished, when a certain milestone is reached, or if there are errors or warnings. Remote control of the robotic system enhances effective use of time, and usability.

Finally, we demonstrated a cloud based software architecture for deploying our user interface. A cloud based implementation is a paradigm shift from single user single platform concept to single user multi platform, multi user single platform, and multi user multi platform approaches. This cloud-based software architecture improves resource sharing and reusability of experiment protocols, and enables working on the robotic system collaboratively, and parallelizing experiments on different robotic systems.

References

[1] 4lab™ automated low volume liquid handling system. URL http://www.4ti.co.uk/instrumentation/automated-liquid-handling. visited on 2017-08-15.

[2] A business case for usability. URL http://www.userfocus.co.uk/articles/usabilitybenefits.html. visited on 2016-11-15.

[3] Cost benefits evidence and case studies. URL http://www.usabilitynet.org/papers/Cost_benefits_evidence.pdf. visited on 2016-11-15.

[4] Andrew alliance. URL http://www.andrewalliance.com. visited on 2016-11-15.

[5] Beeline from htz. URL http://www.htz.biz/samplers_main.htm. visited on 2017-08-15.

[6] Benchsmart™ 96 from mettler toledo. URL http://dk.mt.com/dk/da/home/products/pipettes/high-throughput-platforms/benchsmart-96-pipetting-system.html. visited on 2017-08-15.

[7] Freedom evo 200 from tecan. URL http://diagnostics.tecan.com/products/liquid_handling_and_robotics/freedom_evo_clinical/specifications. visited on 2017-08-15.

[8] Pipetmax automated liquid handling system from gilson. URL http://www.gilson.com/en/pipetmax#.WZZISqoZNPs. visited on 2017-08-15.

[9] Piro from dornier ltf. URL http://www.dornier-ltf.com/products. visited on 2017-08-15.

[10] Year in review 2016: Sbc comparison chart. URL https://www.element14.com/community/docs/DOC-83855/l/element14-year-in-review-2016-sbc-comparison-chart. visited on 2016-11-15.

[11] Starplus from hamilton. URL https://www.hamiltoncompany.com/products/automated-liquid-handling/liquid-handling-workstations/microlab-star-line. visited on 2017-08-15.

[12] Cytomat™ 10 c450 series from thermo fisher. URL https://www.thermofisher.com/order/catalog/product/51031166. visited on 2017-08-15.

[13] X150 from xiril ag. URL http://www.speciation.net/Database/Instruments/Xiril-AG/X150-Automatic-Workstation-;i1808. visited on 2017-08-15.

[14] Ttp labtech mosquito hts. URL https://abiobot.org/. visited on 2017-08-15.

[15] Biomek 4000 from beckman coulter. URL http://www.beckman.com/liquid-handling-and-robotics/instruments/liquid-handlers/biomek-4000. visited on 2017-08-15.

[16] Biomek i7 integrated system. URL http://ca.beckman.com/liquid-handling-and-robotics/instruments/liquid-handlers/biomek-i-series. visited on 2017-08-15.

[17] Bravo from agilent technologies. URL http://www.agilent.com/en-us/products/automation-solutions/automated-liquid-handling/bravo-automated-liquid-handling-platform. visited on 2017-08-15.

[18] The construct, just simulate. URL http://www.theconstructsim.com. visited on 2016-11-15.

[19] epmotion® 96. URL http://www.selectscience.net/products/epmotion-96/?prodID=197236. visited on 2017-08-15.

[20] Ecmascript compatibility table. URL http://kangax.github.io/compat-table/es6/. visited on 2016-11-15.

[21] Evobliss software. URL https://bitbucket.org/afaina/evobliss-software.git. visited on 2016-11-15.

[22] Ezmate from arise biotech. URL http://www.arisebio.com.tw/product_Series.asp. visited on 2017-08-15.

[23] Beaglebone black - geekbench browser, . URL http://browser.primatelabs.com/geekbench2/2356217. visited on 2016-11-15.

[24] measure a computer's processor and memory performance, . URL https://www.geekbench.com. visited on 2016-11-15.

[25] C.h.i.p. computer - geekbench browser, . URL https://browser.primatelabs.com/geekbench2/2624979. visited on 2016-11-15.

[26] Raspberry pi 3 model b - geekbench browser, . URL http://browser.primatelabs.com/geekbench2/2596704. visited on 2016-11-15.

[27] Nano-plotter 2.1 from gesim. URL https://gesim-bioinstruments-microfluidics.com/113/. visited on 2017-08-15.

[28] Ttp labtech mosquito hts. URL http://ttplabtech.com/liquid-handling/mosquito_hts/. visited on 2017-08-15.

[29] Professional digital design for mac. URL https://www.invisionapp.com. visited on 2016-11-15.

[30] Browser market share. URL https://www.netmarketshare.com/browser-market-share.aspx?qprid=0&qpcustomd=0. visited on 2016-11-15.

[31] Micro b processor from bee robotics. URL http://www.beerobotics.com/Products/micro-b-processor/. visited on 2017-08-15.

[32] Cell explorer gene pro from perkinelmer. URL http://www.perkinelmer.com/product/cell-explorer-gene-pro-cegp1000. visited on 2017-08-15.

[33] Cybi-selma. URL https://www.analytik-jena.de/en/lab-automation/products-lab-automation/liquid-handling/cybior-selma.html. visited on 2017-08-15.

[34] Solo from hudson robotics. URL http://hudsonrobotics.com/products/liquid-handling/solo/. visited on 2017-08-15.

[35] S-pipette from apricot design. URL http://www.apricotdesigns.com/html/s-pipette.htm. visited on 2017-08-15.

[36] Tkinter - python wiki. URL https://wiki.python.org/moin/TkInter. visited on 2016-11-15.

[37] User experience terms for bcs foundation certificate in ux. URL https://quizlet.com/149096704/user-experience-terms-for-bcs-foundation-certificate-in-ux-flash-cards. visited on 2016-11-15.

[38] Cybi-well vario from analytik jena. URL https://www.analytik-jena.de/en/lab-automation/products-lab-automation/liquid-handling/cybior-well-family/cybior-well-vario.html. visited on 2017-08-15.

[39] Versa 10 from aurora biomed, . URL http://www.aurorabiomed.com/versa-10/. visited on 2017-08-15.

[40] Versa 110 from aurora biomed, . URL http://www.aurorabiomed.com/versa-110/. visited on 2017-08-15.

[41] Viaflo assist from integra. URL https://www.integra-biosciences.com/en/electronic-pipettes/viaflo-assist. visited on 2017-08-15.

[42] Vertical pipetting station from agilent technologies. URL http://www.agilent.com/en-us/products/automation-solutions/automated-liquid-handling/vertical-pipetting-station. visited on 2017-08-15.

[43] Comparing popular layer-based and code-based prototyping tools. URL http://uxmag.com/articles/comparing-popular-layer-based-and-code-based-prototyping-tools. visited on 2016-11-15.

[44] Mikio Aoyama. Persona-scenario-goal methodology for user-centered requirements engineering. In *15th IEEE International Requirements Engineering Conference (RE 2007)*, pages 185–194. IEEE, 2007.

[45] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[46] M. Bedau, J. S. McCaskill, N. Packard, and S Rasmussen. Living technology: Exploiting life's principles in technology. *Artificial Life*, (1):89–97, 2010.

[47] David Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84, 2014.

[48] Randolph G. Bias and Deborah J. Mayhew. *Cost-Justifying Usability: An Update for the Internet Age*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. ISBN 0120958112.

[49] Stefan Blomkvist. Persona–an overview. *Retrieved November*, 22:2004, 2002.

[50] Åsa Blomquist and Mattias Arvola. Personas in action: ethnography in an interaction design team. In *Proceedings of the second Nordic conference on Human-computer interaction*, pages 197–200. ACM, 2002.

[51] John Brooke. Sus: A retrospective. *J. Usability Studies*, 8(2):29–40, February 2013. ISSN 1931-3357. URL http://dl.acm.org/citation.cfm?id=2817912.2817913.

[52] Frederick P Brooks Jr. *The mythical man-month, anniversary edition: Essays on software engineering*. Pearson Education, 1995.

[53] Charles E Brown. Coefficient of variation. In *Applied multivariate statistics in geohydrology and related sciences*, pages 155–157. Springer, 1998.

[54] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*, pages 5–13. Ieee, 2008.

[55] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.

[56] Filippo Caschera, Steen Rasmussen, and Martin M Hanczyc. An oil droplet division–fusion cycle. *ChemPlusChem*, 78(1):52–54, 2013.

[57] J. Cejkova, M. Novak, F. Stepanek, and M. M. Hanczyc. Dynamics of chemotactic droplets in salt concentration gradients. *Langmuir*, 30(40): 11937–11944, 2014. ISSN 0743-7463. doi: 10.1021/la502624f.

[58] J. Y. C. Chen, E. C. Haas, and M. J. Barnes. Human performance issues and user interface design for teleoperated robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(6):1231–1245, Nov 2007. ISSN 1094-6977. doi: 10.1109/TSMCC.2007.905819.

[59] Edward Clarkson and Ronald C Arkin. Applying heuristic evaluation to human-robot interaction systems. In *Flairs Conference Proceedings*, pages 44–49, 2007.

[60] Kerstin Dautenhahn. Methodology and themes of human-robot interaction: a growing research field. *International Journal of Advanced Robotic Systems*, 2007.

[61] A Faına, F Nejatimoharrami, K Stoy, P Theodosiou, B Taylor, and I Ieropoulos. Evobot: An open-source, modular liquid handling robot for nurturing microbial fuel cells. 2016.

[62] Bob Familiar. *Microservices, IoT and Azure: Leveraging DevOps and Microservice Architecture to deliver SaaS Solutions*. Apress, 2015.

[63] Martin Fowler and Matthew Foemmel. Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, page 122, 2006.

[64] Vitaly Friedman. 10 principles of effective web design. *Smashing Magazine*, 31, 2008.

[65] Michael A. Goodrich and Alan C. Schultz. Human-robot interaction: A survey. *Found. Trends Hum.-Comput. Interact.*, 1(3):203–275, January 2007. ISSN 1551-3955. doi: 10.1561/1100000005. URL http://dx.doi.org/10.1561/1100000005.

[66] Juan Manuel Parrilla Gutierrez, Trevor Hinkley, James Taylor, Kliment Yanev, and Leroy Cronin. Hardware and software manual for evolution of oil droplets in a chemo-robotic platform. *CoRR*, abs/1411.1953, 2014. URL http://arxiv.org/abs/1411.1953.

[67] Juan Manuel Parrilla Gutierrez, Trevor Hinkley, James Ward Taylor, Kliment Yanev, and Leroy Cronin. Evolution of oil droplets in a chemorobotic platform. *Nature communications*, 5, 2014.

[68] Maik Hadorn, Eva Boenzli, Kristian T Sørensen, Harold Fellermann, Peter Eggenberger Hotz, and Martin M Hanczyc. Specific and reversible dna-directed self-assembly of oil-in-water emulsion droplets. *Proceedings of the National Academy of Sciences*, 109(50):20320–20325, 2012.

[69] Martin M Hanczyc. Metabolism and motility in prebiotic structures. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 366 (1580):2885–2893, 2011.

[70] Martin M Hanczyc, Taro Toyota, Takashi Ikegami, Norman Packard, and Tadashi Sugawara. Fatty acid chemistry at the oil-water interface: Self-propelled oil droplets. *Journal of the American Chemical Society*, 129(30): 9386–9391, 2007.

[71] Martin M. Hanczyc, Juan Manuel Parrilla Gutierrez, Arwen Nicholson, Kliment Yanev, and Kasper Støy. Creating and maintaining chemical artificial life by robotic symbiosis. *Artificial Life*, 21(1):47–54, 2015. doi: 10.1162/ARTL_a_00151. URL http://dx.doi.org/10.1162/ARTL_a_00151.

[72] M David Hanson. The client/server architecture. *Server Management*, page 3, 2000.

[73] Bukhary Ikhwan Ismail, Ehsan Mostajeran Goortani, Mohd Bazli Ab Karim, Wong Ming Tat, Sharipah Setapa, Jing Yuan Luke, and

Ong Hong Hoe. Evaluation of docker as edge computing platform. In *Open Systems (ICOS), 2015 IEEE Confernece on*, pages 130–135. IEEE, 2015.

[74] Leander Kahney. *Inside Steve's brain*. Penguin, 2009.

[75] Anne Kaikkonen and Virpi Roto. Navigating in a mobile xhtml application. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '03, pages 329–336, New York, NY, USA, 2003. ACM. ISBN 1-58113-630-7. doi: 10.1145/642611.642669. URL http://doi.acm.org/10.1145/642611.642669.

[76] Keisuke Kato. Lamp (linux, apache, mysql, php). URL http://www.slideshare.net/kiskkato/mean-stack-44934547. visited on 2016-11-15.

[77] Gary King, Robert O Keohane, and Sidney Verba. The importance of research design. *Rethinking Social Inquiry: Diverse Tools, Shared Standards*, pages 181–92, 2004.

[78] Philippe Kruchten. From waterfall to iterative development—a challenging transition for project managers. *Rational Edge, Rational Software*, 2001.

[79] William Lidwell, Kritina Holden, and Jill Butler. *Universal principles of design, revised and updated: 125 ways to enhance usability, influence perception, increase appeal, make better design decisions, and teach through design*. Rockport Pub, 2010.

[80] Flavio Lombardi and Roberto Di Pietro. Secure virtualization for cloud computing. *Journal of Network and Computer Applications*, 34(4):1113–1122, 2011.

[81] James McLurkin, Jennifer Smith, James Frankel, David Sotkowitz, David Blau, and Brian Schmidt. Speaking swarmish: Human-robot interface design for large swarms of autonomous mobile robots. In *AAAI Spring Symposium: To Boldly Go Where No Human-Robot Team Has Gone Before*, pages 72–75, 2006.

[82] Mathias Meyer. Continuous integration and its tools. *IEEE software*, 31 (3):14–16, 2014.

[83] N. Millard. Learning from the 'wow' factor — how to engage customers through the design of effective affective customer experiences. *BT Technology Journal*, 24(1):11–16, 2006. ISSN 1573-1995. doi: 10.1007/s10550-006-0016-y. URL http://dx.doi.org/10.1007/s10550-006-0016-y.

[84] Clive R Newton, Alex Graham, et al. *PCR*. Number Ed. 2. Bios Scientific Publishers Ltd, 1997.

[85] Jeff Nickoloff. *Docker in Action*. Manning Publications Co., 2016.

[86] J Nielsen. Jakob nielsen's alertbox, august 5, 2001: First rule of usability? don't listen to users, 2001.

[87] Jakob Nielsen. Iterative user-interface design. *Computer*, 26(11):32–41, 1993.

[88] Jakob Nielsen. Severity ratings for usability problems. *Papers and Essays*, 54, 1995.

[89] Jakob Nielsen. Progressive disclosure. *Jakob Nielsen's Alertbox*, 2006.

[90] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *SIGCHI Conference Proceedings*, CHI '90, pages 249–256. ACM, 1990. doi: 10.1145/97243.97281. URL http://doi.acm.org/10.1145/97243.97281.

[91] Charlotte EJ Pittman. Involving users in the design process. *Master of Science in Organizational Dynamics Theses*, page 20, 2008.

[92] Joshua Porter. *Designing for the Social Web, eBook*. Peachpit Press, 2010.

[93] Jenny Preece, Yvonne Rogers, and Helen Sharp. *Interaction Design: Beyond Human-Computer Interaction*. John Wiley & Sons, Inc., New York, NY, USA, 2015. ISBN 0471402494.

[94] Eric Ries. *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses*. Crown Books, 2011.

[95] Carlos Sanchez. Scaling docker with kubernetes. *Website. Available online at http://www. infoq. com/articles/scaling-docker-with-kubernetes*, 2015.

[96] Dmitry Savchenko and Gleb Radchenko. Microservices validation: Methodology and implementation. In *CEUR Workshop Proceedings. Vol. 1513: Proceedings of the 1st Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists (Ural-PDC 2015).—Yekaterinburg, 2015*. Уральский федеральный университет, 2015.

[97] Ken Schwaber. Scrum development process. In *Business Object Design and Implementation*, pages 117–134. Springer, 1997.

[98] Gopal Prasad Sharma, Sharanjit Singh, Amardeep Singh, and Ramanpreet Kaur. Virtualization in cloud computing. 2016.

[99] Aaron Steinfeld, Terrence Fong, David Kaber, Michael Lewis, Jean Scholtz, Alan Schultz, and Michael Goodrich. Common metrics for human-robot interaction. In *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-robot Interaction*, HRI '06, pages 33–40, New York, NY, USA, 2006. ACM. ISBN 1-59593-294-1. doi: 10.1145/1121241.1121249.

[100] Joe Stubbs, Walter Moreira, and Rion Dooley. Distributed systems of microservices using docker and serfnode. In *Science Gateways (IWSG), 2015 7th International Workshop on*, pages 34–39. IEEE, 2015.

[101] Desirée Sy. Adapting usability investigations for agile user-centered design. *Journal of usability Studies*, 2(3):112–132, 2007.

[102] Warren H Teichner and Marjorie J Krebs. Laws of visual choice reaction time. *Psychological review*, 81(1):75, 1974.

[103] James Turnbull. *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.

[104] R Victor. Iterative and incremental development: A brief history. *IEEE Computer Society*, pages 47–56, 2003.

[105] M. Wohlsen. Flow robotics, . URL http://www.flow-robotics.com. visited on 2016-11-15.

[106] M. Wohlsen. This robot could make creating new life forms as easy as coding an app., . URL http://www.wired.com/2014/11/opentrons-bio-robots. visited on 2016-11-15.

[107] Miguel G Xavier, Marcelo V Neves, Fabio D Rossi, Tiago C Ferreto, Timoteo Lange, and Cesar AF De Rose. Performance evaluation of container-based virtualization for high performance computing environments. In *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, pages 233–240. IEEE, 2013.

[108] Yuping Xing and Yongzhao Zhan. Virtualization and cloud computing. In *Future Wireless Networks and Information Systems*, pages 305–312. Springer, 2012.

[109] Holly A. Yanco, Jill L. Drury, and Jean Scholtz. Beyond usability evaluation: Analysis of human-robot interaction at a major robotics competition. *Human–Computer Interaction*, 19(1-2):117–149, 2004. doi: 10.1080/07370024.2004.9667342.

[110] Robert H Yolken. Elisa: enzyme-linked immunosorbent assay. *Hospital practice*, 13(12):121–127, 1978.

Appendices

A

Supplementary Videos

1. Oleic anhydride droplet moving in an aqueous system containing a pH sensitive dye.

   https://www.youtube.com/watch?v=S7Ln-fvuUvo

2. Oleic anhydride droplet moving chemotactically towards a solution of higher pH.

   https://www.youtube.com/watch?v=fu5CrRQWpJs

3. Dynamics of Chemotactic Droplets in Salt Concentration Gradients

   https://www.youtube.com/watch?v=P5uKRqJIeSs

4. Nine droplet behaviours discovered, displaying a great deal of complex, unexpected emergent behaviours.

   http://www.nature.com/article-assets/npg/ncomms/2014/141208/ncomms6571/extref/ncomms6571-s3.mov

5. Sensor input feedback for aspirating droplet. A moving droplet is aspirated by the syringe module when the droplet speed goes below a specified threshold.

   https://youtu.be/Mc-eoBajV8U

6. The feasibility of experiments when sensor input feedback affects group droplet behavior (Clustering Experiment)

   https://youtu.be/kbly17Fkfko

B

Universal Robots User Interface

Figure B.0.1 is the main page of the robot software called PolyScope. On this interface, programming the robot begins by selecting ”program robot”.

In order to add commands to the program tree, the users need to click on the “Structure” tab at the top to select required commands. On the structure tab, as can be seen in Figure B.0.2, users can add motions, waypoints (the waypoints tell the robot where it has to position the arm), and other commands. For instance, clicking the “Move” button will insert a waypoint. The location of the waypoint can be either defined by moving the robot arm manually or by using the user interface to define the waypoint by clicking the “Move” Tab. The move tab can be seen in Figure B.0.3. Users can click on the blue arrows to move the robot or push the black Freedrive button on the top back of the touch screen to move the arm manually.

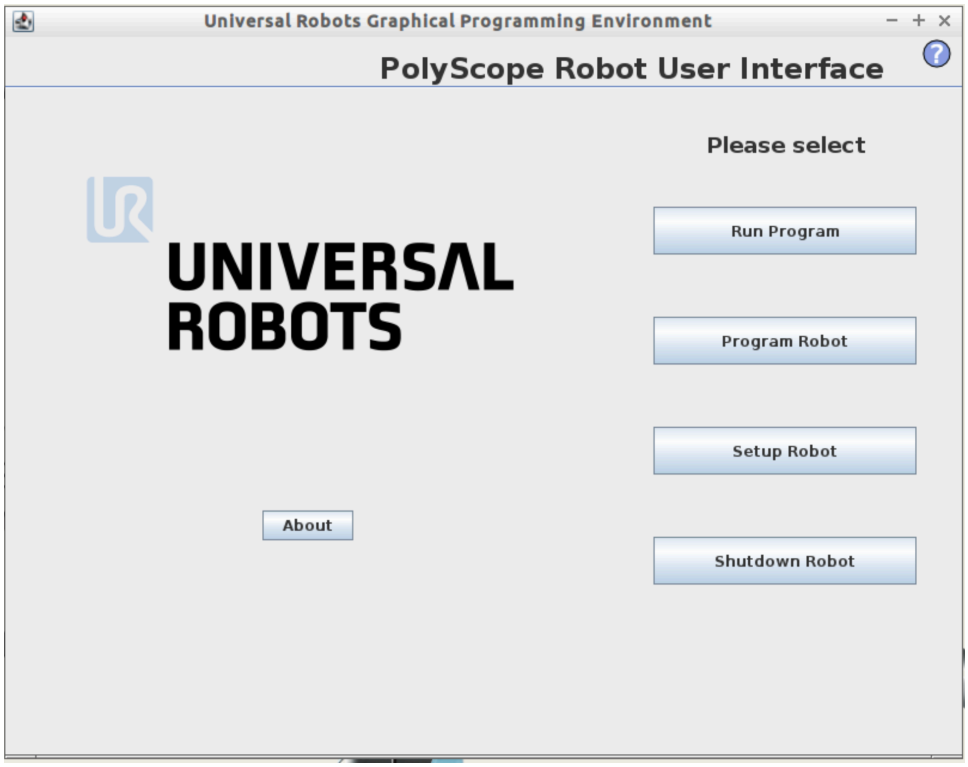


Figure B.0.1: Universal Robots PolyScope home screen.

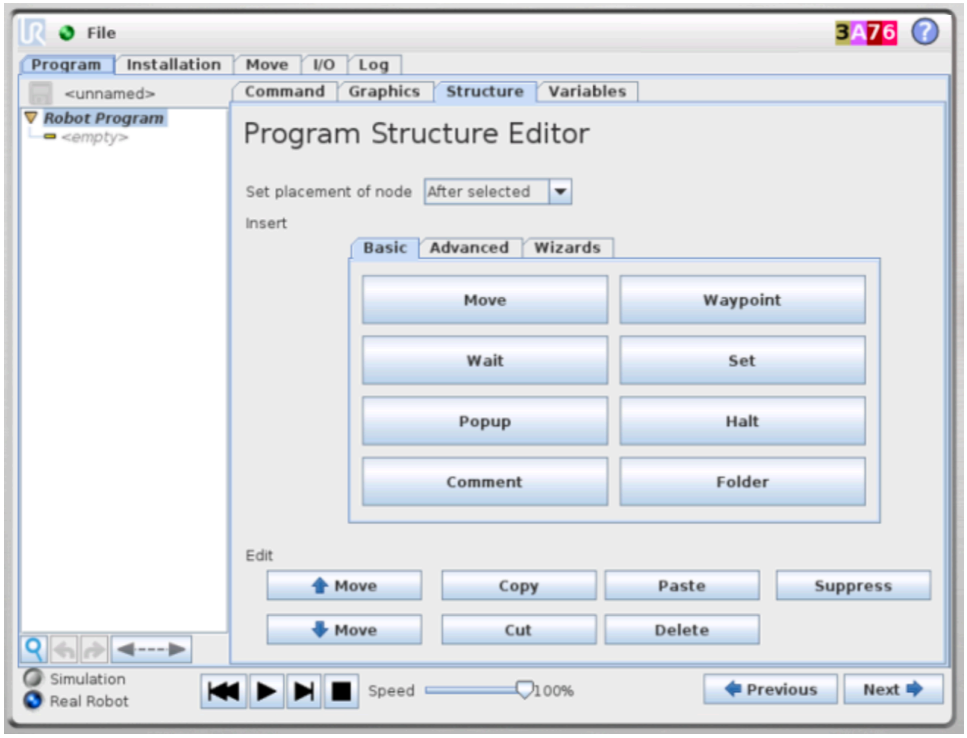


Figure B.0.2: Structure tab.

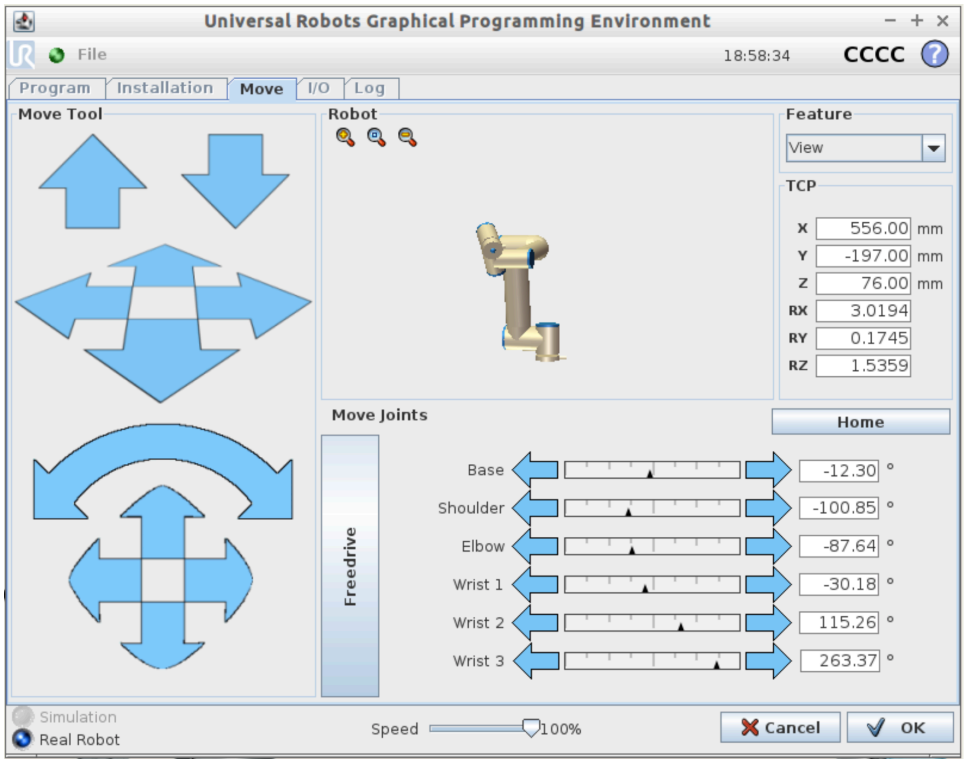


Figure B.0.3: PolyScope move robot interface.

C

Overclocking the Raspberry Pi

In order to overclock the Raspberry Pi, we had to increase the processor, gpu, and sdram frequency for better performance. They can be modified on the Raspberry Pi in the /boot/config.txt file. The "gpu_freq" setting includes "core_freq", "v3d_freq", "h264_freq", and "isp_freq" all in one, so we only need to set the "gpu_freq" and it pulls them all up together. We started with lower values and if it was stable then we increased only one setting at a time, so that we could track which one is causing a problem. Following are the values we have changed in the /boot/config.txt file with the Raspberry Pi working stable:
arm_freq=1370
over_voltage=5
gpu_freq=580
sdram overclock
sdram_freq=580
sdram_schmoo=0x02000020
over_voltage_sdram$_p$ = 6
over_voltage_sdram$_i$ = 4
over_voltage_sdram$_c$ = 4

It should be noted that changing the "up_threshold" on the Raspberry Pi would cause it not to increase the speed until CPU utilization gets above a certain thresh-

old, for instance %75.

D

MEAN Stack Implementation

The following sections describe the building blocks of our MEAN stack application, including package management, database architecture, back-end, front-end, unit testing, and styling.

D.0.1 NPMJS

NPM is the package manager for JavaScript. Using NPM registry, it is possible to find, share, and reuse packages of code from hundreds of thousands of developers, and assemble them in powerful new ways. NPM is the largest existing ecosystem of open source software.

D.0.2 NoSQL Database

NoSQL encompasses a wide variety of different database technologies that were developed in response to the demands presented in building modern applications. Relational databases were not designed to cope with the scale and agility challenges that face modern applications, nor were they built to take advantage of the commodity storage and processing power available today .

Figure D.0.1: MongoDB.

MongoDB

MongoDB (originated from hu***mongo***us) is a NoSQL database. MongoDB is open-source, free, cross-platform, and is based on the concept of documents, and collections. The data structure of a MongoDB database can be seen in Figure D.0.1. A collection in MongoDB is similar to a table in a SQL based database, and a document in MongoDB is similar to a row in a SQL based database. Documents look like JSON. Therefore MongoDB is very flexible for different data types, and database entries don't need to be the same in MongoDB. A field in a MongoDB document is similar to a row in a SQL based database.

Mongoose

Mongoose provides a straight-forward, schema-based solution to model our application data. It includes built-in type casting, validation, query building, and busi-

ness logic hooks out of the box.

API key

The web user interface of our robotic system requires to allow multiple users to control their corresponding robots simultaneously, and also needs to allow a single user to control multiple robotic systems simultaneously. We have made this possible by assigning each robotic system a unique API key to be identified from others. The unique MAC address of each Raspberry Pi is used as the API key. Owing to the use of API key, users can login with their credentials, and access their own robot. This is useful as multiple copies of EvoBot are used in seven laboratories in different countries.

D.0.3 Back-end

Our back-end is implemented using RESTful APIs, and web sockets owing to the power of Nodejs, and Express.

Nodejs

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.

Express

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

socket

The back-end for our user interface has been implemented using web sockets, in addition to common RESTful APIs. The reason is that the functionality provided by traditional RESTful APIs is not sufficient for our application, as the link is terminated after the connection with RESTful APIs. In our situation, the robot and the interface need to actively listen for events, and respond accordingly. The communication between the back-end, front-end and the database will be further explained in section 7.6.1.

D.0.4 Front-end

In this section, we describe how the frontend of our application is developed using ES6, TypeScript, AngularJS 2, Webpack, Jasmine, Karma, and Bootstrap.

ES6 vs ES5

ECMAScript 2015 (ES2015) is the specification of JavaScript approved in 2015. ES2015 is also called ES6 as it is the sixth edition of the standard. ES5, also called Vanilla JavaScript is the older version of JavaScript which has been around since 2009.

There are several features in ES6 of which we have taken advantage in our application, including default parameters, template literals, multi-line strings, destructuring assignment, enhanced object literals, arrow functions, promises, block-scoped constructs Let and Const, classes, and modules. Since the standard is fairly new, the browsers don't support ES6 code fully, e.g. Chrome 56 supports 96% of ES6 features, and IE supports 11% of ES6 features as can be seen in Figure D.0.2 [20]. However, IE has 23% of market share as can be seen in Figure D.0.3 [30], and even Chrome does not support 100% of the features. For this reason, we use tools like babel, or webpack module loaders that can convert ES6 code to ES5 code. Therefore we write our application in ES6 code, and then it is transformed to ES5 code that can run in every browser.

Typescript

TypeScript is a superset of JavaScript improving the language by providing optional static types and classes for object-oriented programming. TypeScript was developed by Anders Hejlsberg, lead architect of C ♯ from Microsoft. For our application, we write ES6 code in TypeScript. The TypeScript compiler is responsible for compiling the TypeScript code to ES5 understandable by all browsers.

AngularJS 2

AngularJS 2 is an open-source web application framework backed by Google. The final version of Angular was released in September 2016, after around two years of active development. AngularJS 2 extends HTML vocabulary, therefore the resulting environment is expressive. As can be seen in Figure D.0.4, AngularJS 2 comprises of four main building blocks; namely components, directives, services,

Figure D.0.2: ECMAScript compatibility table.

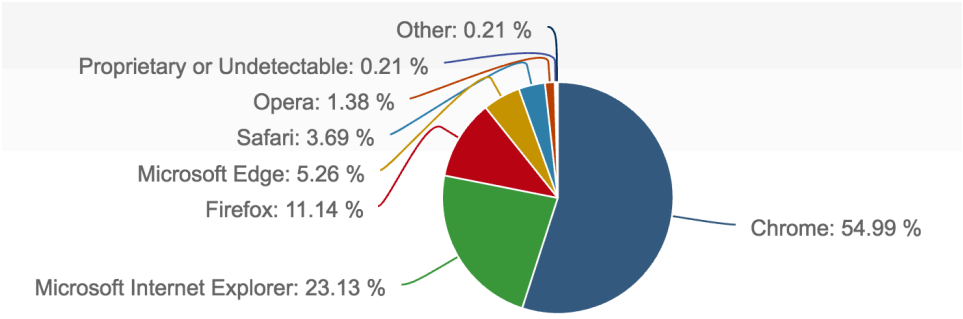


Figure D.0.3: Browser market share.

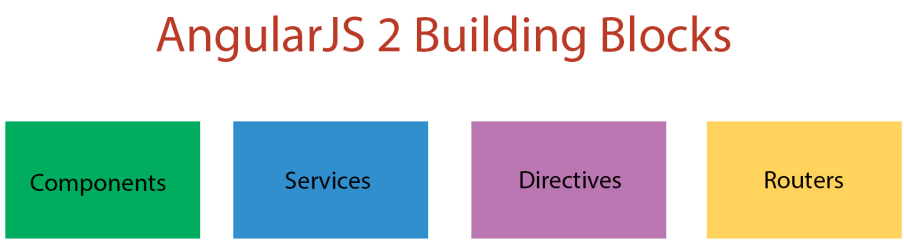


Figure D.0.4: AngularJS 2 building blocks.

and routers. Our application components encapsulate the data, template, and behavior of the view. The components allow us to reuse our code, and scale up our application easily. Services are delegated any logic not related to our view , e.g. sending requests to our back-end API to save or retrieve data. We use directives to modify DOM (Document Object Model) elements, and extend their behavior. Finally, routers are responsible for navigation to our different user interfaces or the transitions in a specific interface.

Packing modules together

We use webpack as the module bundler, to pack our modules together. As can be seen in Figure D.0.5 webpack takes modules with dependencies and generates static assets representing those modules. Webpack module loaders are responsible for compiling ES6 code to ES5 code, and can transform files from a different language like CSS or HTML, or inline images as data URLs . The downside of using

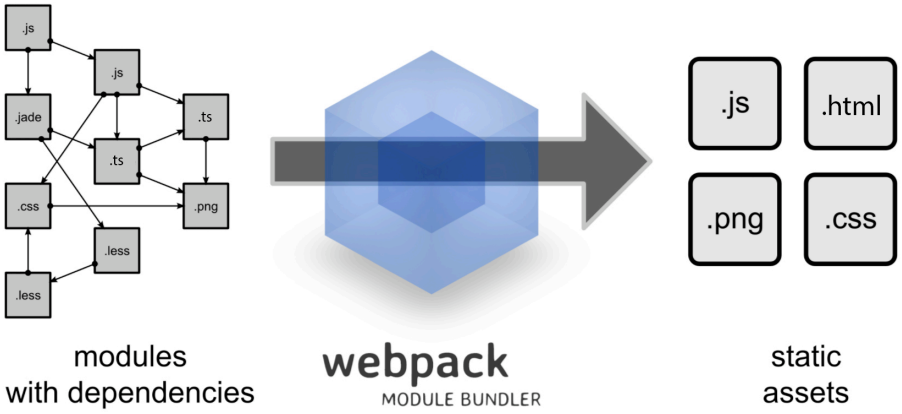


Figure D.0.5: Webpack.

webpack is a build step in the project, as the Es6 needs to be compiled into ES5 code. However, this step is fast, and worth the effort.

UNIT TESTING

Unit testing is a method of testing individual units of source code. Therefore functionality of different application units is tested separately, and when developers change code, we can assure the tested functionality of code is not broken. Owing to separation of the application architecture into components in AngularJS 2, our code is easily unit testable. We use Jasmine and Karma for unit testing our application. Jasmine is a behavior driven development JavaScript framework for testing Angular applications. Karma is a JavaScript command line tool that can be used to spawn a web server which loads our application's source code and executes our tests.

STYLING

By using CSS, and Bootstrap styling, we optimize use of screen real state based on device type. For example the navigation sidebar will disappear when accessing the user interface on a mobile phone.

E

Papers

1. **Farzad Nejatimoharrami**, Andres Faina, Kasper Stoy, "New Capabilities of EvoBot: A Modular, Open-Source Liquid Handling Robot", SLAS Technology Journal

2. **Farzad Nejatimoharrami**, Andrés Faiña, Jitka Čejková, Martin Hanczyc and Kasper Stoy, "Robotic Automation to Augment Quality of Artificial Chemical Life Experiments", In proceedings of the Artificial Life Conference (ALIFE), pp. 634-635, 2016.

3. **Nejatimoharrami F.**; Faina, A.; Jovanovic, A.; Olivier, S.;Chignell, M.; Stoy, K., "UI Design for an Engineering Process: Programming Experiments on a Liquid Handling Robot", Accepted in IEEE International Conference on Robotic Computing, April 2017

4. **Farzad Nejatimoharrami**, Kasper Stoy, Andres Faina " An Open-Source, Low-Cost Robot for Performing Reactive Liquid Handling Experiments", the fifth Annual International SLAS (Society for Laboratory Automation and Screening) Conference (SLAS 2016)

5. **Farzad Nejatimoharrami**, Andres Faina, Kasper Stoy, "A Low Cost Standalone Open-Source Robotic Platform for Performing Feedback based Liquid Handling Experiments", Accepted in the Sixth Annual International SLAS

(Society for Laboratory Automation and Screening) Conference (SLAS 2017)

6. Andrés Faiña, **Farzad Nejatimoharrami**, Kasper Stoy, Pavlina Theodosiou, Benjamin Taylor and Ioannis Ieropoulos, "EvoBot: An Open-Source, Modular Liquid Handling Robot for Nurturing Microbial Fuel Cells", In proceedings of the Artificial Life Conference (ALIFE), pp. 626-633, 2016.

7. Andrés Faiña, **Farzad Nejatimoharrami**, Kasper Stoy, "Towards EvoBot: A liquid-handling robot able to automatize and optimize experiments based on real-time feedback", Workshop on Exploiting Synergies Between Biology and Artificial Life Technologies: Tools, Possibilities, and Examples, Fourteenth International Conference on the Synthesis and Simulation of Living Systems, New York, NY

8. Theodosiou, P., Faina, A., **Nejatimoharrami, F.**, Stoy, K., Greenman, J., Melhuish, C., & Ieropoulos, I. (2017, July). EvoBot: Towards a robot-chemostat for culturing and maintaining Microbial Fuel Cells (MFCs). In Conference on Biomimetic and Biohybrid Systems (pp. 453-464). Springer, Cham.

9. Janska P., Zadrazil A., **Nejatimoharrami F.**, Faiña A., Stoy K., Cejkova J. (2016). Collective behaviour in droplet systems, 43rd International Conference of SSCHE, Tatranske Matliare, Slovakia 23.-27.5.2016

10. Andrés Faiña, **Farzad Nejatimoharrami**, and Kasper Stoy, "EvoBot: An Open-Source, Modular Liquid Handling Robot", under review in IEEE Transactions on Mechatronics.

11. **Farzad Nejatimoharrami**, Andres Faina, Kasper Stoy, "A Low-Cost Open-Source Cloud-based Liquid Handling Robotic Platform for Performing Remote Real-Time Collaborative Experiments", under review in the seventh Annual International SLAS (Society for Laboratory Automation and Screening) Conference (SLAS 2018)