Language & Tools for Context-Aware Biology

by

Charles Fracchia

Submitted to the Program in Media Arts and Sciences
School of Architecture and Planning
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2014

Signature redacted

Author ....
Program in Media Arts and Sciences
School of Architecture and Planning
Aug 22, 2014

Signature redacted

Certified by..................
Joseph Jacobson
Associate Professor of Media Arts and Sciences
Thesis Supervisor

Signature redacted

Accepted by................
Pattie Maes
Interim Academic Head
Program in Media Arts and Sciences

Language & Tools for Context-Aware Biology

by

Charles Fracchia

Submitted to the Program in Media Arts and Sciences
School of Architecture and Planning
on Aug 22, 2014, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Current biological research workflows make use of disparate, poorly integrated systems that cause large mental burden on the scientist leading to mistakes on often long, complex and costly experimental procedures. The lack of open tools to assist in the collection of distributed experimental conditions and data is largely responsible making protocols difficult to debug and laboratory practice hard to learn. In this thesis, we describe an open Protocol Descriptor Language (PDL) and system to enable a context-rich, quantitative approach to biological research. We detail the development of a closed-loop pipetting technology and a wireless, sample temperature sensor that integrate with our Protocol Description platform enabling novel, real-time experimental feedback to the researcher thereby reducing mistakes and increasing overall scientific reproducibility.

Thesis Supervisor: Joseph Jacobson
Title: Associate Professor of Media Arts and Sciences

Thesis Committee

Signature redacted

Professor Joseph Jacobson......

Thesis Supervisor
Associate Professor of Media, Arts and Sciences

Signature redacted

Professor George Church. ....

Member, Thesis Committee
Professor of Genetics, Harvard Medical School

Signature redacted

Professor Neil Gershenfeld.... ..

Member, Thesis Committee
Professor of Media Arts and Sciences
Director, Center for Bits and Atoms

Signature redacted

Dr Shuguang Zhang ....

Member, Thesis Committee
Principal Research Scientist, MIT Media Laboratory

Don't ask what the world needs. Ask what makes you come alive, because what the world needs is people who have come alive.

Howard Thurman

This document is dedicated to all the people who have enabled me and continue to help me find my own answer. For your support and kindness, my sincerest thanks.

In particular:

Dr. Shuguang Zhang
Prof. Joe Jacobson
Prof. George Church
Linda Peterson
Giovanni, Alice and Cecile Fracchia
Laura Beth Mours
Adam Marblestone

Contents

1 Introduction **19**

1.1 Protocols: The Script Unit for Biology 19

1.2 Laboratory Environment 20

1.3 Data Collection in Biology 21

1.3.1 Electronic Lab Notebooks 22

1.3.2 Biological Research Equipment 22

1.4 Reproducibility in Biology 24

1.5 Design Paradigms in Biological Research 24

1.6 Teaching in Biological Laboratories 25

1.7 The Rise of Synthetic Biology 25

1.8 Biology as a Service (BaaS) 26

1.9 Context-Rich Approach 26

2 PDL: Protocol Descriptor Language for Biological Research **29**

2.1 A Context-Aware Protocol 29

2.2 System Architecture 30

2.2.1 Walk-through 32

2.2.2 Protocol Timeline Visualization 32

2.2.3 Data Aggregation Engine 33

2.2.4 Notes and Comments Visualization 34

2.2.5 Notifications 36

2.2.6 Timer 38

2.3 Simulation and Outcome Predictability 38

2.4 Robotic Automation 39

2.5 Reproducibility 40

3 A Closed-Loop, Context-Aware Pipetting System **41**

3.1 System Architecture 42

3.2 Tracking and Annotating Well Plates 43

3.2.1 Well Detection 43

3.2.2 Corner Well Detection 44

3.2.3 Finding Well Rows 45

3.3 Tracking Pipette Position 47

3.4 Volume Sensing 50

3.5 User Interface Feedback 51

3.6 Pipette Control 51

3.6.1 Reverse Engineering the EDP3 52

3.6.2 Custom Interface Board for the EDP3 53

3.6.3 Open Communication Library in Python: pyEDP3 55

4 Continuous Monitoring of Sample Temperature **57**

4.1 Hardware Design 57

4.1.1 Electronics Architecture 58

4.1.2 Temperature Sensor 58

4.1.3 Temperature Calibration 60

4.1.4 Wireless Communication 61

5 Deployment **63**

5.1 Deploy or Die 63

5.2 Rapid Prototyping 63

5.3 Production 64

5.4 Dissemination and Market Reach 65

6 Conclusions & Future Directions **67**

6.1 Simulation Tools 67

6.2 Analytics for Data-Driven Biological Research 68

6.3 Contextual Sensors 68

A Protocol Descriptor Language Specification v4 **69**

B Closed-Loop Pipetting Computer Vision Code **71**

C Rainin EDP3 Communication Specifications **85**

C.1 Packet Structure 85

C.1.1 Example Command 85

C.1.2 Example Responses 85

C.2 Commands 85

C.3 Checksum Calculation 86

C.3.1 Example 87

D pyEDP3 Serial Port Example **89**

D.1 Command Names 89

E I^{2}C Firmware for Communication with STLM75 Temperature Sensor **91**

List of Figures

1-1 Typical biological laboratory bench 21

1-2 Example of commonly used biological equipment. Note the space used by the computer needed to run the Nanodrop DNA spectrophotometer. 23

2-1 PDL system architecture. Solid lines represent elements of the system that were implemented as part of this work. Dotted lines refer to other elements compatible with the PDL system but are not implemented as part of this work. 30

2-2 Walk-through of the PDL system. The first step is to use the writing module to create the protocol (1) and export it to a PDL file. The protocol can then be uploaded to the system for future saving and execution (2). Once uploaded, the interface will take the user to the execution screen for the protocol (3). 31

2-3 A competent cells protocol visualized using the Timeline feature of the Protocol Descriptor Language system. Original competent cells protocol courtesy of Dr. Tom Knight. 32

2-4 Path across the system architecture of a typical sensor data packet . . 33

2-5 Example of a JSON packet sent by the temperature sensor designed in this work 34

2-6 The protocol for λred recombination annotated by its author with information that would be shared after multiple protocol failures. Protocol courtesy of Dr. Marc Lajoie. 35

2-7 Left: Reference implementation snippet for qualitative comments in PDL. Right: A qualitative comment is displayed as a pop-up at the start of a step and when the user hovers over the target region. 36

2-8 Left: Reference implementation snippet for quantitative comments in PDL. Right: A quantitative comment is shown as a heat map of the three ranges: ideal, passable and unacceptable. 37

2-9 Sample notifications being displayed on the Web interface. A simple color scheme from green to red is employed to denote severity/importance of the notification. 37

2-10 Left: A common lab timer. It consists of an LCD screen for displaying time, a buzzer and a few buttons to navigate through a couple of timers and start/stop them at will. Right: Sample notification integration where the user is informed of the timer's completion via SMS or through a heads-up display on Google Glass. 39

3-1 System architecture for the closed-loop pipetting system. The overhead camera provides video to the embedded computer running the computer vision code in real-time. The code detects the 96-wells of the plate, annotates them with their address and sends a message to the Web interface displayed on the tablet to provide real-time pipetting feedback to the user. An electronic pipette is used to *close the loop* in our system and prevent pipetting into the wrong well. 42

3-2 A number of commercially available plates. Note the diversity in color, number and position of outer notches, etc. 44

3-3 Circle detection being performed on a common 96-well translucent plate 45

3-4 Corner well detection using the quadrant method for reduced search time 46

3-5 Left: Fitting a line to find row wells. Right: Missing wells compensation algorithm working when fingers obstruct the view. 47

3-6 Final result of annotation process 48

3-7 Color-filtering process applied to tip tracking. Left: The raw image with a tip colored blue. The tip was colored more than necessary for effect. Right: The result of the color filtering mask showing only the tip. 49

3-8 Algorithm for volume detection using overhead video from the USB camera. The plate cross-over volume is determined as the minimum volume necessary for forming continuous layer surface inside of the well. This volume varies depending on the specific plate used and the diameter of the bottom of the well 50

3-9 Left: Well with no water inside. Center: Well with 40 µL which is the cross over volume for this well, destroying the pattern. Right: demagnifacation effect caused by the liquid in the well. In this case: 100 µL 51

3-10 The MIT logo is drawn on the well plate using the closed-loop pipetting system developed in this work 52

3-11 The annotated view of the EDP3 electronics following reverse engineering 53

3-12 Left: We connected a logic analyzer to the EDP3 battery port to capture and reverse engineer the pipette's communication port. Right: Captured response packets on the response line after sending data to the EDP3 using the remote communication program. Program courtesy of Steve Konrad. 54

3-13 Left: Board layout for the wireless adaptor using the exposed communication port on the Rainin EDP3 pipette. Right: The wireless board installed in the battery compartment of the EDP3. The board was designed to maximize backwards compatibility and require as few modificiations as possible. 55

4-1 Left: Sample wireless temperature sensor electronic annotated board design. Right: Assembled prototype within case and with sensor inside the Eppendorf tube 58

4-2 Temperature-sensing speed tests of the DS18B20. This shows that the DS18B20 is slow (approx. 20m) to adapt to large temperature changes, in particular when moving toward colder environments. This is in part due to the heat generated by the system by being continuously on. However, we found that different temperature sensors were able to adapt to the same differentials in a matter of seconds. 59
4-3 Temperature-adaptation experiment using the STLM75 temperature sensor with different holders in ice (left) and ice-water mix (right). . . 60
4-4 STLM75 temperature-sensor adaptation time validation experiments in heating conditions inside an incubator 61

List of Tables

5.1 Bill of materials (BOM) for the wireless temperature sensor device described in this thesis 65

Chapter 1
Introduction

Biological research laboratories are spaces regulated by the BioSafety Level (BSL) guidelines[63] where scientists carry out their experiments. These experiments are described in documents called protocols, which outline the procedures step-by-step. Oftentimes, the procedures require the use of instruments (e.g. centrifuge) to apply the sought transformations to the samples. With the exception of certain specialized industrial processes, the laboratory environment today is still dominated by disparate, poorly integrated workflows, making it difficult to collect a comprehensive picture of the experiment. With that in mind, this thesis develops a suite of open, integrated technologies that enable a context-rich approach to biological research. This includes the creation of a Protocol Descriptor Language (PDL), a closed-loop pipetting system and sample temperature sensors.

In this document, we first outline the biological research environment and its current limitations before detailing the architecture of the Protocol Descriptor Language and execution engine, the closed-loop pipetting system and sample-sized networked temperature systems. We will also discuss the implications this work has on biology massive online open courses (MOOCs) and quantitative-based learning in biological experiments.

1.1 Protocols: The Script Unit for Biology

In biology, protocols are documents that describe the operations required to carry out a specific experiment. These documents can vary greatly in their depth and formatting as they are often written and consumed by the same individual. As shown in 2-6, this often leads to protocols that lack a lot of information considered implicit or common knowledge by the original author.

Protocols are frequently laid out as bulleted or numbered lists of transformations to be applied to the samples. In many cases, the time frame within which these transformations occur is important as many of the steps involve chemical reactions whose efficiencies are often time-dependent. Because of the widespread time-sensitivity of these operations, they are most often sequential and parallel operations are only introduced when the scientist is familiar with and proficient in the protocol's steps and details. This is due to the fact that many steps involve the combination of various reagents and materials often held at specific temperatures (refrigerator, freezer, etc.) or make use of common resources shared among a lab. Furthermore, the handling of multiple, parallel protocols is quickly impeded by the mental burden it places on the scientist and thus is rarely performed in labs that rely on human operation.

All these factors have contributed to the current culture in biological research labs, where debugging of biological protocols is so arduous that they are often repeated first and not re-assessed until the failure is more consistent. Beyond the obvious time and cost associated with this practice, it is made clear that the current environments do not provide the right tools and enough contextual data to conduct reliable reproducible research (see section below).

1.2 Laboratory Environment

While the exact layout of these laboratories can vary, the common unit of the laboratory is the bench, as pictured in 1-1. Benches are working surfaces most often made of chemically inert resins to avoid reactivity with materials in an eventual spill. While lab benches are common to almost any laboratory space, the way they are used or even occupied varies widely based on the research context. Two such major contexts exist: the academic and industrial contexts. In the former, manual labor is more prevalent, while in the latter, robotics and other high-throughput platforms are more commonly used. This is in large part due to the emphasis placed on the process and scalability in industrial settings. As a result, benches in academic settings are generally designed around the scientist as a central element, whereas robotics and automation platforms are more at the center of the attention in more industrial settings. However, while their use and design of labs may differ, both academic and industrial academic research environments rely on equipment to perform, simplify, and abstract certain tasks carried out in their research.

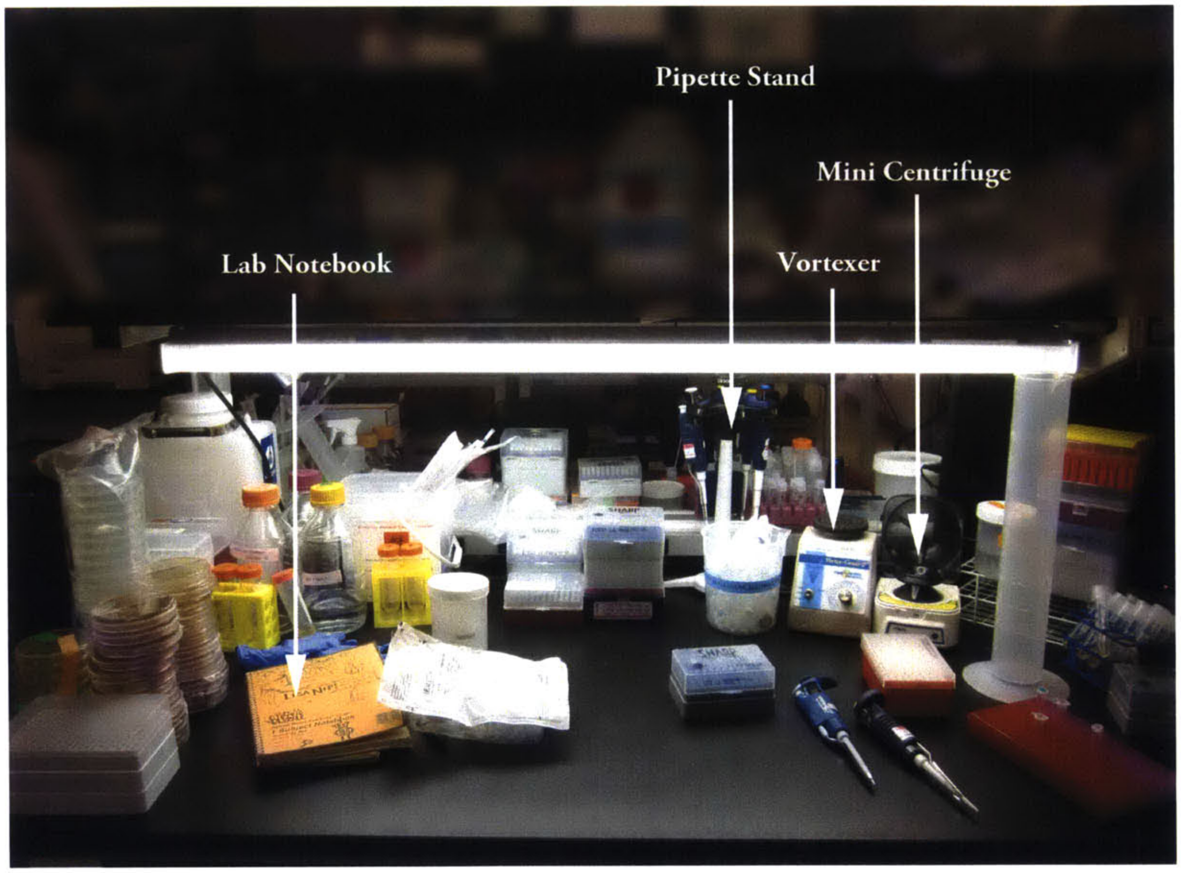


Figure 1-1: Typical biological laboratory bench

1.3 Data Collection in Biology

In laboratory settings, data collection can mean many different things and involve varied processes. In many cases, data collection relies on the machines used to export data in a common format, such as spreadsheets. However, many processes in biological research generate complex visual outputs that are solely analyzed and interpreted by the scientist. In both cases, results are compiled in a lab notebook, and all variations to the experimental procedure or setup should be recorded within it.

1.3.1 Electronic Lab Notebooks

Electronic lab notebooks are electronic counterparts of the traditional paper-based lab notebook. They often rely on content management platforms like MediaWiki[33][45] or note-taking applications like Evernote[20]. The overwhelming majority of current electronic lab notebooks use either a keyboard or voice for input (using text-to-speech). Unfortunately, the highly technical jargon routinely used in these experiments makes the latter a non viable option. While dedicated keyboards do not pose the same problem of input accuracy, they present their own challenges, including cleanliness, typing ergonomy and input speed. The first is a problem that relates to cross-experiment contamination. In fact, in laboratories, keyboards are often considered as potential sources of contamination due to the difficulty associated with thoroughly cleaning them[58][82]. While this particular problem could be solved by using a silicone, water-resistant keyboard, the typing ergonomy of such a keyboard is significantly impacted. Finally, when comparing traditional notebooks with their electronic counterparts, the celerity associated with taking a note on paper often trumps navigating an electronic interface and typing. This is especially important since many experiments are time sensitive and thus the rigidity associated with an electronic input method can lead to significant annoyance on the part of the user.

All this means that while the use of electronic lab notebooks has increased in recent years[78], they have yet to gain widespread adoption. We postulate that this slow progression is due to two major factors: the lack of an ergonomic input interface and the limited added value that electronic lab notebooks provide.

1.3.2 Biological Research Equipment

Biological equipment stretches from the simplest devices, like vortexers and centrifuges to complex systems like liquid handling robots and fluorescent automated cell sorters (FACS) (see Figure 1-2). Nearly all of the equipment currently offered in the biological research field are proprietary systems that rarely possess computer communication interfaces. When they do, these are rarely detailed for the end user and are often meant to be restricted for the manufacturer's internal use. There are two main reasons that explain this phenomenon: companies treat software as a separate revenue model, often charging for software with very basic functionality and second, biologists rarely possess the computational skills required to communicate with the devices.

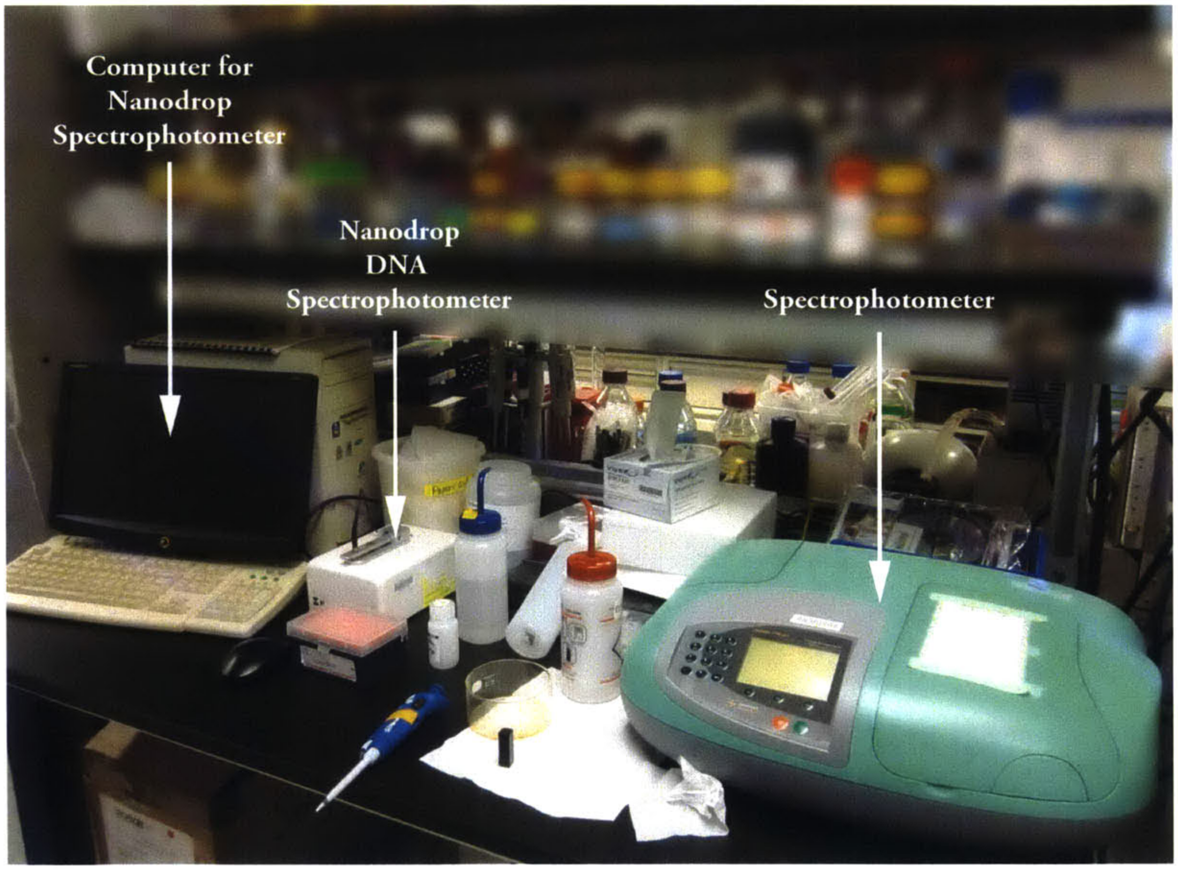

Computer for Nanodrop Spectrophotometer
Nanodrop DNA Spectrophotometer
Spectrophotometer

Figure 1-2: Example of commonly used biological equipment. Note the space used by the computer needed to run the Nanodrop DNA spectrophotometer.

All of this makes the modification of these machines and their cross-vendor integration nearly impossible and thus context-gathering from machines similarly unattainable. Yet biological research equipment is involved in many steps of a biological protocol and information about their particular parameters would be highly enabling for troubleshooting workflows and experiments.

1.4 Reproducibility in Biology

The lack of reproducibility in biological sciences has been an issue highlighted numerous times in the past[57][60][61]. In two separate studies, it was found that as little as 11%[61] and 25%[67] of the results were reproducible. The importance of protocols in experimental biology make them a key target for improving reproducibility. While some journals have created infrastructures for sharing and distributing protocols openly[38], the formats employed do not enable the centralized collection and comparison of experimental results. In fact, these exchanges mainly serve the purpose of disseminating protocol procedures between scientists, and do not currently possess the ability to quantifiably compare outcomes. It is important to note that this inability is not due to the journals' infrastructure but rather the lack of available, affordable technologies to collect the necessary data inside the laboratory. The technologies we present in this work are geared toward filling that void and enabling a quantitative approach to scientific reproducibility.

1.5 Design Paradigms in Biological Research

Designed biological experiments is a varied process, reflecting the diversity of tasks that are accomplished in biological protocols. Modern biology experimental design can go from DNA sequence design[42] to protein design[79] and population dynamics simulation. For this reason, a unified tool for experiment design is currently intractable. However, some efforts exist in the synthetic biology field to create a unified system to automate the experimental design process[9]. Likely due to the steep learning curve and workflow adaptations they require, these tools have not yet seen widespread adoption within the field.

1.6 Teaching in Biological Laboratories

A challenge involved in teaching modern molecular biology is often the laboratory practice. Laboratory work requires not only a correctly accredited facility but also a plethora of specialized technical equipment. For that reason, the teaching of biological laboratory practice remains difficult to democratize and disseminate widely. With the recent advent of Massive Open Online Courses (MOOCs)[16][11][56][53], the challenge to provide tools that are low cost, require little infrastructure and assist the learning process has only intensified. Furthermore, the tools currently used in laboratories largely do not collect data that allows quantitative assessment of success in real time.

1.7 The Rise of Synthetic Biology

While the conceptualization of the field can be traced back to the early 20th century[72], the establishment of synthetic biology as a field describing the engineering of synthetic systems for technological use and understanding of natural biology did not spread widely until around the year 2000[62]. During that time, the field attracted a number of engineers which, led to a number of synthetic devices being built using genetic circuitry, including an oscillator[66] and a light-sensitive kinase photographic system[74]. Furthermore, the establishment by a small group of MIT researchers, of the first international Genetically Engineered Machines (iGEM) competition[28], further boosted the growth of the field by providing training grounds for many undergraduates. While this creates a large amount of diversity in the direction of synthetic biology research, we note that the field has done little work to develop hybrid bioelectronic communication mechanisms[68]. We view the development of synthetic biology as an opportunity to bridge some of the technologies described in this work in order to create cellular-level context for biological protocols.

1.8 Biology as a Service (BaaS)

The rise of synthetic biology has also been a catalyst for the development of a new model for the biotechnology industry whereby individual processes are outsourced to service companies. Taking inspiration from the evolution that occurred in the software industry[84], we refer to this new model as Biology as a Service (BaaS). Examples of BaaS include current DNA synthesis companies, like Gen9[21] or IDT[27], which provide fragments that can be as long as 3,000 bp in the case of the former. This allows research labs and companies to remove the need to have an in-house synthesizer and thus remove complexity and development time. Others, like Ginkgo BioWorks[24], are aiming to expand the BaaS paradigm by abstracting away the entire process of genetically modifying organisms. Analogous to the effect the SaaS model had on the software industry, we expect Biology as a Service to enable more abstraction, thus lowering the barrier to entry for doing biological engineering. However, if the recent history of the software industry provides any clues, the current proprietary approach still widespread in the biotech industry should be abandoned in favor of a more open set of tools. This is the key reason motivating the open release of the tools detailed in this work. In due time, we also aim to create the business infrastructure required to ensure vast distribution and use of this platform while maintaining its open-source foundations[8].

1.9 Context-Rich Approach

The technologies and infrastructure laid out in this document enable the collection and visualization of disparate data sources as they relate to the experimental protocol in a context-rich environment. The context of an experiment can be defined broadly and efforts in this thesis should be considered a first step towards gathering more variables in the future. Common experimental variables worth measuring include temperature, pH, optical density, luminosity, various gas concentrations etc. However, beyond direct physical variables, the condition and measurements from the equipment also constitute an important target for achieving context-richness. In fact, most modern equipment will produce data in electronic media, but only rarely in an open format.

Achieving a context rich environment in biological laboratories would enable more reproducible experimentation, easier debugging of experiments and increased understanding of protocol variability.

Chapter 2

PDL: Protocol Descriptor Language for Biological Research

In the complex, multivariate environment of molecular biological research, protocols are a necessary guide that allow scientists to describe and reproduce steps. However, the currently widely used methods fall short in their ability to enable multivariate data collection, cross-protocol correlation, mental-burden reduction and use as a teaching tool.

To solve these problems, we created the Protocol Descriptor Language (PDL), an open-source schema for describing biological research protocols. At its heart, PDL is a simple XML structure that allows biologists to write their protocols easily in a format that is both human and machine readable. Alongside the core specification, we also created a Web-based platform that allows biologists to run PDL protocols, collect data from disparate data sources including temperature sensors and smart pipettes as well as assist in the redaction of new protocols. The suite of tools presented in this document also present themselves as ideal learning aides for laboratory biology.

2.1 A Context-Aware Protocol

Current, commonly used methods for writing protocols traditionally make use of word processors and bulleted lists. Such technology is not conducive to the use of interactive protocols that incorporate contextual information to aid protocol execution and quality assessment in real time. For this purpose, we created a protocol running application as part of the Web-based infrastructure developed for this thesis. The protocol-runner application leverages lightweight, modern communication protocol[37] to gather real-time sensor data from devices detailed later in this document. The runner application uses a PDL protocol file as input to aggregate, compute and visualize information relevant to the procedures carried out. There are five main features that enable context awareness in the protocol and researcher: protocol timeline visualization, data aggregation engine, notes and comments visualization, notifications and stopwatch.

2.2 System Architecture

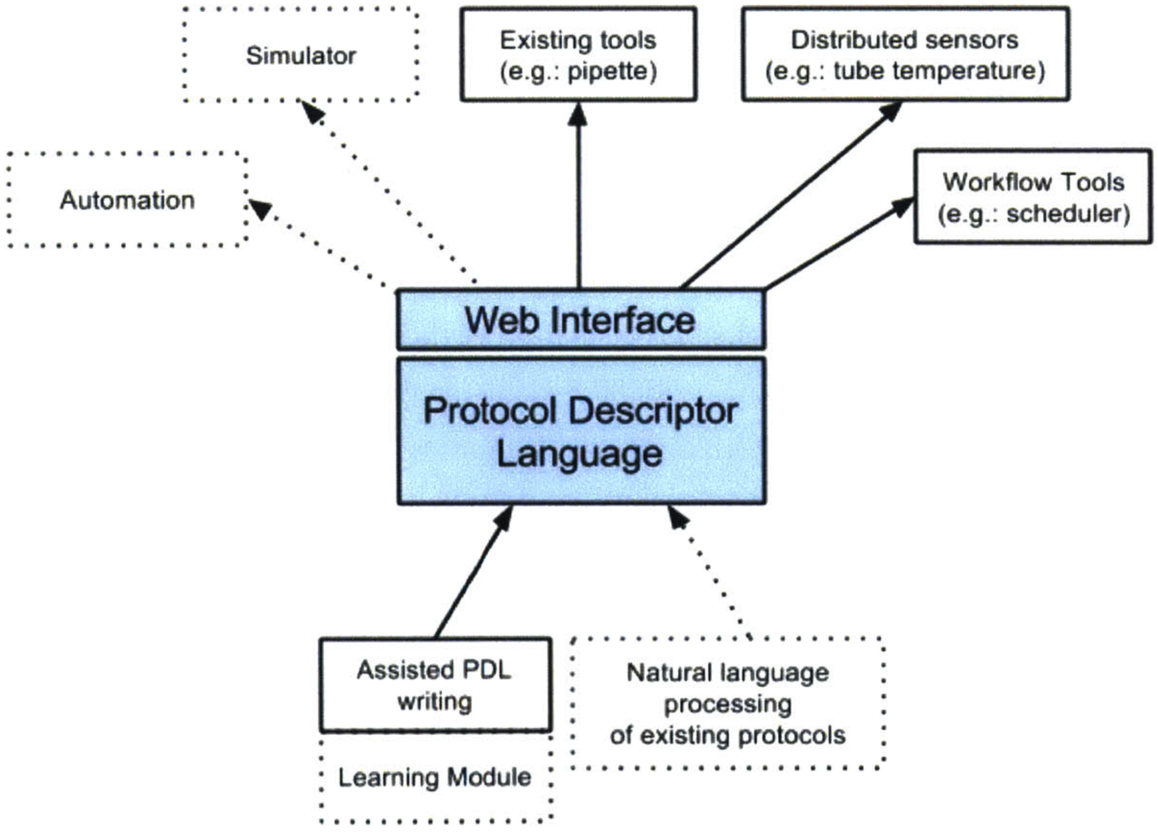
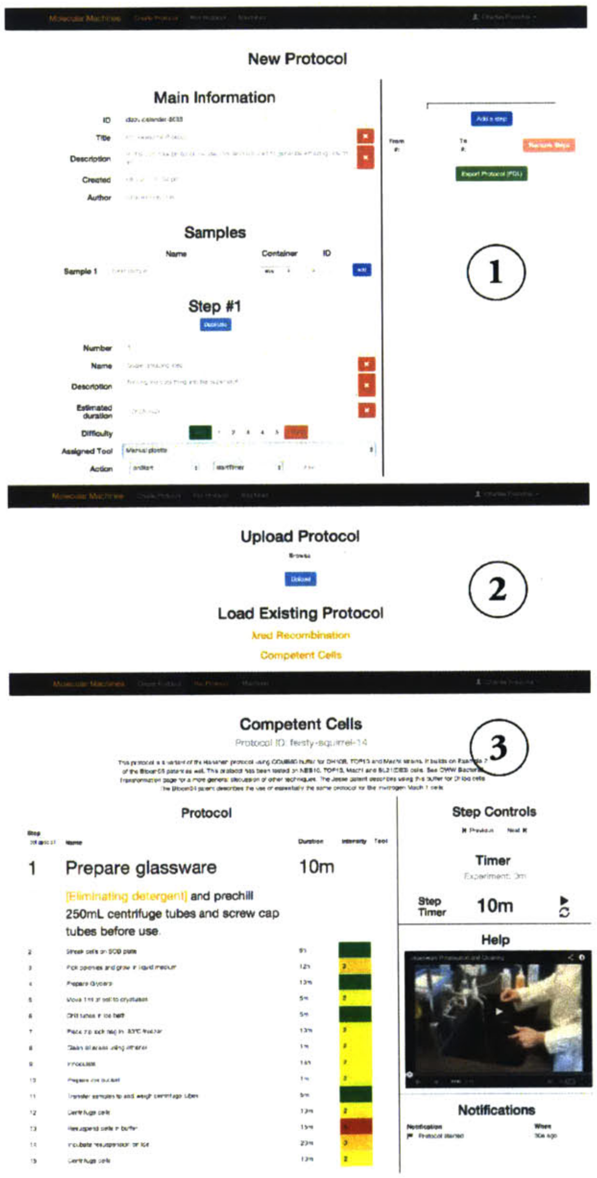

Simulator
Existing tools (e.g.: pipette)
Distributed sensors (e.g.: tube temperature)
Automation
Workflow Tools (e.g.: scheduler)
Web Interface
Protocol Descriptor Language
Assisted PDL writing
Learning Module
Natural language processing of existing protocols

Figure 2-1: PDL system architecture. Solid lines represent elements of the system that were implemented as part of this work. Dotted lines refer to other elements compatible with the PDL system but are not implemented as part of this work.

In this section, we describe the overall system architecture for the Protocol Descriptor Language and its associated Web-based implementation. The system diagram is visible in Figure 2-1. The core of the system rests on the PDL specification

(Appendix A), which is interpreted by the Web-based application. The application then implements all connections to devices, or other tools. This architecture permits an easy sharing of small files between scientists yet provides a central system with extensible features. Furthermore, the choice of the Web interface permits greater availability since all that is required of the user is to have a device equipped with a Web browser.



1 Prepare glassware 10m

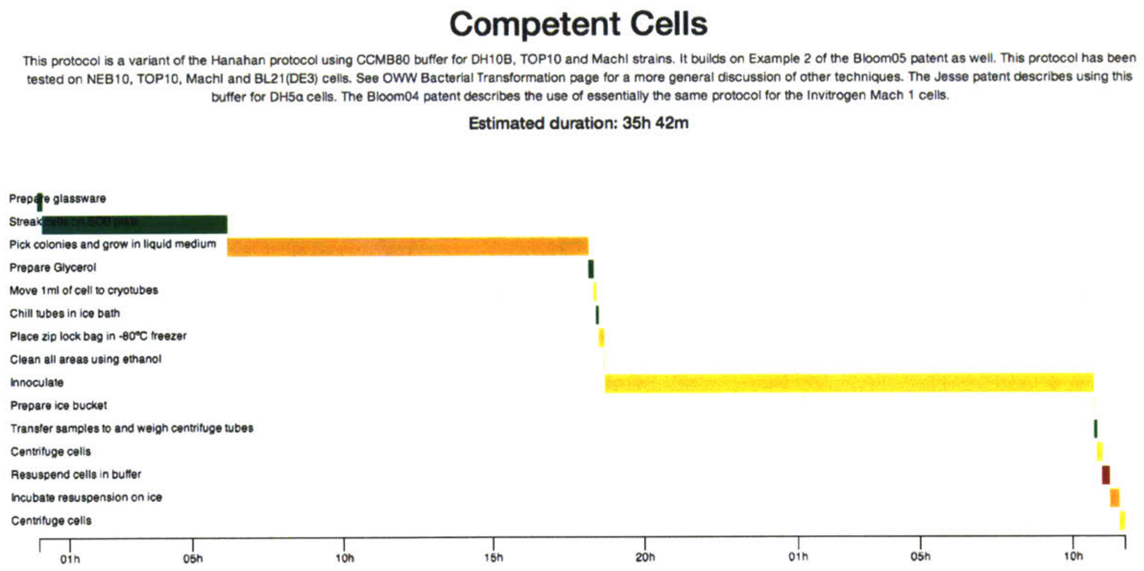
[Eliminating detergent] and prechill 250mL centrifuge tubes and screw cap tubes before use.

Figure 2-2: Walk-through of the PDL system. The first step is to use the writing module to create the protocol (1) and export it to a PDL file. The protocol can then be uploaded to the system for future saving and execution (2). Once uploaded, the interface will take the user to the execution screen for the protocol (3).

2.2.1 Walk-through

The system currently provides the ability to write, upload and execute PDL files (see Figure 2-2) and is available at `www.pdl.io`. Future plans include building an extensible framework on top of the core PDL functionalities described in this thesis to enable a modular approach to biological research. Drawing inspiration from the *app store* model [5] [25], we are making provisions in the framework to enable custom applications to access data gathered by the PDL system and interface with workflow-specific devices and processes.

2.2.2 Protocol Timeline Visualization


Competent Cells

This protocol is a variant of the Hanahan protocol using CCMB80 buffer for DH10B, TOP10 and Machl strains. It builds on Example 2 of the Bloom05 patent as well. This protocol has been tested on NEB10, TOP10, Machl and BL21(DE3) cells. See OWW Bacterial Transformation page for a more general discussion of other techniques. The Jesse patent describes using this buffer for DH5α cells. The Bloom04 patent describes the use of essentially the same protocol for the Invitrogen Mach 1 cells.

Estimated duration: 35h 42m

Prepare glassware
Streak cells on SOB plate
Pick colonies and grow in liquid medium
Prepare Glycerol
Move 1ml of cell to cryotubes
Chill tubes in ice bath
Place zip lock bag in -80°C freezer
Clean all areas using ethanol
Innoculate
Prepare ice bucket
Transfer samples to and weigh centrifuge tubes
Centrifuge cells
Resuspend cells in buffer
Incubate resuspension on ice
Centrifuge cells
01h
05h
10h
15h
20h
01h
05h
10h

Figure 2-3: A competent cells protocol visualized using the Timeline feature of the Protocol Descriptor Language system. Original competent cells protocol courtesy of Dr. Tom Knight.

As discussed previously, protocols can be complex series of steps that can last several days. This makes the tracking of every single step in time intractable for a an individual and is what constitutes a significant portion of the mental burden issue in biological labs. In order to address this issue, we used and extended a JavaScript timeline graphing library[12]. Figure 2-3 shows a sample protocol displayed as a timeline. Steps are color coded based on their expected complexity as described in the PDL format with the <intensity>tag. This enables the user to find at a glance which steps require more attention, thus minimizing the protocol's planning burden on the user. This feature is important in preventing potential timing conflicts between the user's other activities (like meetings or discussions) and the protocol he/she is running.

2.2.3 Data Aggregation Engine

The data aggregation engine runs as a separate Node.js[40] application from the main Web-server implementation. Its role is to monitor sensor messages relayed through the MQTT broker and insert the corresponding information in a database for storage. While Node's asynchronous input/output execution model[83] would allow for the data aggregation engine to be run within the main application, we opted for a separation of the two loops in order to insulate either of them against the other's halted execution.

In order to interface with the MQTT broker relaying the incoming sensor data, we make use of the MQTT package[39] and set the callback function to handle sensor data input. The data is transmitted as a JSON[29] payload by the sensor, with only one required key: the sensor value. Figure 2-4 shows the path taken by the data from the sensor all the way to the user interface.

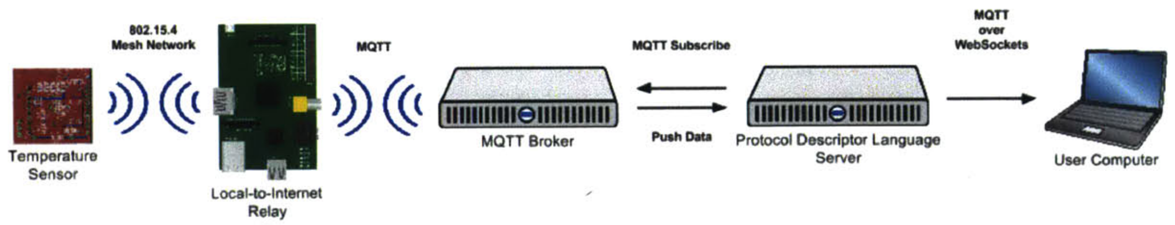


Figure 2-4: Path across the system architecture of a typical sensor data packet

The sensor can also send a time-stamp to specify the time of the sensor reading. The majority of sensors are expected to send data as time-stamp / value pairs, but in case of a missing time-stamp, the time at which the payload is received will be used as time-stamp by the database insertion application. Figure 2-5 shows a typical JSON payload for a sensor.

```
{
  "time" : 1407809212,
  "value" : 24
}
```

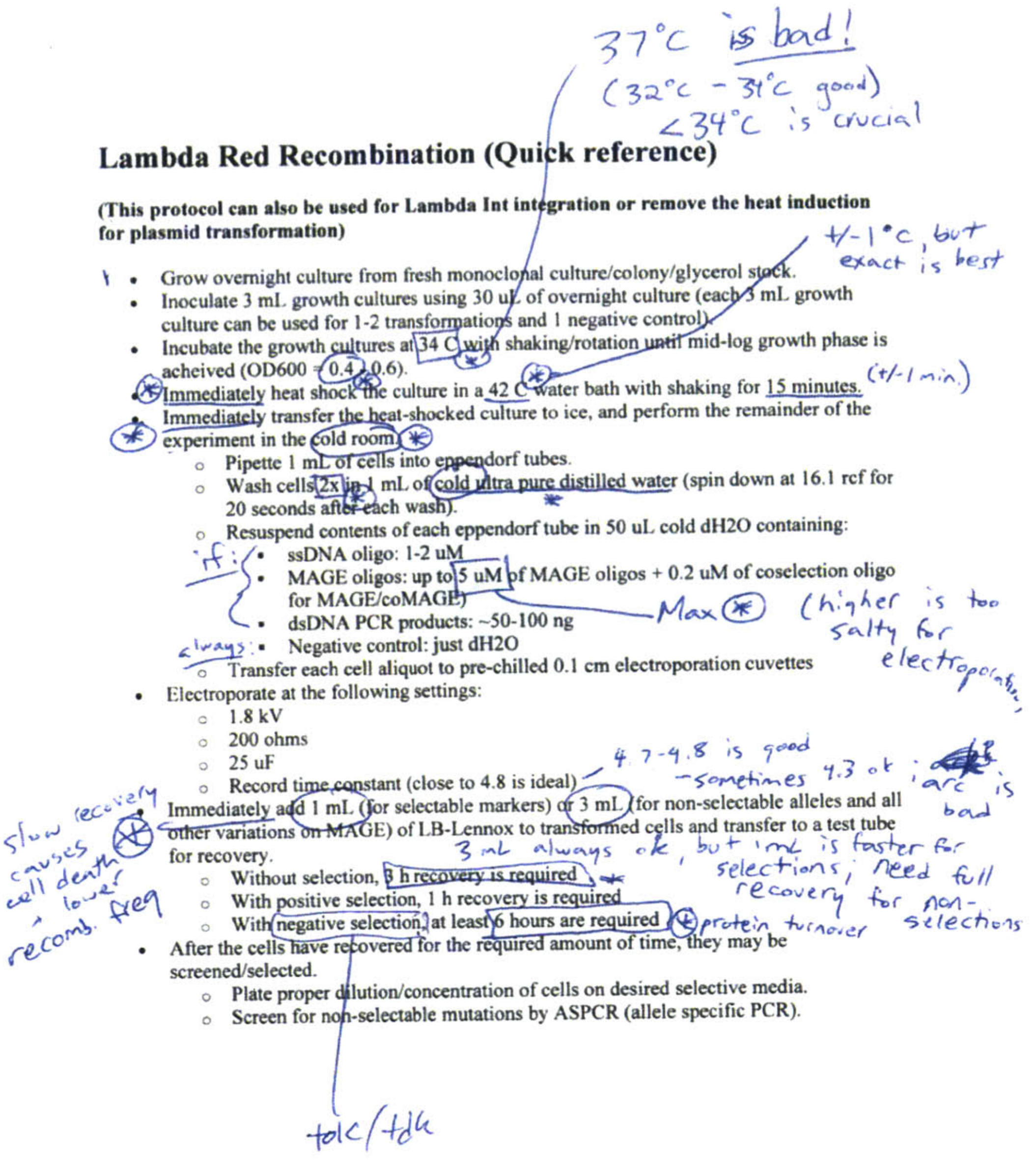
Figure 2-5: Example of a JSON packet sent by the temperature sensor designed in this work

The database that was chosen for this implementation is MongoDB[35]. The rationale behind choosing MongoDB is three fold: First, MongoDB is a document-oriented NoSQL database making it ideal for interaction with JSON payloads. Second, is the greater speed compared to relational databases (e.g., MySQL). Finally, MongoDB is more readily available for use with Node applications through the use of well-established packages[36]. In this model, decoupling the data storage and visualization also enables greater scalability of the overall system by allowing independent control over each element.

2.2.4 Notes and Comments Visualization

As shown in Figure 2-6, protocols can contain complex annotations, notes and comments that are crucial to the successful running of a protocol. These annotations can be either qualitative or quantitative. In the former case, the annotations will usually provide more information about a specific process or provide a rationale for the particular procedure. In the latter, a value judgement is placed on a quantitative range of a particular reagent or condition. For example, as can be seen at the top of Figure 2-6, the author reveals in an annotation that it is crucial that the temperature be under 34 °C and avoid 37 °C.

These notes and comments are part of the Protocol Descriptor Language specification detailed in this thesis. They are displayed as part of the Web interface that helps to run protocols. This is part of the main Node application that is rendered to the user within his/her Web browser. Bringing information to the attention of the



37°C is bad!
(32°C - 34°C good)
<34°C is crucial

Lambda Red Recombination (Quick reference)

(This protocol can also be used for Lambda Int integration or remove the heat induction for plasmid transformation)

+/- 1°C, but exact is best

- Grow overnight culture from fresh monoclonal culture/colony/glycerol stock.
- Inoculate 3 mL growth cultures using 30 uL of overnight culture (each 3 mL growth culture can be used for 1-2 transformations and 1 negative control).
- Incubate the growth cultures at 34 C with shaking/rotation until mid-log growth phase is acheived (OD600 = 0.4-0.6).
- Immediately heat shock the culture in a 42 C water bath with shaking for 15 minutes. (+/-1 min.)
- Immediately transfer the heat-shocked culture to ice, and perform the remainder of the experiment in the cold room.
  - Pipette 1 mL of cells into eppendorf tubes.
  - Wash cells 2x in 1 mL of cold ultra pure distilled water (spin down at 16.1 rcf for 20 seconds after each wash).
  - Resuspend contents of each eppendorf tube in 50 uL cold dH2O containing:
    - if: ssDNA oligo: 1-2 uM
    - MAGE oligos: up to 5 uM of MAGE oligos + 0.2 uM of coselection oligo for MAGE/coMAGE) — Max (higher is too salty for electroporation)
    - dsDNA PCR products: ~50-100 ng
    - always: Negative control: just dH2O
  - Transfer each cell aliquot to pre-chilled 0.1 cm electroporation cuvettes
- Electroporate at the following settings:
  - 1.8 kV
  - 200 ohms
  - 25 uF
  - Record time constant (close to 4.8 is ideal) — 4.7-4.8 is good - sometimes 4.3 ok; arc is bad
- Immediately add 1 mL (for selectable markers) or 3 mL (for non-selectable alleles and all other variations on MAGE) of LB-Lennox to transformed cells and transfer to a test tube for recovery.

  slow recovery causes cell death + lower recomb. freq

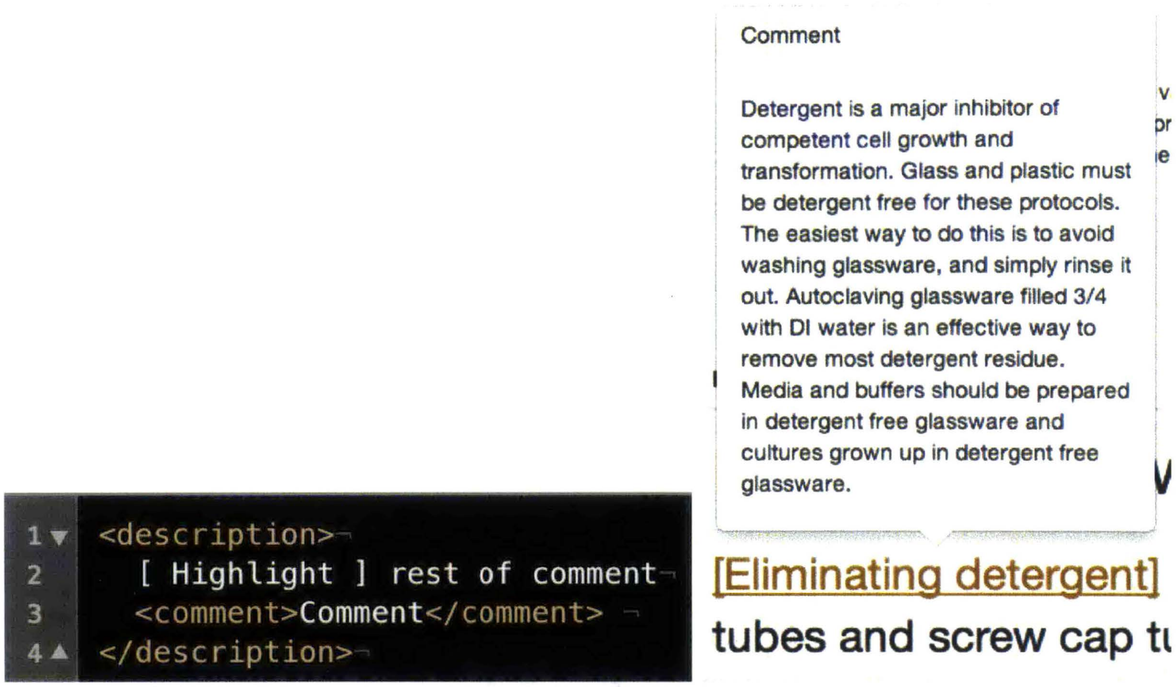
  3 mL always ok, but 1mL is faster for selections; need full recovery for non-selections
  - Without selection, 3 h recovery is required
  - With positive selection, 1 h recovery is required
  - With negative selection, at least 6 hours are required — protein turnover
- After the cells have recovered for the required amount of time, they may be screened/selected.
  - Plate proper dilution/concentration of cells on desired selective media.
  - Screen for non-selectable mutations by ASPCR (allele specific PCR).

tolC/tdk

Figure 2-6: The protocol for λred recombination annotated by its author with information that would be shared after multiple protocol failures. Protocol courtesy of Dr. Marc Lajoie.

user like notes and comments can be overwhelming to him/her and can compromise the quality of the overall user interface. In particular, given our goal to reduce mental burden, displaying too many notes and informational comments may be counterproductive. For this reason, we adopt a just-in-time approach[70] to displaying the notes and comments encoded in a protocol.

Qualitative comments are simply displayed as text pop-ups (see Figure 2-7) upon



```
<description>
  [ Highlight ] rest of comment
  <comment>Comment</comment>
</description>
```

Comment

Detergent is a major inhibitor of competent cell growth and transformation. Glass and plastic must be detergent free for these protocols. The easiest way to do this is to avoid washing glassware, and simply rinse it out. Autoclaving glassware filled 3/4 with DI water is an effective way to remove most detergent residue. Media and buffers should be prepared in detergent free glassware and cultures grown up in detergent free glassware.

[Eliminating detergent]

tubes and screw cap t

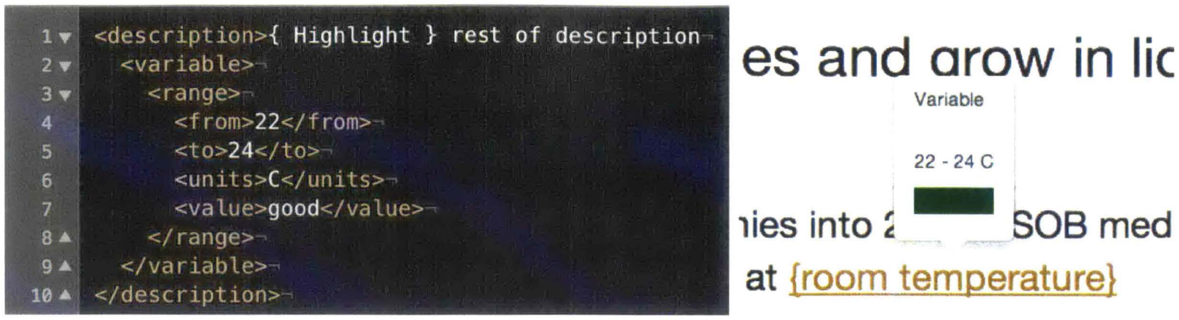
Figure 2-7: Left: Reference implementation snippet for qualitative comments in PDL. Right: A qualitative comment is displayed as a pop-up at the start of a step and when the user hovers over the target region.

hovering over the target region and at the start of the step. Qualitative comments are encoded in the Protocol Descriptor Language specification using [] to bound the comment region and the <comment>tag for its contents.

Quantitative comments on the other hand are displayed as a heat map of the numerical ranges from ideal (green) to passable (orange) to unacceptable (red). Quantitative comments are expressed in the PDL using {} to denote bounds and the <variable>tag is used to declare the ranges. The reference implementation and a visual example of quantitative comments is shown in Figure 2-8. The current implementation of the variable tag supports inclusive ranges; however, we expect future work to add support for a more complex range description.

2.2.5 Notifications

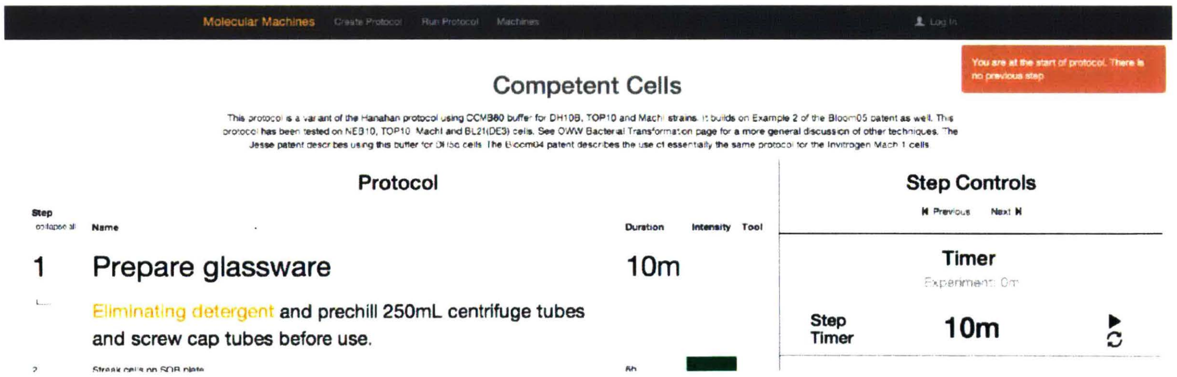
The notification engine has two main components: the client-side and server-side components. The former is coded in JavaScript, resides on the client-side of the



```
<description>{ Highlight } rest of description
  <variable>
    <range>
      <from>22</from>
      <to>24</to>
      <units>C</units>
      <value>good</value>
    </range>
  </variable>
</description>
```

Figure 2-8: Left: Reference implementation snippet for quantitative comments in PDL. Right: A quantitative comment is shown as a heat map of the three ranges: ideal, passable and unacceptable.

Web application (i.e., the user's Web browser), and is triggered by specific real-time operations like connection errors or specific protocol conditions being met. It is used to provide feedback to the user within the Web interface. On the other hand, the server-side component of the notification engine is used to communicate asymmetrically with the user. The current implementation includes communication channels like text messaging (SMS) and push notifications to wearable computing platforms like Google Glass. Examples of notifications being displayed are shown in Figure 2-9.



You are at the start of protocol. There is no previous step

Competent Cells

This protocol is a variant of the Hanahan protocol using CCMB80 buffer for DH10B, TOP10 and Mach1 strains. It builds on Example 2 of the Bloom05 patent as well. This protocol has been tested on NEB10, TOP10, Mach1 and BL21(DE3) cells. See OWW Bacterial Transformation page for a more general discussion of other techniques. The Jesse patent describes using this buffer for DH5α cells. The Bloom04 patent describes the use of essentially the same protocol for the Invitrogen Mach 1 cells.

Protocol

Step	Name	Duration	Intensity	Tool
1	Prepare glassware	10m		
	Eliminating detergent and prechill 250mL centrifuge tubes and screw cap tubes before use.			
2	Streak cells on SOB plate	6h		

Step Controls

Figure 2-9: Sample notifications being displayed on the Web interface. A simple color scheme from green to red is employed to denote severity/importance of the notification.

2.2.6 Timer

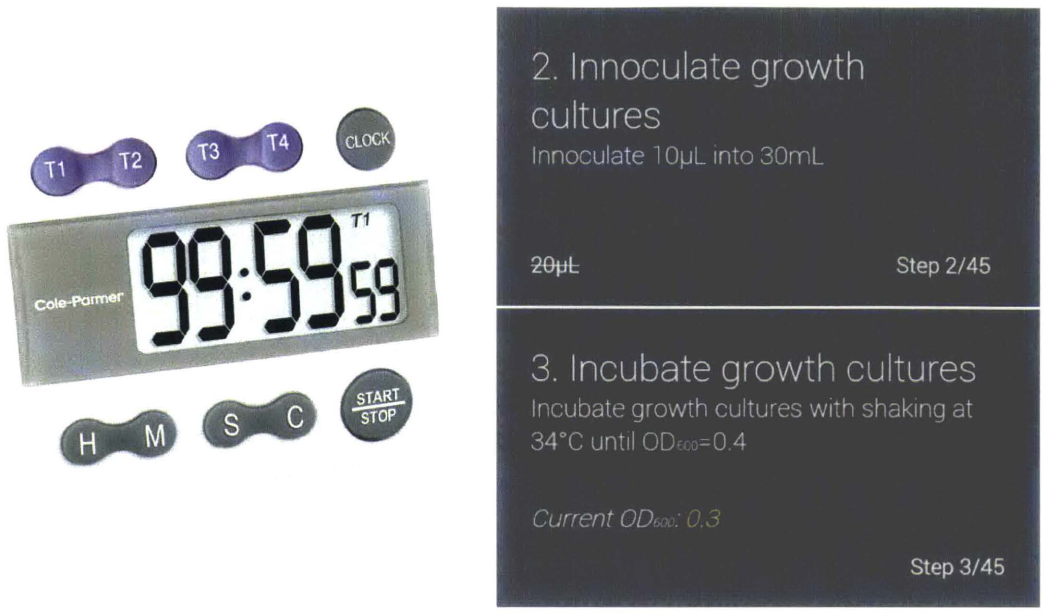
Timers are paramount to the correct execution of biological protocols. Timers are used to regulate the time of individuals steps of a protocol and physical timer devices are very commonly used in laboratories to help scientists. A common timer present in labs is picture in Figure 2-10. These timers usually possess only a beeper to provide feedback when reaching their target. While this is sometimes sufficient, it is common for the timer to go unanswered because of the user not being able to hear the beeping or not being in physical proximity to it.

The timer engine in the PDL system is run in software and rests upon the time values specified in the protocol PDL file. The timer is displayed in software on the Web application to provide feedback to the user. While not completed in time for this thesis, we are currently designing a physical networked timer that would communicate with the PDL infrastructure, thus allowing features like automatic timer setting as well as starting and stopping based on the rest of the experimental context. Currently, however, the timer engine allows integration with the notification setting, and in particular with the SMS and wearable computer communication channels.

2.3 Simulation and Outcome Predictability

The ability to simulate a protocol solely based on its machine-readable description is a desirable feature that would significantly enhance the speed of the design-build-test (DBT) cycle in biology. Enhancing the DBT cycle is recognized to be an important and enabling goal for biological engineering [13], and the system described in this thesis lays the ground work for the future development of a simulation package leveraging the information collected in the PDL system.

Two approaches can be used to create such a simulation package: bottom-up and top-down. The bottom-up approach implies simulating all physical parameters of the experiment from the ground up. While much work has been dedicated to simulating various parts of a biological system *ab initio*[64][59][76], these are limited



2. Innoculate growth cultures
Innoculate 10μL into 30mL

~~20μL~~ Step 2/45

3. Incubate growth cultures
Incubate growth cultures with shaking at 34°C until OD_{600}=0.4

Current OD_{600}: 0.3

Step 3/45

Figure 2-10: Left: A common lab timer. It consists of an LCD screen for displaying time, a buzzer and a few buttons to navigate through a couple of timers and start/stop them at will. Right: Sample notification integration where the user is informed of the timer's completion via SMS or through a heads-up display on Google Glass.

to specific subsystems and are yet to be compiled in models that enable holistic simulation of all aspects of an experiment. The top-down approach on the other hand implies first modeling the overall behavior of the system before attempting to model subsystems[65].

The data generated by the system described in this thesis is well suited to providing environmental variables used as parameters in models, whether they be bottom-up or top-down. Given that capability, we foresee useful future work to integrate real-time computation (where possible) of biological models with the data generated by the PDL system described in this thesis.

2.4 Robotic Automation

As discussed in the previous chapter, robotic automation platforms play an important role in industrial environments. However, they are currently limited by proprietary formats, rigid workflows and programming steps, making them intractable for low to medium throughput experiments and inadequate for rapid prototyping. It would therefore be desirable to simplify the actuation of these automation platforms and better integrate them into the rest of the biological laboratory workflows. The protocol descriptor language system is an ideal candidate for performing this function as it describes the operations of the entire protocol in a format that is designed to be both human and machine readable.

While the current version of the PDL system does not provide means to actuate robotic platforms, we expect to add functionality for actuating some of the most popular liquid handling platforms[4][54] in the near future.

2.5 Reproducibility

By virtue of keeping track of all protocol-related events, the PDL system; allows greater reproducibility and traceability of individual operations and conditions. Further work is needed to establish quantitative data showing the effect of the system, however, qualitative feedback relating to the interface has been encouraging. In particular, we aim to collect long tail data regarding sample temperature variability as well as step completion time variability. This will in turn allow us to perform analytics on protocol efficiency and variability and ultimately provide the user with reproducibility information. This information will be delivered in real time within the context of the protocol and will include information like the deviance of time or temperature from average values.

Chapter 3

A Closed-Loop, Context-Aware Pipetting System

Pipetting operations are at the core of nearly every molecular biology protocol. They provide a way to precisely move, mix or separate liquids, whether they be samples, reagents or other materials. However, a major issue of manual pipetting rests in the feedback it provides to the user. In fact, many of the liquids that are carried using pipettes in molecular biology experiments are transparent and do not present any distinguishing features. While a low number of operations or using robotic platforms may not present an issue, the reliable tracking of recent pipetting steps is difficult for a human operator. This leads to a commonly experienced issue in laboratories regarding content uncertainty. Recent efforts[86] demonstrate the use of a Web application running on a tablet to provide visual pipetting instructions for well plates. However, this work does not provide any feedback based on the actual movements of the user or pipette actuation. Other systems[23][17][3][23] rely on custom hardware to provide the pipetting instructions but still lack the ability to have real-time feedback, on the user's actions.

To solve this problem, we have developed a computer vision system able to track the position of the pipette tip relative to a pipetting plate. We also show how this information can be integrated with an electronic pipette through the Protocol Descriptor Language system to provide real-time pipetting feedback including volume quantity and sample content.

3.1 System Architecture

The different components of the closed-loop pipetting system are outlined in the system architecture in Figure 3-1.

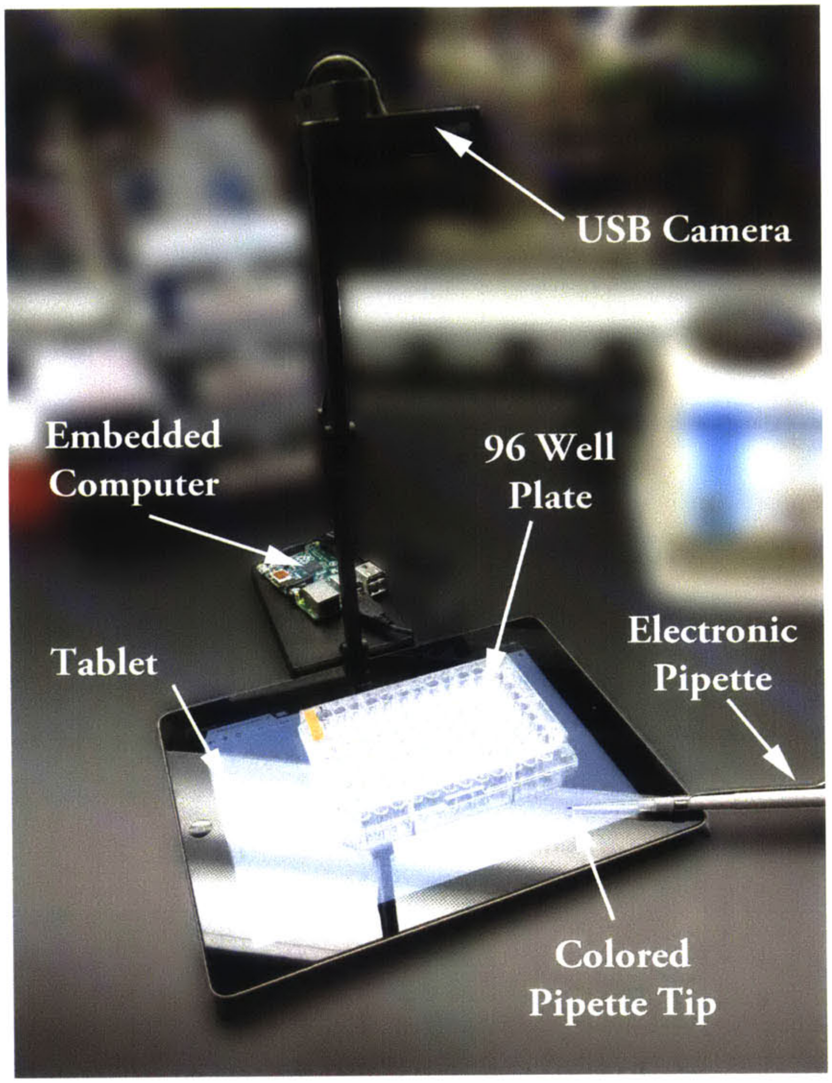


Figure 3-1: System architecture for the closed-loop pipetting system. The overhead camera provides video to the embedded computer running the computer vision code in real-time. The code detects the 96-wells of the plate, annotates them with their address and sends a message to the Web interface displayed on the tablet to provide real-time pipetting feedback to the user. An electronic pipette is used to *close the loop* in our system and prevent pipetting into the wrong well.

The system is composed of two main parts: the computer vision system and the electronic pipette. We have built the system to relax as many constraints as was possible and reasonable. Currently, the system only requires the use of a USB camera overhead and special tips whose sole modifications are their colored ends. The colored ends were achieved using a blue permanent marker. The color can be changed in the code and is therefore not restricted to our arbitrary choice of blue.

This project makes use of the popular OpenCV[43] library for Python to carry out the computer vision portion of this work. The task of tracking the user's pipetting patterns can be subdivided into two problems: first, obtaining positions and naming each of the wells in the plate being used and, second, obtaining the current position of the pipette tip. Once these two subproblems are solved, the current well into which the user is pipetting can be extracted by comparing the position of the tip to the position of the closest circle.

3.2 Tracking and Annotating Well Plates

The computer vision task of tracking and annotating well plates rests on the assumption that a defining, recognizable pattern can be extracted and tracked in a sufficiently agnostic way as to avoid restricting the user in their choice of plate. Upon examining a vast range of commercially available plates (see Figure 3-2), it became apparent that a number of features would not be suitable for tracking multiple types of plates. For example, while notches in the overall rectangular frame are often present at the corners of the plate, the position and number of each are not standardized across different vendors and plate types. The number of wells in a plate is determined by the number of rows and columns regimenting the grid pattern in which the wells are arranged. Plates are often referred to by the number of wells they have. Common formats are either 96-well plates: 8 rows (A through H) by 12 columns (1 through 12); and 384-well plates: 16 rows (A through P) by 24 columns (1 through 24).

3.2.1 Well Detection

With the exception of some cell growth plates, the vast majority of plates have circular wells arranged in the grid pattern previously described. We therefore decided to base

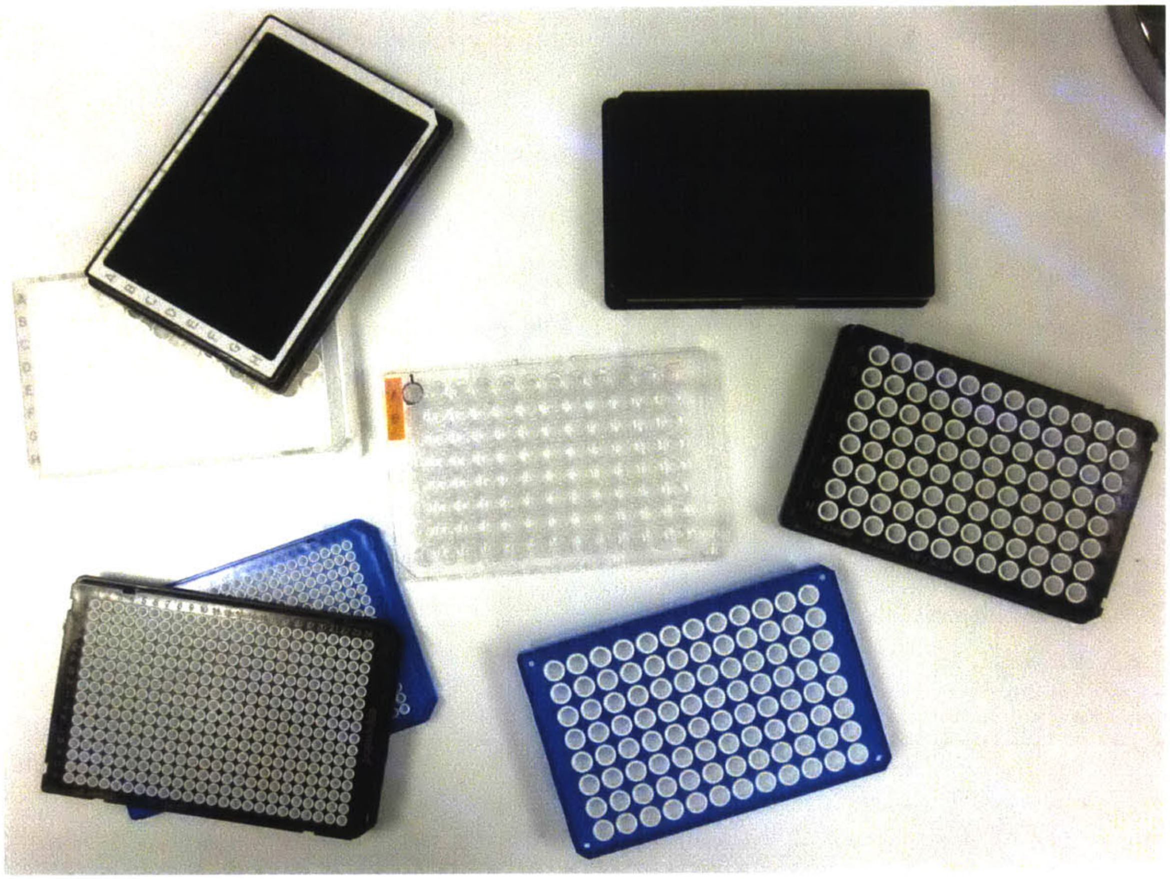


Figure 3-2: A number of commercially available plates. Note the diversity in color, number and position of outer notches, etc.

our detection on this common feature. Circle detection is performed in our system by using the OpenCV HoughCircles[44] transform. Internally, this transform first performs edge detection using the Canny algorithm[71] before using a gradient search method[85] to detect circles on the image. The algorithms using the transform each have input parameters that affect the overall result of the circle detection. The parameters were chosen to ensure maximum detection of the circles representing the wells in varied lighting and viewing angle conditions. A sample image of the circle detection being performed on a plate is shown in Figure 3-3

3.2.2 Corner Well Detection

Key to determining the bounds of a well by solely using the circular well as the feature is the ability to detect the edge wells, in particular the four corner wells. In order

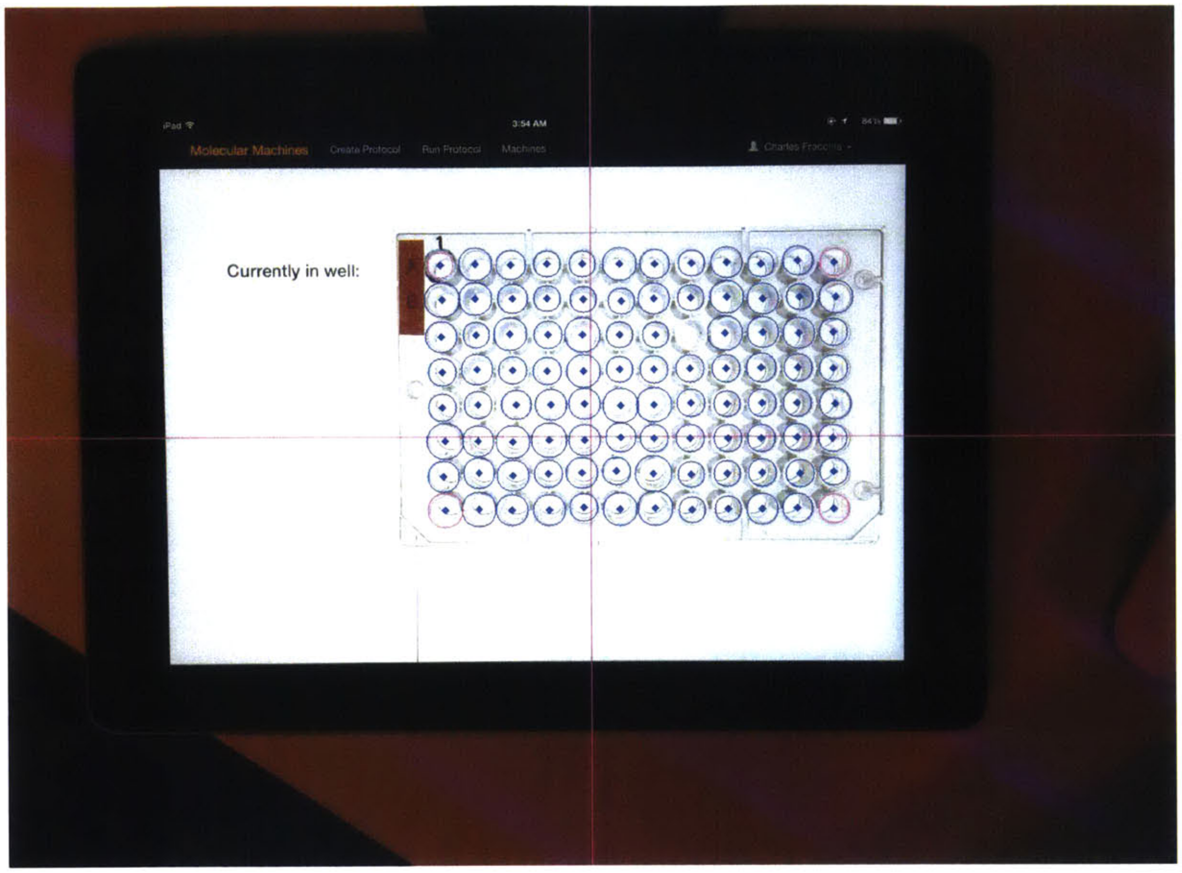


Figure 3-3: Circle detection being performed on a common 96-well translucent plate

to reduce search time through all detected circles from the previous step, the image is split into four equally sized quadrants. This is acceptable since the optimal use of this program involves placing the plate roughly centered on the image. The corner wells are then detected by computing the distance from each of the wells within the quadrant to their respective corners. Corner wells are determined as those having the smallest overall distance to their respective corners. An annotated frame showing corner well detection is visible in Figure 3-4.

3.2.3 Finding Well Rows

Once corner wells are detected, we now need to detect the rows of wells in the plate. To this end, we first ask the user to ensure that the plate is upright with well A1 in the top-left corner, allowing us to associate the top-left and top-right corner wells as A1 and A12 respectively. Using the center of each well obtained from the circle

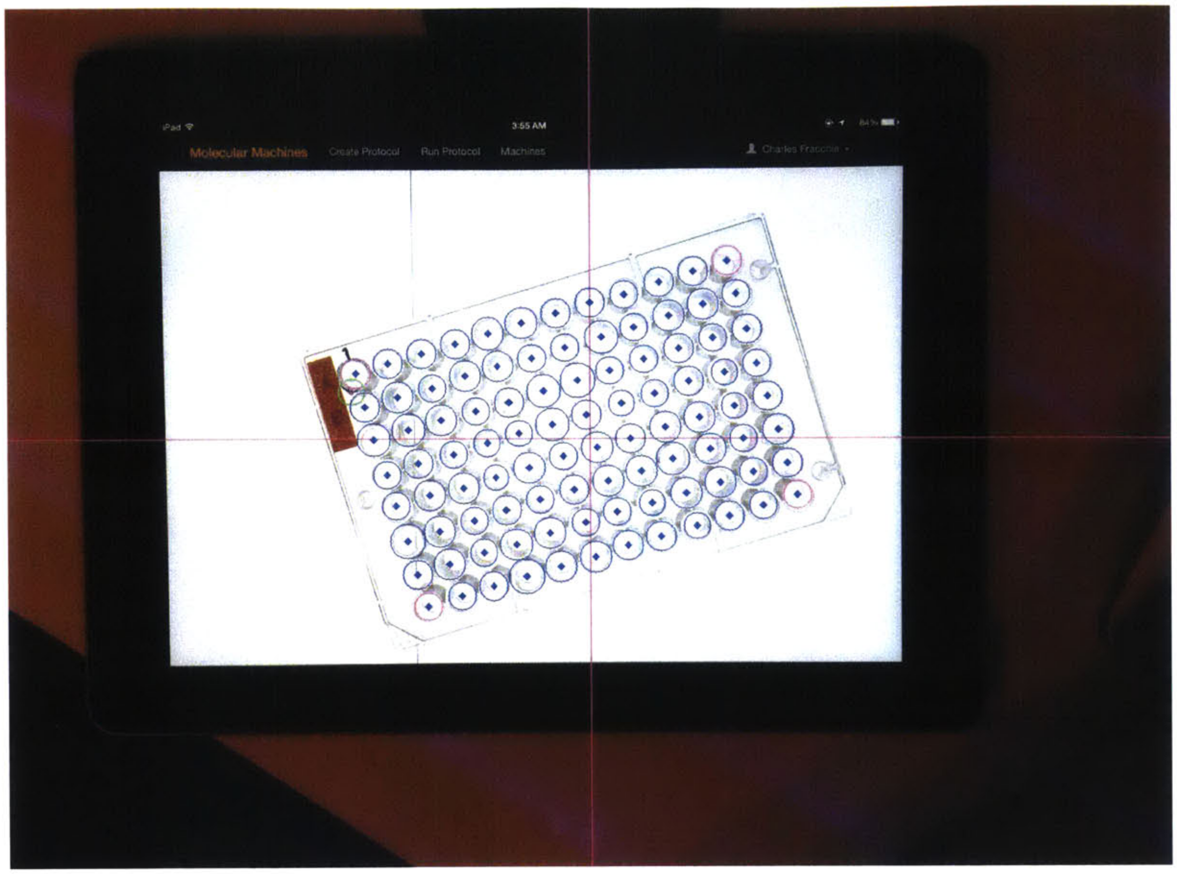


Figure 3-4: Corner well detection using the quadrant method for reduced search time

detection step described previously, we fit a line passing through both points. Using the knowledge that wells are arranged following a grid pattern, we thus expect all A row wells to be within some minimal distance from the fitted line. Once all row wells are detected, we iterate from both edge wells at columns 1 and 12 inward to associate the detected circle with the particular well.

It is important to note that some lighting conditions can affect the circle detection algorithm and prevent all circles from being detected, which can disrupt the proper annotation of the wells. In order to mitigate the effect of lighting artifacts, we check the distance between adjacent wells and if it is found to be larger than the average distance between detected circles on the overall plate, we instantiate the missing circle along the fitted line. This allows us to fill in wells that may be missing due to lighting or other aberrations. Figure 3-5 shows the result of fitting the line, finding the row wells and annotating each of them with their correct well address.

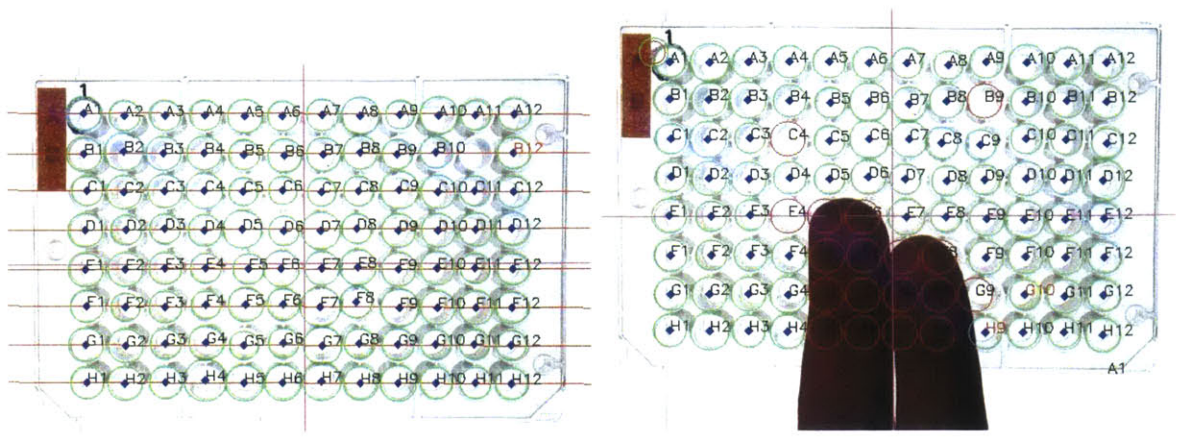


Figure 3-5: Left: Fitting a line to find row wells. Right: Missing wells compensation algorithm working when fingers obstruct the view.

The whole process is then repeated for each row. The following row edge wells are determined by calculating the wells with minimum distance from the previous edge wells but are not along the previously fitted row axis. The end result of this row-by-row annotation is shown in Figure 3-6. Tests conducted in different lighting conditions and plate orientations show that this annotation algorithm is robust and virtually devoid of wrongful well addressing.

3.3 Tracking Pipette Position

Tracking the pipette position relative to the detected wells is necessary to provide pipetting position feedback to the user. Following a number of failed attempts to create an effective classifier for detecting the position of pipetting tips in a scene, we opted to use color filtering in conjunction with a colored tip as a viable solution.

In our efforts to train the classifier, it quickly became apparent that the high transparency of a typical pipetting tip was an issue for discerning it in a scene. In fact, we found that instead of matching the tip, the classifier was matching objects based on the background color present in positive images. We also created a method for detecting the tip using background subtraction, but due to its significant impact on overall performance and celerity of the program, this approach was rejected. A final technique we attempted was to use the Features from Accelerated Segment

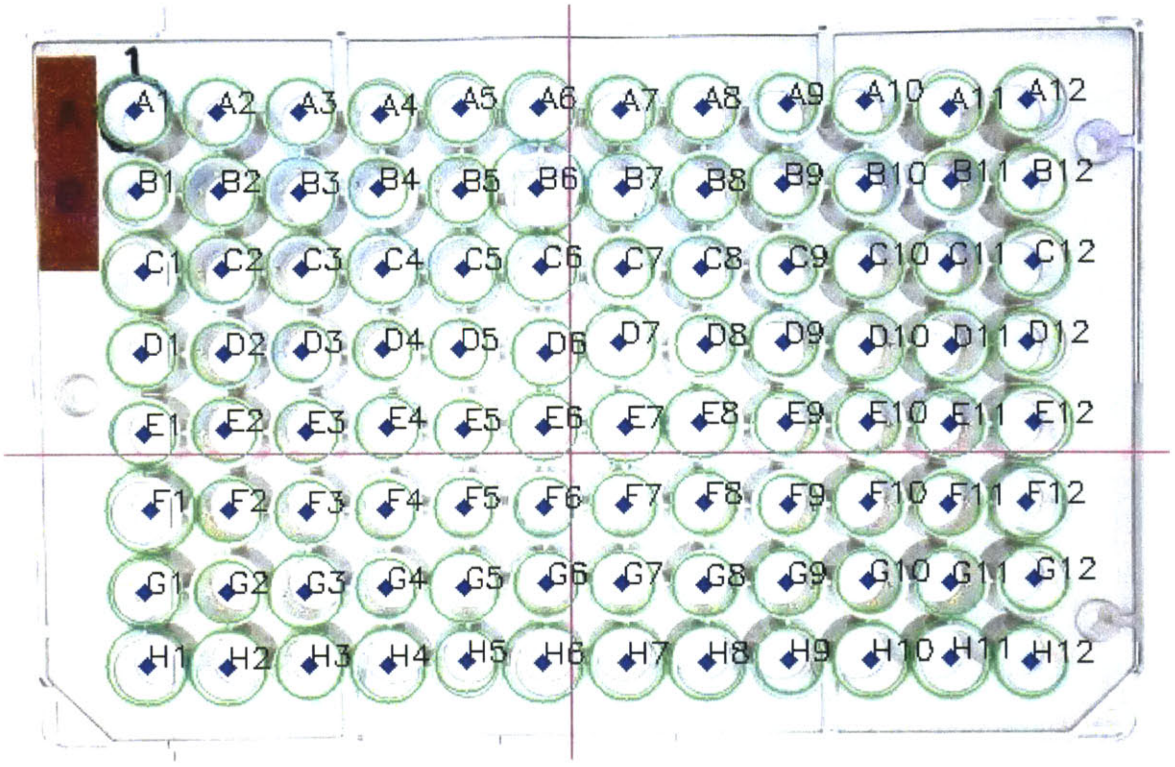


Figure 3-6: Final result of annotation process

Test (FAST) algorithm[81] to detect desirable features in the tip. Unfortunately, this approach worked inconsistently, often failing to find the very edge point of the pipette tip, leading to unreliable estimation of overall tip position.

In order to successfully and reliably track the position of the tip, we opted for tracking a custom-colored tip by applying color filtering to the frame. The choice of color is arbitrary and can be readily changed. We settled on blue in order to provide a larger contrast with the iPad present in the preferred configuration of the system. The result of the color filtering is shown in Figure 3-7.

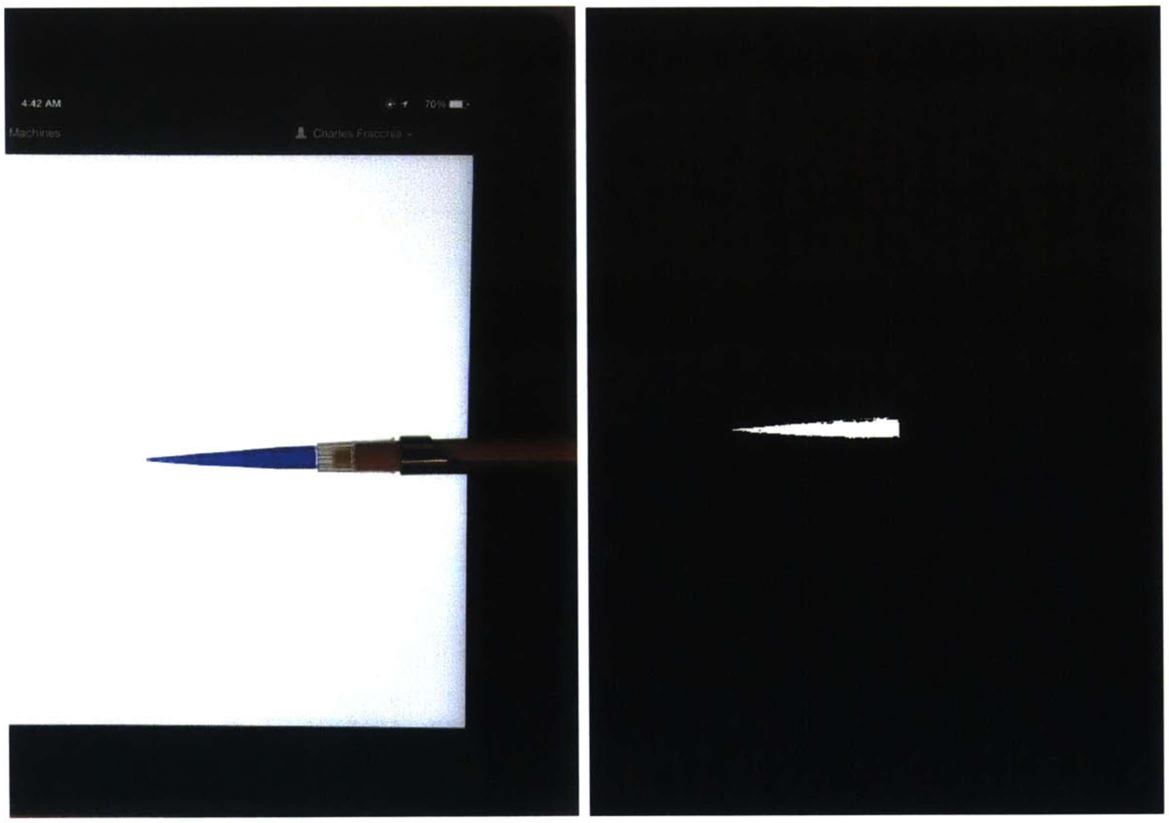


Figure 3-7: Color-filtering process applied to tip tracking. Left: The raw image with a tip colored blue. The tip was colored more than necessary for effect. Right: The result of the color filtering mask showing only the tip.

3.4 Volume Sensing

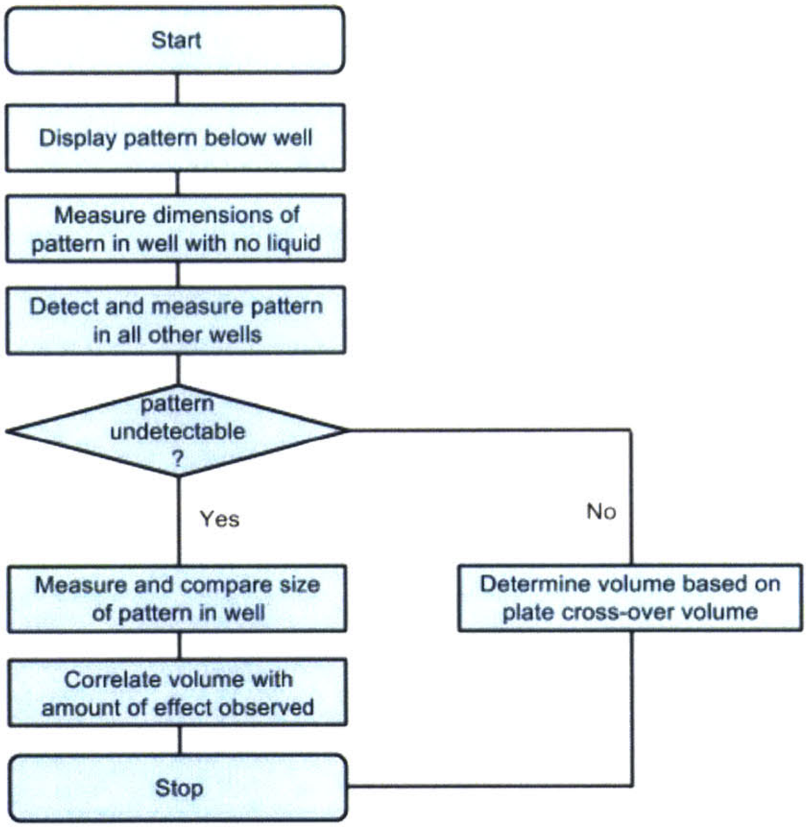

Start
Display pattern below well
Measure dimensions of pattern in well with no liquid
Detect and measure pattern in all other wells
pattern undetectable ?
Yes
No
Measure and compare size of pattern in well
Determine volume based on plate cross-over volume
Correlate volume with amount of effect observed
Stop

Figure 3-8: Algorithm for volume detection using overhead video from the USB camera. The plate cross-over volume is determined as the minimum volume necessary for forming continuous layer surface inside of the well. This volume varies depending on the specific plate used and the diameter of the bottom of the well

Using the same overhead camera, we outline an algorithm to detect the volume of liquids contained in individual wells. The proposed algorithm is outlined in Figure 3-8. At the center of its implementation, the algorithm uses a pre-defined pattern to calculate how much volume there is in the well based on phenomenon that the liquid in the well acts like a lens with respect to the pattern displayed below by the tablet. The principle is demonstrated in Figure 3-9.

It is important to note that this current implementation of volume sensing rests upon the use of cylindrically shaped wells inside the plate. We will extend the approach in future to allow for a greater diversity of well profiles.

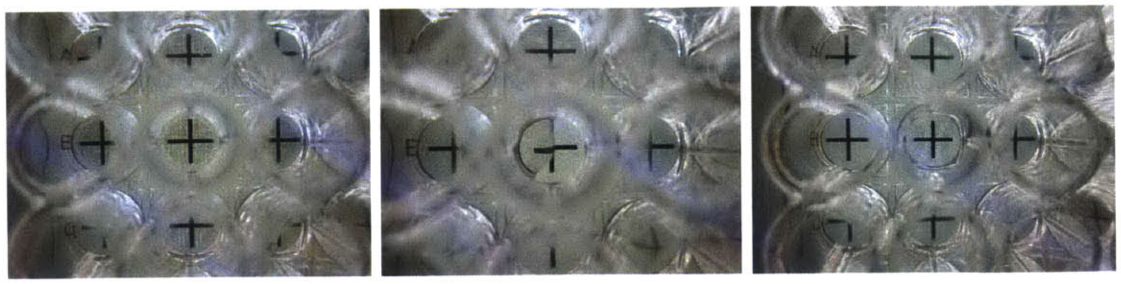


Figure 3-9: Left: Well with no water inside. Center: Well with 40 µL which is the cross over volume for this well, destroying the pattern. Right: demagnifacation effect caused by the liquid in the well. In this case: 100 µL

3.5 User Interface Feedback

Building a closed-loop system meant for human operation also implies providing feedback to the user. This is achieved in real-time in our system by using the PDL Web interface, where we display a drawing of the well being used and highlight the well where the tip is placed. An example of this system is shown in Figure 3-10. The computer vision portion of the system communicates which well to highlight by publishing the highlighted well message to the centralized MQTT[37] broker.

3.6 Pipette Control

Now that we have real-time positional feedback of the pipette tip, all we need to achieve a closed-loop system is the ability to control the pipetting process. Normal manual pipettes are not suitable for this as they rely on manual actuation by the scientist and do not possess an electronic interface. We therefore need to employ an electronic pipette in order to achieve our desired closed-loop system. While there is a healthy offering of different electronic pipettes[55][18][47][22][51][26], only two were confirmed to have a communication interface (one of which was discontinued). We chose to work with the Rainin EDP3 pipette because of availability within the lab and low-cost procurement options (eBay). It is important to note that despite their remote control ability, neither of the two electronic pipettes has open specifications for their communication port. This is mainly because, as discussed in a previous chapter,

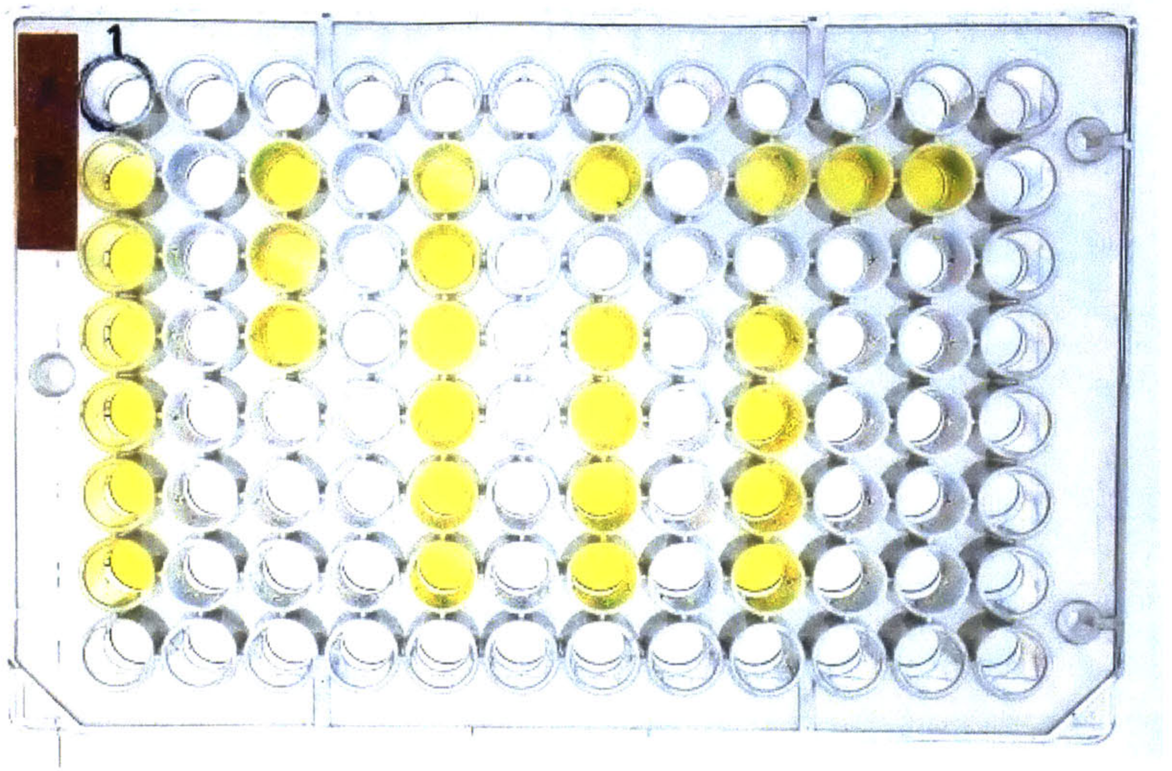


Figure 3-10: The MIT logo is drawn on the well plate using the closed-loop pipetting system developed in this work

most bio-instrumentation companies see software as a separate revenue model that open communications specifications could negate.

3.6.1 Reverse Engineering the EDP3

Due to the lack of an open communication specifications for the chosen Rainin EDP3 pipette, we were forced to reverse engineer it. By inspecting the pipette electronics closely, we discovered that the battery-charging port (see Figure 3-11) present at the back of the bottom of the battery compartment had more traces beyond the expected battery voltage and ground.

Using a logic analyzer[50] wired to each of the terminals on the battery connector, we were able to collect all communications from the EDP3 (see Figure 3-12). However, in its normal state, the EDP3 pipette did not seem to output any data. Despite the pipette being a discontinued device, and thanks to the help of a Rainin engineer, we

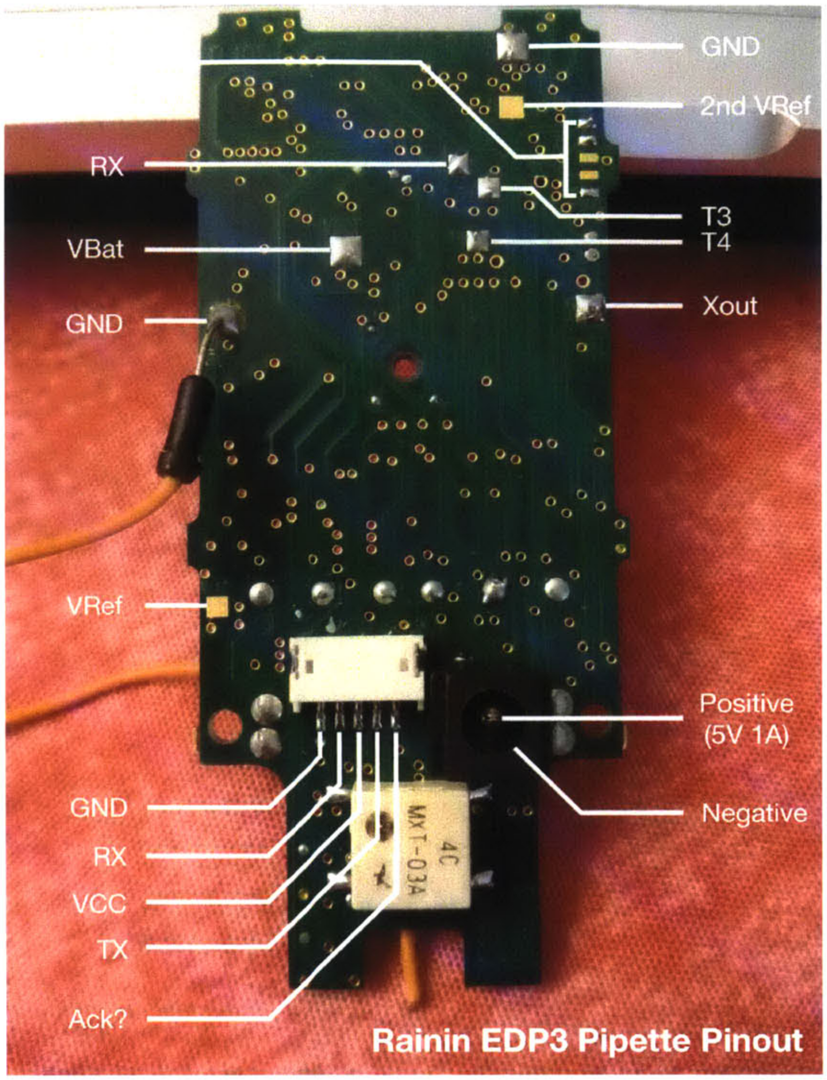


Figure 3-11: The annotated view of the EDP3 electronics following reverse engineering

were able to obtain a compiled version of the Windows utility used to control the EDP3 remotely. Using two USB-to-Serial-FTDI cables and reversing the transmit and receive wires, we were able to collect the serial commands used to communicate with the pipette. Appendix C contains a summary table of packet structure, available commands and sample packets.

3.6.2 Custom Interface Board for the EDP3

Having reverse engineered the pin functions of the connector present in the battery compartment of the EDP3, we designed a custom circuit board to enable wireless

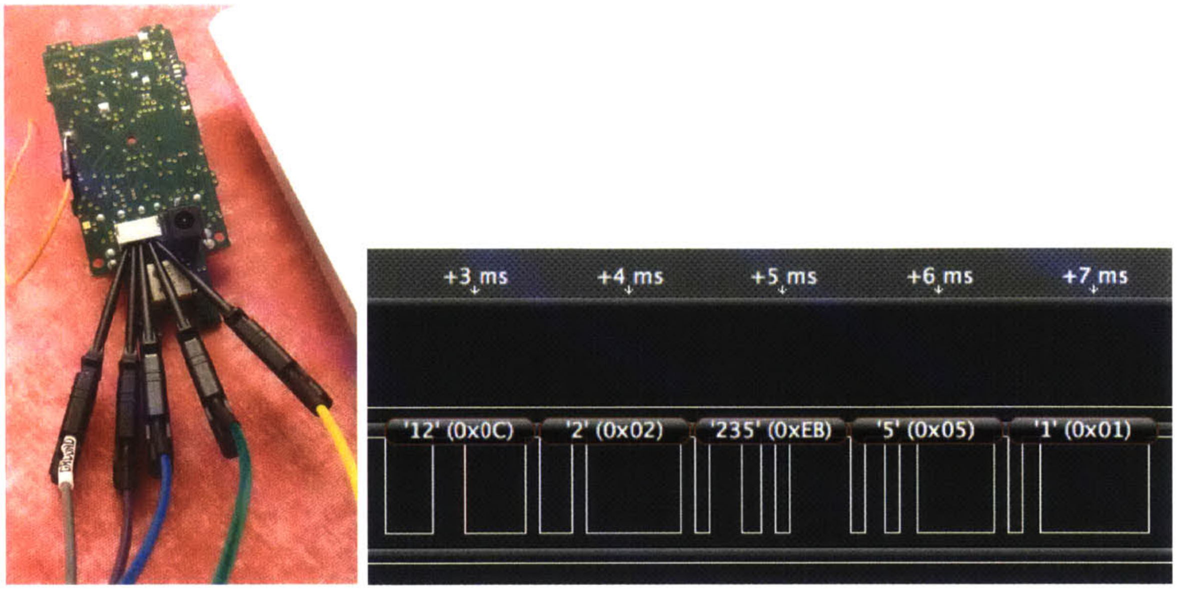


Figure 3-12: Left: We connected a logic analyzer to the EDP3 battery port to capture and reverse engineer the pipette's communication port. Right: Captured response packets on the response line after sending data to the EDP3 using the remote communication program. Program courtesy of Steve Konrad.

control of the pipette. We used an XBee[14] operating over the 802.15.4 wireless communication layers to achieve the wireless bridge. Due to the simplicity and single-sided nature of the designed board (see Figure 3-13), we fabricated our own circuits using a small 3-axis circuit mill[49]. The board makes use of the JST ZH 5-position connectors natively used in the EDP3 pipette. This allows us to enhance the capability of the pipette by installing our board *in line* with the rest of the system, making installation both easy and reversible. The only modification that is required is the use of a larger back plate due to the greater thickness of the overall battery compartment. To solve this problem, we have designed a 3-D printable back cover and made it available on the popular 3-D design sharing Website Thingiverse[2]. In Figure 3-13, the original EDP3 battery can be seen below the circuit board in black. It is connected using its native connector to the circuit board and provides power for the whole system, including the added wireless connector. While the current design of the wireless bridge is relatively power-hungry (approximately 40 mA running), it does not make use of the sleeping capabilities of the radio. These capabilities are demonstrated later in this thesis as part of the wireless sensor platform that was designed. In this case however, we found that the native battery was able to sustain the whole modified system (pipette + wireless radio) for a little over three hours. This autonomy was found to be sufficient especially when taking into account that the EDP3 is designed to recharge its battery every time it is replaced on its stand.

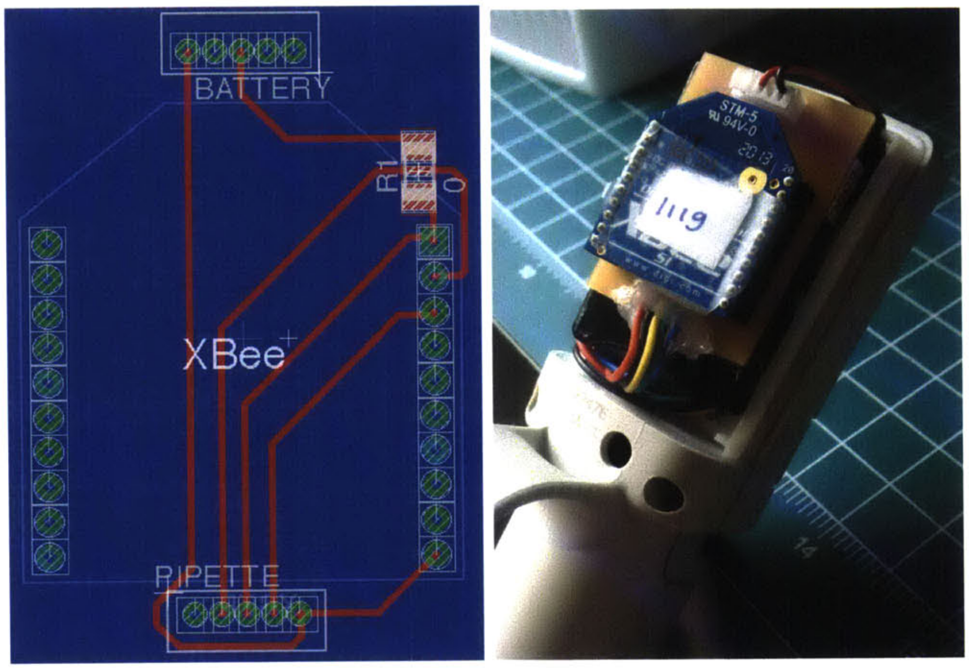


Figure 3-13: Left: Board layout for the wireless adaptor using the exposed communication port on the Rainin EDP3 pipette. Right: The wireless board installed in the battery compartment of the EDP3. The board was designed to maximize backwards compatibility and require as few modificiations as possible.

3.6.3 Open Communication Library in Python: pyEDP3

Using the custom interface board describe above allows us to send the EDP3 commands directly to the pipette. However, that is still an arduous process given that the command structure uses raw bytes to be passed through the serial port. To simplify the process, we have created a Python package named pyEDP3. This packages makes crafting and parsing of EDP3 commands significantly easier and is available on GitHub[46]. An example using the pyEDP3 package to send commands to the wireless-enabled EDP3 pipette is outlined in Appendix D. Notably, this package allows the user to interact with the pipette using english language aliases for each command instead of the byte equivalents.

Chapter 4

Continuous Monitoring of Sample Temperature

Experimental temperature conditions have been shown to significantly affect a number of biological processes[73][77][69][80]. Yet it is currently difficult to collect sample temperature data continuously. To solve this problem, we designed an open battery-powered temperature sensor designed to fit on top of an Eppendorf tube, a common format for containing samples in biology labs. The sensor is designed to be carried *alongside* the experiment's samples and thus be exposed to the same temperature conditions. This approach prevents sample contamination by not requiring that the sensor be placed inside of the sample solution. The sensor interfaces with the PDL system to provide real-time sample temperature information to the scientist as well as store the data for future analysis, including cross-experiment reproducibility assessments.

4.1 Hardware Design

The hardware was custom designed with the following requirements in mind: wireless communication, battery operation and overall small footprint, with the temperature sensor portion being able to fit within an Eppendorf tube. Alongside this thesis, we release the schematics, firmware and other software as open hardware and software available on GitHub[19]. We would like to acknowledge Andrew Payne for his contribution to the design and production of the sensor platform.

4.1.1 Electronics Architecture

Figure 4-1 shows the overall electronics design of the sensor. The platform revolves around the ATTiny85 microcontroller[7], which regulates communications via the wireless transceiver and sensing of temperature through the digital temperature sensor. This particular microcontroller was chosen because of its small footprint and larger memory (8 KB) compared to the ATTiny45 (4 KB). For accessibility and openness reasons, we opted to use the Arduino framework[6] to program the firmware on board the microcontroller. The whole platform is powered by a 150 mA h battery ideally sized to fill the space between the printed circuit board (PCB) and the bottom of the XBee module for optimal space usage.

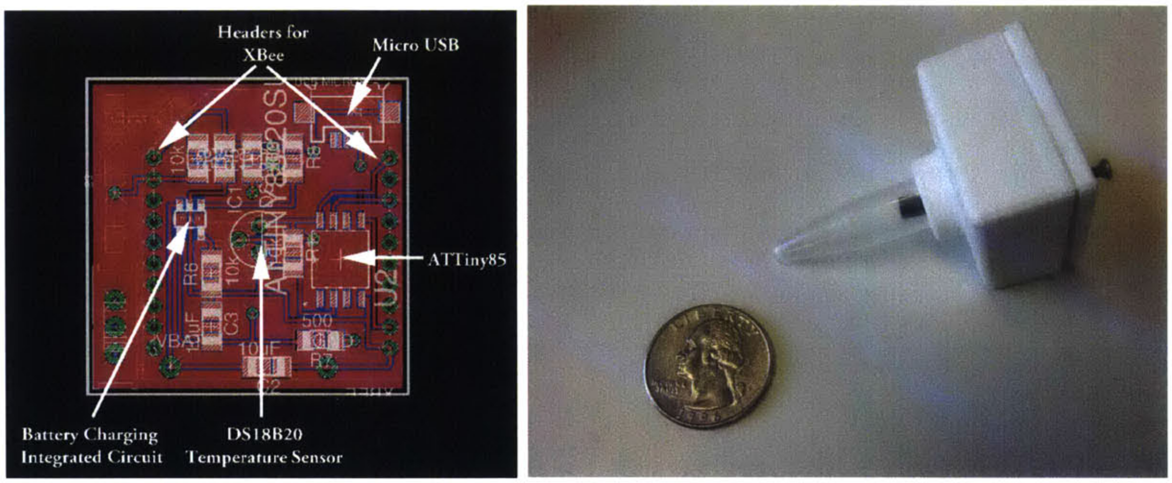

Headers for XBee
Micro USB
ATTiny85
Battery Charging Integrated Circuit
DS18B20 Temperature Sensor

Figure 4-1: Left: Sample wireless temperature sensor electronic annotated board design. Right: Assembled prototype within case and with sensor inside the Eppendorf tube

4.1.2 Temperature Sensor

The current revision of the hardware makes use of the Maxim DS18B20[32] digital temperature sensor. This was initially chosen for its use of the 1-wire protocol, which only requires a single wire for digital communication[1], compared to an otherwise minimum of two wires like I^2C. However, this design choice is currently being reconsidered in favor of a more generic approach that leverages I^2C due to the wider availability of precision temperature sensors using the protocol. Furthermore, tests summarized in Figure 4-2 highlight the slow adaptation time of the DS18B20 sensor to cold temperatures. This is unsuitable for our purposes as the DS18B20 can take up to 20 minutes to adapt to the cold temperatures expected in ice-water baths used in biological experiments. We are therefore currently evaluating alternatives for the temperature sensor, yet by restricting ourselves to I^2C communication, we prevent the need for PCB layout changes for the main board.

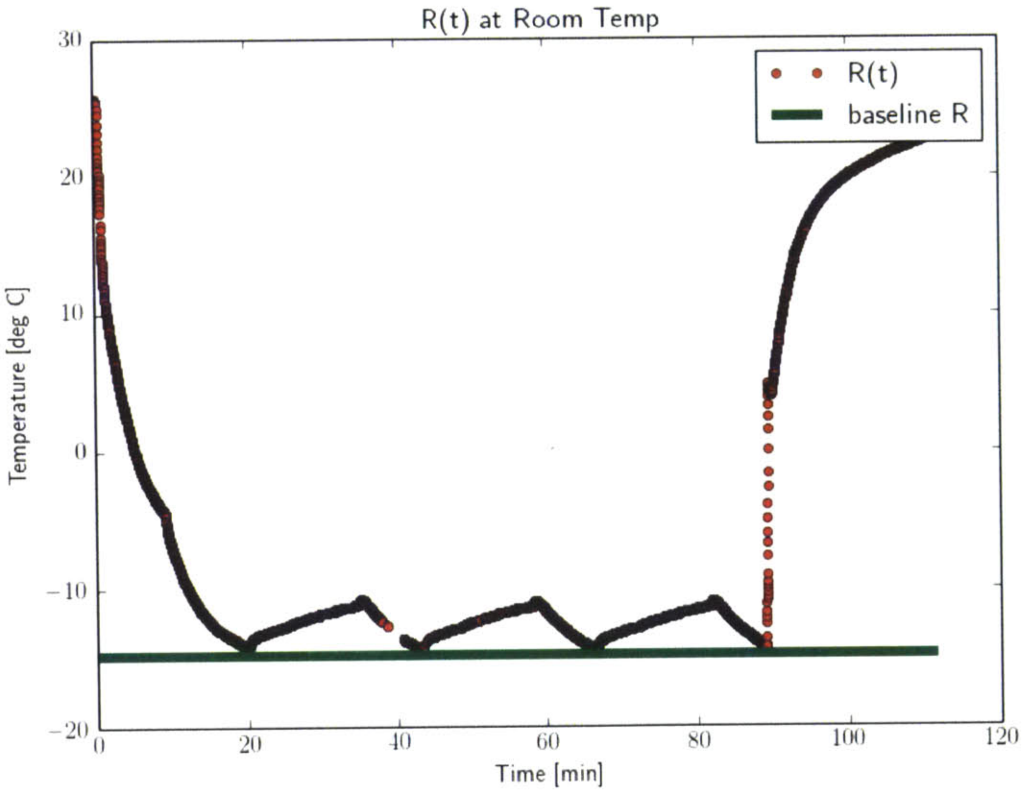


Figure 4-2: Temperature-sensing speed tests of the DS18B20. This shows that the DS18B20 is slow (approx. 20m) to adapt to large temperature changes, in particular when moving toward colder environments. This is in part due to the heat generated by the system by being continuously on. However, we found that different temperature sensors were able to adapt to the same differentials in a matter of seconds.

4.1.3 Temperature Calibration

In order to determine the real-world time constants of the temperature sensor, we conducted an experiment summarized in Figure 4-3. These experiments were conducted using the STLM75[52] I^2C temperature sensor due to the DS18B20's inadequately slow response time. Based on the result presented in Figure 4-3, the most adaptive configuration would be to use the sensor without a tube. However, two reasons make this option more difficult to carry out. First, this option would require thorough waterproofing of the electronics, adding significant complexity to the production and quality checking of the device. Second, the lack of a plastic tube does not reflect accurately the conditions of the sample liquid, which is the target of our measurements. The firmware used to collect this data from the STLM75 temperature sensor is available in Appendix E.

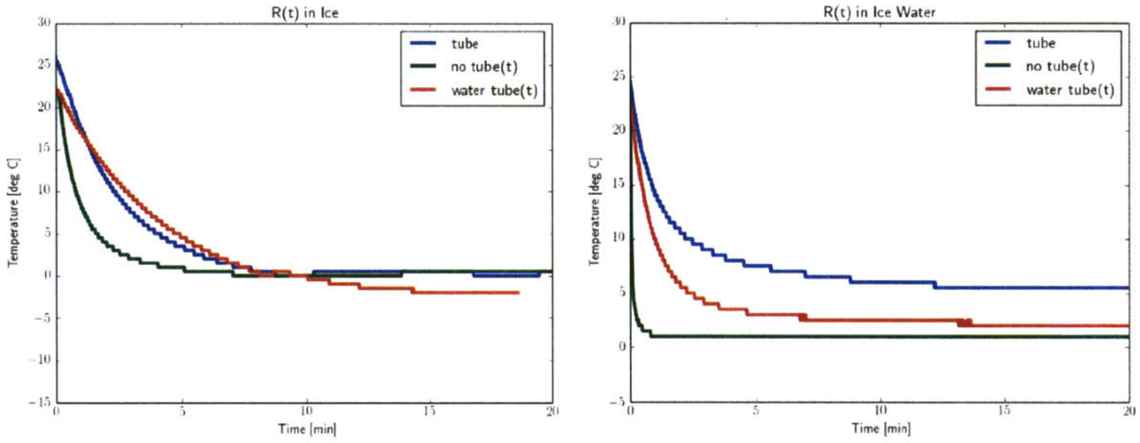


Figure 4-3: Temperature-adaptation experiment using the STLM75 temperature sensor with different holders in ice (left) and ice-water mix (right).

It can be seen in Figure 4-3 that the most efficient adaptation occurs with no tube covering the sensor. This is to be expected since the sensor is in more direct contact with the measured medium. The water tube is the second best due to the higher thermal conductivity of water compared to air in the case of the simple tube. However, it is important to note that the tube has a clear and significant insulating impact on the measurement system. Comparing the data from the ice-only (left) and ice-water (right) test cases, it is clear that the water plays an important role in speeding up the transition to the desired chilled temperature. This suggests, unsurprisingly, that in experiments where chilling the sample is time-sensitive, ice-water mixture should be sharply preferred to ice alone. Importantly, our sensor platform enables the user to verify that these parameters are reproducibly used.

These experiments were also carried out in heating environments common in biological experiments like that of the incubator to confirm behavior consistency (see Figure 4-4).

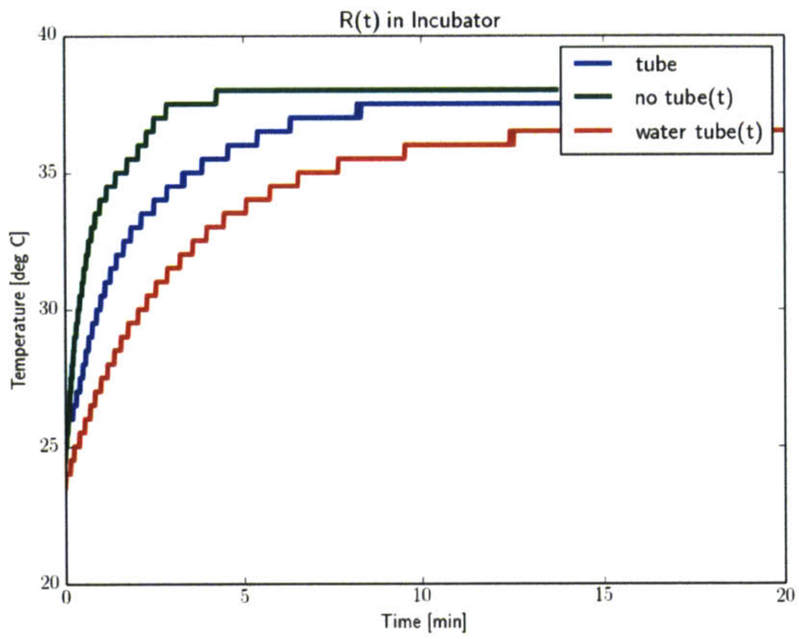


Figure 4-4: STLM75 temperature-sensor adaptation time validation experiments in heating conditions inside an incubator

4.1.4 Wireless Communication

The wireless communication portion of the sensor employs XBee modules[14] to implement a ZigBee 802.15.4 mesh network[15]. We opted for this radio module due to its maturity, documentation, flexibility and ability to run a true mesh topology, unlike most of its competitors[48][41]. While peak power consumption of the XBee during data transmission can be as high as 70 mA, we make use of the sleep pin to maximize time at low power modes where power consumption is rated at 3 µA. Early endurance tests have shown operation for several continuous days on a single charge.

Chapter 5
Deployment

5.1 Deploy or Die

"Deploy or Die" refers to the new approach for research introduced by the director of the MIT Media Lab Joi Ito[30]. This mantra places deployability, manufacturability, scale and robustness at the center of research objectives. This approach follows from Nicholas Negroponte's own approach at the Media Lab of "Demo or Die"[75], itself a departure from the common academic adage "Publish or Perish".

This new emphasis on deployability significantly affects the way one conducts research. In this thesis, we took particular care to build a system that is deployable and addresses the needs of the biological research community. It can be seen in the speed of development of the temperature sensor platform: under two months for full design, parts sourcing and manufacturable design from two individuals who had little prior experience with electronics. The overall approach is also reflected in the choice of technologies for the PDL system, opting for widely available, cross-platform compatible technologies to lower the barrier to entry as much as possible. Finally, our commitment to open software and hardware is key in achieving deployability into the real world, permitting other scientists and engineers to expand the system and its capabilities.

5.2 Rapid Prototyping

Rapid prototyping technologies played an essential role in reducing the time to having deployment-ready versions of the technologies described in this document. The early versions of electronic circuits were fabricated using a circuit board mill[49], allowing us to perform early development in-house. Using this method, functional circuits could be milled, assembled and tested in under 30 minutes, thus drastically increasing our ability to settle on circuit designs.

5.3 Production

Deployment of the technologies described in this document cannot be achieved without taking into account manufacturing constraints and volumes. While we expect the bulk of that effort to be achieved as further work, there are some key aspects that have arisen from our initial explorations. The systems that have been constructed and detailed in this document all consist of electronic devices and their respective enclosing cases. Therefore, production of said devices needs to encompass PCB fabrication, electronic assembly and injection molding of plastic cases.

When considering PCB fabrication and electronics assembly, the bill of materials (BOM) of the components and PCB complexity are key factors in determining the unit cost of the device. The bill of materials for the temperature sensor is summarized in Table 5.1. The PCB fabrication costs (not including assembly and testing) were estimated to be $3 per board, using a supplier in the United States. However, PCB fabrication alone is not sufficient to build a realistic estimate of the overall production. We are currently working with different partners to determine the cost of *stuffing* and testing our circuits either separately (potentially greater shipping costs) or combining this with PCB fabrication.

Finally, we are also currently examining the different options regarding the production of the casing. Injection molding prototyping facilities in the United States exist but often require a minimum of a few thousand dollars, making rapid prototyping costly and difficult. We are currently exploring capabilities more globally as part of a collaboration with the city of Shanghai in China, which should enable us to rapidly prototype molds far quicker and relatively inexpensively.

Item	Cost (At Quantity 100)
XBee	\$ 19
LiPo Battery (150 mA h)	\$ 4.76
Toggle Switch	\$ 0.46
Push Button	\$ 0.76
STLM75 Temperature Sensor	\$ 0.69
Micro USB Connector	\$ 0.30
Red LED	\$ 0.11
Green LED	\$ 0.10
Blue LED	\$ 0.15
ATTiny85 Microcontroller	\$ 1.12
LiPo Charging Integrated Circuit	\$ 0.42
Male In-Circuit System Programmer Header	\$ 0.83
XBee Female Socket	\$ 0.50
10 µF Capacitor	\$ 0.07
0.1 µF Capacitor	\$ 0.14
10 kΩ Resistor	\$ 0.01
499 Ω Resistor	\$ 0.01
Total	**\$ 29.29**

Table 5.1: Bill of materials (BOM) for the wireless temperature sensor device described in this thesis

5.4 Dissemination and Market Reach

Producing a device and system in a scalable manner is only part of the question when exploring deployment. The question of market validation and ultimately reach is crucial in achieving true dissemination. Otherwise the project risks, as it is often the case in academia, to fail due to the lack of customers or the product being *over engineered.* While we are currently very actively researching this aspect, there are some resources that we have been able to leverage at the institute that have proved invaluable in helping us avoid common mistakes and are having an important guiding influence on the process.

First, the Venture Mentoring Service[34] at MIT has been a key resource in helping us frame the problem within a deployment perspective, helping us develop a keener sense for notions of minimal viable product (MVP) and ensuring that our candidate devices for deployment hit a core pain point in our industry. In particular, this service is made all the more unique by its ability to understand the subtleties that emerge from seeking deployment as an academic metric of success, even if that means employing what may seem at first glance like a commercial strategy better suited for a stand-alone entity. We believe that *Deploy or Die* requires those lines to be blurred in order to successfully attain market dissemination as an academic goal and that VMS plays a central role in achieving this.

Second, the Media Lab itself has been home to a number of successful crowdfunding campaigns, combining exposure and a form of market validation. We believe a crowdfunding campaign for the projects described in this thesis may be an ideal way to do forward market validation on a smaller scale and go all the way through manufacturing and delivery of a few hundred units, the key in this case being the celerity with which we could design, build and deploy.

Chapter 6

Conclusions & Future Directions

In this work, we developed a language and tools that forms the basis for a context-aware biological research paradigm. It allows writing and execution of scientific protocols, providing real-time feedback of pipetting operations and temperature conditions of the experiment. All work performed in this thesis has been made openly available. The Protocol Descriptor interface is reachable at `www.pdl.io` allowing scientists to use the resources from this thesis for their experiments using any device with a Web browser. The code and instructions to enable pipetting feedback are available on GitHub[10], along with hardware schematics and firmware code for the networked temperature sensor[19].

As mentioned earlier, the work developed in this thesis sets out some of the necessary technologies for achieving context awareness in biological settings. We discuss future directions and desirable features of the system below.

6.1 Simulation Tools

Given the lengthy and complex nature of many biological protocols, system simulation plays an important role in shortening the design-build-test (DBT) cycle of experiments. However, the complexity of simulating biology bottom-up through all layers of rationalization (from molecular interactions to population dynamics) makes overall system simulation very difficult. Therefore, we plan to use data generated by the system described in this thesis to inform models and build simulation systems. To enable such integration, we plan to develop a public API, allowing the extraction of contextual experiment information from the PDL system. Visual simulation can also play an important role in the educational process of running experiments in the lab and partnerships with virtual lab environment makers[31] is a direction to be considered.

6.2 Analytics for Data-Driven Biological Research

The PDL system detailed in this work has the potential to change how experimental biology is executed from a predominantly qualitative process to a more quantitative, data-driven one. This can be achieved by embedding data analytics into the PDL Web interface and displaying quantitative information to the user. For example, using the system described in this thesis, we aim to provide average temperature distribution of individual steps to the user and generate an alert when it deviates more than a given percentage from the average. In this example, the average temperature of a step is computed using step start and stop times from the PDL system and sample temperature from the temperature sensor device to provide statistical information about the average duration of steps correlated with overall success of the protocol. This example shows how the system can provide unique metrics to improve scientific reproducibility.

6.3 Contextual Sensors

Future work on this project will also include the design of more wireless sensors to allow a more thorough collection of experimental variables (e.g., carbon dioxide, humidity, etc). While these sensors are macroscopic, we can foresee the footprint and scale of these sensors being reduced by orders of magnitudes in the future, not only through electronics miniaturization, but also by using other technologies and approaches like synthetic biology, to create sensors that operate at the same scale as the target molecules and processes. Past work[68] serves as an example how a bacterium could be reprogrammed to produce an electronically sensible reporter to achieve far greater context awareness by creating a hybrid bio-electronic interface.

Appendix A
Protocol Descriptor Language Specification v4

```
<protocol>
  <id></id>
  <name></name>
  <description></description>
  <created></created>
  <relationship></relationship>
  <samples>
    <sample>
      <id></id>
      <container></container>
      <contents></contents>
    </sample>
  </samples>
  <steps>
    <step>
      <number></number>
      <name></name>
      <description>[Comment #1]
        <comment>Comment Content</comment>
      </description>
      <difficulty></difficulty>
      <duration>10m</duration>
      <onStart>
        <timer>10m</timer>
      </onStart>
      <onClick>
```

```
        <timer>10m</timer>
      </onClick>
      <onComplete>
        <notify>
          <email />
          <phone />
        </notify>
      </onComplete>
    </step>
  </steps>
</protocol>
```

Appendix B
Closed-Loop Pipetting Computer Vision Code

```python

import numpy as np
import cv2, random, os
import time, sys, math
import mosquitto

windowName = "Well Classifier"
cv2.namedWindow(windowName, cv2.CV_WINDOW_AUTOSIZE)
camera_index = 0
cam = cv2.VideoCapture()
cam.open(camera_index)
fgbg = cv2.BackgroundSubtractorMOG()                    # Used in background substraction

def addMissingWells(frame, axisWells, columns, avgDistance, avgDiameter, vx, vy, cx, cy, bufferPixels=10, draw=True):
    """Add the non detected circles to the list"""
    pass
    #print "Average distance between wells: ", avgDistance
    if len(axisWells) != columns:
        for k, well in enumerate(axisWells):
            if k == 0:
                pass
            elif k == len(axisWells)-1:
                pass
            else:
                distanceToPrevious = distanceBetween((well[0],well[1]), axisWells[k-1])
                #print "Well: ",well
                #print "Distance to previous: ", distanceToPrevious
                if distanceToPrevious > avgDistance+bufferPixels:
                    #print "Distance Greater than Expected: should add missing circle"
                    xVal = axisWells[k-1][0]+avgDistance-bufferPixels/2
                    yVal = vy*xVal + cy-vy*(cx+10)
                    axisWells.insert(k, (np.float32(xVal), yVal[0], avgDiameter))
                    drawCircle(frame, (np.float32(xVal), yVal[0], avgDiameter), (0,0,255))

    return axisWells

def annotateAllWells(frame, A1, Aend, rows, columns, allCircles, avgDistance, avgDiameter, bufferPixels=5, draw=True):
    """Label all the wells based on giving the corner wells """
    pass
    alphabet = ["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z"]
    annotatedWells = {}
    wellsByAxis = []
    corner = A1
    end = Aend
    for currRow in xrange(rows):
        cArray = []
        try:
            cArray.append(np.array([corner['circle'][0], corner['circle'][1]]))
            cArray.append(np.array([end['circle'][0],end['circle'][1]]))
        except TypeError:
            print "Moose under a train"
            break

        vx, vy, cx, cy = cv2.fitLine(np.array(cArray), cv2.cv.CV_DIST_L2, 0, 0.01, 0.01)
        if draw:
            cv2.line(frame, (int(cx-vx*width), int(cy-vy*width)), (int(cx+vx*width), int(cy+vy*width)), (0, 0, 255))
        axisWells = sorted(findAllWellsOnAxis(frame, allCircles, vx, vy, cx, cy))
        #axisEquation = vy*well[0] + cy-vy*(cx+10)
        if len(axisWells) < columns:
            axisWells = addMissingWells(frame, axisWells, columns, avgDistance, avgDiameter, vx, vy, cx, cy)
            #print "ERROR: Found a fewer wells than expected on axis %s!!!" % alphabet[currRow]

        wellsByAxis.append( axisWells )

        previousWell = "099"
        for i, well in enumerate(axisWells):
            #print "i+1: %s and previouswell:%s" % (i+1, int(previousWell[1:]))
            if i != 0:
                if i+1 == int(previousWell[1:]):
                    pass
                    #print "Skipping this well because already assigned due to skip"
                else:
                    distance = distanceBetween(annotatedWells[previousWell],well)
                    if distance > avgDistance+bufferPixels:
                        #print "*****"
                        #print "Distance is greater than the average one"
                        #print "*****"
                        hops = round(distance / avgDistance)
                        wellAddress = "%s%s" % (alphabet[currRow],int(i+hops))
                        cv2.putText(frame,wellAddress,(well[0],well[1]), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255))
                        #print "Assigning: %s" % wellAddress
                        annotatedWells[wellAddress] = well
                        previousWell = wellAddress
                    else:
                        wellAddress = "%s%s" % (alphabet[currRow],i+1)
                        #print "Well Address: ",wellAddress,"\t Val:",(well[0],well[1]),"\t type: ",type(well[0]),'\t',type(well[1])
                        cv2.putText(frame,wellAddress,(well[0],well[1]), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,0))
                        annotatedWells[wellAddress] = well
                        previousWell = wellAddress
            else:
                    wellAddress = "%s%s" % (alphabet[currRow],i
                      +1)
                    cv2.putText(frame,wellAddress,(well[0],well
                      [1]), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                      (0,0,0))
                    annotatedWells[wellAddress] = well
                    previousWell = wellAddress

        #print annotatedWells
        allCircles = removeCirclesFromList(allCircles,
          axisWells)
        corner = findCircleClosestTo(corner['circle'],
          allCircles, 10)
        end = findCircleClosestTo(end['circle'], allCircles,
           10)

    return annotatedWells


def calculateAvgDiameter(circles):
    """Calculate the average distance between circles"""
    pass
    diameters = [circle[2] for circle in circles]
    return np.mean(diameters)

def calculateAvgDistance(circles):
    """Calculate the average distance between circles"""
    pass
    distances = []
    for i, circle in enumerate(circles[:len(circles)-1]):
        for secondCircle in circles[i+1:]:
            distance = distanceBetween(circle,secondCircle)
            #print distance, circle[2]
            if distance < 3*circle[2]:
                distances.append(distance)

    return np.mean(distances)

def distanceBetween(pt1,pt2):
    """Calculate the distance between two points"""
    pass
    return math.sqrt((pt2[0] - pt1[0])**2 + (pt2[1] - pt1
      [1])**2)

def doColorMask(frame,lowerColor,higherColor):
    """df"""
    pass
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    lowerColor = np.array(lowerColor)
    higherColor = np.array(higherColor)
    mask = cv2.inRange(hsv, lowerColor, higherColor)
    res = cv2.bitwise_and(frame, frame, mask=mask)

    return hsv, mask, res

def doHoughDetection(frame,grayFrame,circleBounds,cannyUpperBound=50,minCenterDistance=20,draw=True):
    """Search for circles in frame and return annotated frame"""
    pass
    circles = cv2.HoughCircles(grayFrame,\
                               cv2.cv.CV_HOUGH_GRADIENT,\
                               1,\
                               minCenterDistance,\
                               param1=cannyUpperBound,\
                               param2=30,\
                               minRadius=circleBounds[0],\
                               maxRadius=circleBounds[1]\
                           )
    if circles != None:
        for i in circles[0,:]:
            if draw:
                cv2.circle(frame,(i[0],i[1]),i[2],(255,0,0),1)  # draw the outer circle
                cv2.circle(frame,(i[0],i[1]),2,(255,0,0),3)  # draw the center of the circle
    try:
        return frame, circles[0]
    except TypeError:
        return frame, None

def doBGSubstraction(frame, addMask=None, maxCorners=50, minDistance=5, draw=True):
    """Does the background subtraction and returns points for the corners detected """
    pass
    unraveledCorners = []
    fgmask = fgbg.apply(frame)
    if addMask != None:
        cv2.imshow("Add Mask", addMask)
        fgmask = fgmask - addMask
    else:
        print "addMask is None"
    cv2.imshow("Post Mask", fgmask)
```

```python
    edges = cv2.Canny(fgmask,1,100)
    corners = cv2.goodFeaturesToTrack(fgmask,maxCorners
       ,0.01,minDistance)
    if corners != None:
        corners = np.int0(corners)
        #print "Found %s corners" % len(corners)
        for i in corners:
            x,y = i.ravel()
            unraveledCorners.append((x,y))
            if draw:
                drawCircle(frame,(x,y,10),(255,0,255))

    return unraveledCorners


def drawCircles(frame, circles, color=(0,0,0)):
    """Draws a all circles to the frame"""
    pass
    try:
        frame = cv2.cvtColor(frame, cv2.COLOR_GRAY2BGR)
    except:
        pass
    if circles != None:                          #Do only if
       there are circles detected
        for circle in circles:
            drawCircle(frame,circle,color)

def drawCircle(frame, circle, color=(0,0,0)):
    x = circle[0]
    y = circle[1]
    r = circle[2]
    cv2.circle(frame, (int(x),int(y)), int(r), color, 1)
def drawLine(frame,pt1,pt2, color=(0,0,0)):
    try:
        frame = cv2.cvtColor(frame, cv2.COLOR_GRAY2BGR)
    except:
        pass
    cv2.line(frame, pt1, pt2, color, 1 ,8, 0)
def drawQuadrants():
    drawLine(frame,(width/2,0),(width/2,height),(255,0,255))
    drawLine(frame,(0,height/2),(width,height/2),(255,0,255)
       )

def findCircleClosestTo(position, circles, minDistance=0):
    """Find closest circle to a set point"""
    pass
    x = position[0]
    y = position[1]
    closest = {"circle":None, "distance":500000}
    for circle in circles:
        distance = math.sqrt((circle[0] - x)**2 + (circle[1]
            - y)**2)
        if distance < closest['distance'] and distance >
          minDistance:
            #print "Is new closest, with distance: %s (
              previous best:%s)" % (distance, closest['
              distance'])
            closest['distance'] = distance
            closest['circle'] = circle
        else:
            #print "Is farther"
            continue

    return closest
def findAllWellsOnAxis(frame,wells,vx,vy,cx,cy,bufferPixels
  =5):
    """This function calculates the line's equation and
      returns the circles that are on it within small buffer
      """
    pass
    correctWells = []
    yIntersect = cy-vy*(cx+10)
    for well in wells:
        if well[1] < vy*well[0]+yIntersect+bufferPixels and
          well[1] > vy*well[0]+yIntersect-bufferPixels:
            correctWells.append(well)
            drawCircle(frame, well, (0,255,0))

    return correctWells



def findTipPoint(frame, cnt, draw=True):
    """docstring for findTipPoint"""
    pass

    white = np.nonzero(cnt)
    #print white
    #print "***"
    xArr = []
    yArr = []
    for i in xrange(len(white[0])):
        xArr.append(white[1][i])
        yArr.append(white[0][i])
```

```python

    #print xArr, yArr
    if xArr != [] and yArr != []:
        minX = min(xArr)
        maxX = max(xArr)
        minY = min(yArr)
        maxY = max(yArr)

        leftMost = ( xArr[xArr.index(minX)], yArr[xArr.index
          (minX)] )
        rightMost = ( xArr[xArr.index(maxX)], yArr[xArr.
          index(maxX)] )
        upMost = ( xArr[yArr.index(minY)], yArr[yArr.index(
          minY)] )
        downMost = ( xArr[yArr.index(maxY)], yArr[yArr.index
          (maxY)] )

        #print "leftMost:",leftMost
        #print "rightMost:",rightMost
        #print "upMost:",upMost
        #print "downMost:",downMost
        #return leftMost, rightMost, upMost, downMost
        if draw:
            drawCircle(frame, (leftMost[0], leftMost[1], 15)
              , (0,255,0))
            drawCircle(frame, (rightMost[0], rightMost[1],
              15), (0,255,0))
            drawCircle(frame, (upMost[0], upMost[1], 15),
              (0,255,0))
            drawCircle(frame, (downMost[0], downMost[1], 15)
              , (0,255,0))
        return leftMost
    else:
        return None

def findTipPointOLD(tipPoints, imgWidth):
    """Finds the tip most point among candidate tip points
      """
    pass
    sortedPoints = sorted(tipPoints)
    print sortedPoints
    if sortedPoints != []:
        minMaxDist = distanceBetween(min(sortedPoints),max(
          sortedPoints))
    else:
        minMaxDist = 0
    print "MinMax Distance: %s" % minMaxDist
    if minMaxDist > 500:
        if sortedPoints != []:
            #print [point for point in sortedPoints]
            avg = np.mean([point[0] for point in
              sortedPoints])
            #print "Average: %s" % avg
            minPoint = min(sortedPoints)
            maxPoint = max(sortedPoints)
            #print "Min: %s Max: %s" % (minPoint[0],maxPoint
              [0])
            if avg > imgWidth/2:                #Right handed
                tipPoint = minPoint
            elif avg < imgWidth/2:              #Left handed
                tipPoint = maxPoint
            else:
                tipPoint = None
            return tipPoint
        else:
            return None
    else:
        return None

def getFeaturesFromMask(mask, maxCorners=50, minDistance=5,
  draw=True):
    """docstring for getFeaturesFromMask"""
    pass
    unraveledCorners = []
    edges = cv2.Canny(mask,1,100)
    corners = cv2.goodFeaturesToTrack(mask,maxCorners,0.01,
      minDistance)
    if corners != None:
        corners = np.int0(corners)
        #print "Found %s corners" % len(corners)
        for i in corners:
            x,y = i.ravel()
            unraveledCorners.append((x,y))
            if draw:
                drawCircle(frame,(x,y,3),(255,0,255))

    return unraveledCorners

def normalizeCircle(circle):
    """docstring for normalizeCircle"""
    pass
    return (circle[0], circle[1], circle[2])

def normalizeCircles(circleArray):
    normalizedCircles = []
    for circle in circleArray:
        normalizedCircles.append(normalizeCircle(circle))
    return normalizedCircles
def on_mqtt_connect(rc):
    if rc == 0:
    #rc 0 successful connect
        print "Connected to MQTT broker"
    else:
        raise Exception

def on_mqtt_disconnect(rc):
    if rc == 0:
    #rc 0 successful connect
        print "Disconnected from MQTT broker"
    else:
        raise Exception

def pushToMQTT(topic, message):
    """Push specific message to specific topic on dime.
      smartamerica.io broker"""
    pass
    broker = "dime.smartamerica.io"
    port = 1883
    #create an mqtt client
    mypid = os.getpid()
    client_uniq = "arduino_pub_"+str(mypid)
    mqttc = mosquitto.Mosquitto(client_uniq)
    #connect to broker
    mqttc.connect(broker, port, 60, True)
    mqttc.on_connect = on_mqtt_connect
    mqttc.on_connect = on_mqtt_disconnect
    mqttc.publish(topic, message)
    print "Published %s to topic: %s" % (message, topic)
    mqttc.disconnect()
def removeCirclesFromList(circleList,circlesToRemove):
    """This function will the 'circlesToRemove' from the
      circleList"""
    pass
    result = list(circleList)
    for remCircle in circlesToRemove:
        try:
            result.remove(remCircle)
        except ValueError:
            pass
    return result

```

```
def rotateFrame(frame, angle):
    h = frame.shape[0]
    w = frame.shape[1]
    M = cv2.getRotationMatrix2D((w/2,h/2), angle, 1)
    rotated_frame = cv2.warpAffine(frame,M,(w,h))
    return rotated_frame

def scaleFrame(frame,scale):
    """docstring for scaleFrame"""
    pass
    height, width = frame.shape[0:2]
    height_display, width_display = scale * height, scale *
      width
    # you can choose different interpolation methods
    frame_display = cv2.resize(frame, (int(width_display),
      int(height_display)), interpolation=cv2.INTER_CUBIC)
    return frame_display

def wellsWithin(circles,xBounds,yBounds):
    """"""
    pass
    circlesWithin = []
    for circle in circles:
        if xBounds[0] < circle[0] < xBounds[1] and yBounds
          [0] < circle[1] < yBounds[1]:
            circlesWithin.append(circle)

    return circlesWithin

def whichWell(tipPoint, annotatedWells, avgDiameter):
    """Determines which well the tip is in"""
    pass
    currentWell = None
    for well in annotatedWells:
        #print "Checking:", well
        dist = distanceBetween(tipPoint,(annotatedWells[well
          ][0],annotatedWells[well][1]))
        if dist < avgDiameter+avgDiameter/4:
            currentWell = well
            break
    return currentWell


previousWell = None
fixedCamera = False
while 1:
    ret, frame = cam.read()
```

```python
    if not fixedCamera:
        frame = scaleFrame(frame,0.5)
        rawFrame = frame.copy()
    else:
        frame = rotateFrame(frame,180)
    height, width, depth = frame.shape
    grayFrame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    if not fixedCamera:
        hsv, mask, res = doColorMask(frame,[99, 150, 127],[124, 255, 255])
        corners = getFeaturesFromMask(mask)
        ret,thresh = cv2.threshold(mask,127,255,0)
        contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
        cv2.drawContours(frame,contours,-1,(0,255,0),1)
        tipPoint = findTipPoint(frame, mask)
        #cv2.imshow('Color Mask', mask)

    drawQuadrants()

    if fixedCamera:
        frame, circles = doHoughDetection(frame, grayFrame, [10,45]) #Used when scaling down
    else:
        frame, circles = doHoughDetection(frame, grayFrame, [15,30])
    #cv2.imshow(windowName, frame)
    if circles != None:
        circles = normalizeCircles(circles)
        avgDistance = calculateAvgDistance(circles)
        avgDiameter = calculateAvgDiameter(circles)
        circlesWithin = wellsWithin(circles,[0,width/2],[0,height/2])       #Within top left quadrant
        A1 = findCircleClosestTo((0,0), circlesWithin)
        if A1['circle'] != None:
            drawCircle(frame, A1['circle'], (255,0,255))

        circlesWithin = wellsWithin(circles,[width/2,width],[0,height/2])
        A12 = findCircleClosestTo((width,0), circlesWithin)
        if A12['circle'] != None:
            drawCircle(frame, A12['circle'], (255,0,255))

        circlesWithin = wellsWithin(circles,[0,width/2],[height/2,height])
        H1 = findCircleClosestTo((0,height), circlesWithin)
        if H1['circle'] != None:
            drawCircle(frame, H1['circle'], (255,0,255))

        circlesWithin = wellsWithin(circles,[width/2,width],[height/2,height])
        H12 = findCircleClosestTo((width,height), circlesWithin)
        if H12['circle'] != None:
            drawCircle(frame, H12['circle'], (255,0,255))

        if A1['circle'] != None and A12['circle'] != None and H1['circle'] != None and H12['circle'] != None:
            #Filter down the feature points given by the Shi-Tomasi to the area of the plate
            plateRegion = [ (A1['circle'][0],A1['circle'][1]) , (A12['circle'][0],A12['circle'][1]) , (H12['circle'][0],H12['circle'][1]) , (H1['circle'][0],H1['circle'][1]) ]
            if not fixedCamera:
                featureChangesInPlate = wellsWithin(corners,[A1['circle'][0]-100,A12['circle'][0]+100], [A1['circle'][1]-100,H1['circle'][1]+100])
            #drawCircles(frame, [ [cir[0],cir[1],5] for cir in featureChangesInPlate ],(122,255,0))
            annotatedWells = annotateAllWells(frame, A1, A12, 8, 12, circles, avgDistance, avgDiameter)

            if fixedCamera:
                tipPoint = (width/2-12, height*2/3+3)
            else:
                print " " #tipPoint = findTipPoint()
            #print "Tip point is @:", tipPoint

        try:
            tipPoint
        except NameError:
            tipPoint = None

        if tipPoint != None and annotatedWells != None:
            drawCircle(frame, (tipPoint[0], tipPoint[1], 12), (0,0,255))
            currentWell = whichWell(tipPoint, annotatedWells, avgDiameter)
        else:
            #pushToMQTT('/charles/well',"none")
            pass

        try:
            currentWell
        except NameError:
            currentWell = None

        if currentWell != None and currentWell != previousWell:
            pushToMQTT('/charles/well',currentWell)
            previousWell = currentWell
            cv2.putText(frame,currentWell,(width*2/3,height*2/3), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,0))
        elif currentWell == None:
            previousWell = None
            if tipPoint != None:
                #pushToMQTT('/charles/well',"none")
                cv2.putText(frame,"no idea :p",(width*2/3,height*2/3), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,0))

        elif currentWell != previousWell:
            cv2.putText(frame,previousWell,(width*2/3,height*2/3), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,0))

    #cv2.imshow(windowName, frame)
    cv2.imshow('Raw Frame', rawFrame)
    c = cv2.waitKey(5)
    if c == ord('q'):
        exit(0)
    elif c == ord('s'):
        filename = time.strftime("%a_%d_%b_%Y_%H-%M-%S", time.localtime())    #Set filename to day, date, local time
        cv2.imwrite("%s.jpg" % filename,frame)
```

Appendix C
Rainin EDP3 Communication Specifications

C.1 Packet Structure

Packets sent to the EDP3 pipette can have variable length based on the number of data bytes required by the command. Both commands to, and responses from the pipette follow this structure:

0xCommand 0xDataSize 0xChecksum [0xData$_0$ .. 0xData$_n$]

C.1.1 Example Command

0x01 0x00 0xFE	Gets the pipette firmware version

C.1.2 Example Responses

0x05 0x02 0xE4 0x14 0x00	Sets the speaker power to 20%
0x06 0x01 0xF7 0x01	Makes a high beep

C.2 Commands

When the response data field is none, it means the pipette just acknowledges the command with a standard packet structure but with no data within it. For example, sending 0x03 0x01 0xF7 0x04 returns 0x03 0x00 0xFC. In other words, just an acknowledgement that the pipette received a request of command type 0x03 (simulate key press).

Command	Data Bytes	Description	Response	Example Command
0x01	none	get firmware version	0xVV 0xVV 0x01 0x02 0xE7 0x15 0x00 Version: 1.5	0x01 0x00 0xFE
0x02	none	get serial number	16 bytes of serial number	0x02 0x00 0xFD
0x03	0xKK 00 : None 01 : Up 02 : Down 03 : Mode 04 : Reset 05 : Trigger 1 06 : Trigger 2	simulate key press (KK)	none	0x03 0x01 0xF7 0x04 Simulate Reset Keypress
0x04	none	get speaker power 00 : OFF >: percentage in Hex	0xPP 0xPP 0x04 0x02 0xE5 0x14 0x00 Power: 20%	0x04 0x00 0xFB
0x05	0xPP 0xPP	set speaker power (PP)	none 0x05 0x00 0xFA	0x05 0x02 0xE4 0x14 0x00
0x06	0xBB 0xBB 01: High Beep 02: Medium Beep 03: Low Beep 04: Tic 05: Warble 06: Double Beep 07: Triple Beep	make speaker sound	none 0x06 0x00 0xF9	0x06 0x01 0xF7 0x01
0x07		get LCD segments	9 bytes for LCD segments	
0x08		set LCD segment ON		
0x09		set LCD segment OFF		
0x0A	none	get motor speed	0xSP 0x0A 0x01 0xEA 0x0A Speed: 10	0x0A 0x00 0xF5
0x0B		set motor speed		
0x0C	none	get motor state Motor Move (MM) 00: Initialize 01: Home 02: Blowout 03: Overshoot 04: Pickup 05: Dispense	0xMM 0xRR MM: Last Motor Move RR: Reserved 0x0C 0x02 0xEB 0x05 0x01 Last Move: Dispense	0x0C 0x00 0xF3
0x0D	none	get motor position	0xMV 0xMO 0xVO 0xVO 0x0D 0x02 0x82 0x6B 0x03	0x0D 0x00 0xF2
0x0E		run motor		

C.3 Checksum Calculation

$$255 - \text{sum}(\text{data}_0 \text{ .. } \text{data}_n)$$

C.3.1 Example

Packet: 0x06 0x01 0xCC 0x01

Checksum: 255 - (6 + 1 + 1) = 247 = 0xF7

Full Packet: 0x06 0x01 0xF7 0x01

Appendix D

pyEDP3 Serial Port Example

```

import pyEDP3
import serial

ser = serial.Serial('/dev/tty.usbserial-A800HATZ',9600)
ex = pyEDP3.Packet(command="makeSpeakerSound", soundType="
  warble")

ser.write(ex.packet)
```

D.1 Command Names

Command Name	Command Argument	Command Byte
getFirmwareVersion	None	0x01
getSerialNumber	None	0x02
simulateKeyPress	keyCode	0x03
getSpeakerPower	None	0x04
setSpeakerPower	speakerPower1 speakerPower2	0x05
makeSpeakerSound	soundType	0x06
getLCDSegments	None	0x07
setLCDSegmentOn	lcdSegment	0x08
setLCDSegmentOff	lcdSegment	0x09
getMotorSpeed	None	0x0A
setMotorSpeed	motorSpeed	0x0B
getMotorState	None	0x0C
getMotorPosition	None	0x0D
runMotor	None	0x0E

Appendix E
I^2C Firmware for Communication with STLM75 Temperature Sensor

```
#include <SoftwareSerial.h>
#include <TinyWireM.h>
#include <USI_TWI_Master.h>

#define STLM75_ADDR 0x48
#define BAUDRATE 9600
#define DUTY_CYCLE_ON 25
#define DUTY_CYCLE_OFF 950

SoftwareSerial tinySerial(8, 1);

void setup() {
  TinyWireM.begin();
  tinySerial.begin(BAUDRATE);
  pinMode(3, OUTPUT);
  delay(3000);
}

void poll() {
  byte msb;
  byte lsb;
  byte flip = 255;
  float temp = 0;

  delay(DUTY_CYCLE_OFF);
  TinyWireM.beginTransmission(STLM75_ADDR);
  TinyWireM.requestFrom(STLM75_ADDR, 2);
  msb = TinyWireM.read();
  lsb = TinyWireM.read();

  if (msb > 127) {
    temp = (-1)*(msb^flip);
  }
  else {
    temp = msb;
  }
  if (lsb == 128) {
      temp += 0.5;
  }

  digitalWrite(3, LOW);
  delay(DUTY_CYCLE_ON);
  tinySerial.print("R= ");
  tinySerial.print(temp, 1);
  tinySerial.print("\n");
  delay(DUTY_CYCLE_ON);
  digitalWrite(3,HIGH);
}


void loop() {
  poll();
```

```
}
```

Bibliography

[1] 1-wire Protocol. http://www.maximintegrated.com/en/products/1-wire/flash/overview/. Accessed: 2014-08-01.

[2] 3D Model for EDP3 Pipette Back Cover. http://www.thingiverse.com/thing:448602. Accessed: 2014-08-01.

[3] ACTGene SpotLiter 96well Plate Light Tracker. http://www.actgene.com/SpotLiter.html. Accessed: 2014-08-01.

[4] Agilent Bravo Liquid Handling Platform. http://www.chem.agilent.com/en-US/products-services/Instruments-Systems/Automation-Solutions/Bravo-Automated-Liquid-Handling-Platform/Pages/default.aspx. Accessed: 2014-08-01.

[5] Apple Application Store. http://store.apple.com. Accessed: 2014-08-01.

[6] Arduino Open Source Electronics Platform. http://www.arduino.cc/. Accessed: 2014-08-01.

[7] Atmel ATTiny85 8-bit Microcontroller. http://www.atmel.com/devices/attiny85.aspx. Accessed: 2014-08-01.

[8] BioBright. http://www.biobright.org/. Accessed: 2014-08-01.

[9] CIDAR Laboratory at Boston University. http://cidarlab.org/. Accessed: 2014-08-01.

[10] Closed Loop Pipette GitHub Repository. https://github.com/charlesfracchia/closed-loop-pipette. Accessed: 2014-08-01.

[11] Coursera. https://www.coursera.org/. Accessed: 2014-08-01.

[12] d3 Timeline. https://github.com/jiahuang/d3-timeline. Accessed: 2014-08-01.

[13] DARPA Living Foundries program. http://www.darpa.mil/Our_Work/BTO/Programs/Living_Foundries.aspx. Accessed: 2014-08-01.

[14] Digi XBee Modules. http://www.digi.com/xbee/. Accessed: 2014-08-01.

[15] DigiMesh Network White Paper. http://www.digi.com/pdf/wp_zigbeevsdigimesh.pdf. Accessed: 2014-08-01.

[16] EdX. https://www.edx.org/. Accessed: 2014-08-01.

[17] Embi Tec LI-2100 LightOne Pro. http://embitec.com/li2100-lightone-pro-384-and-96-well.html. Accessed: 2014-08-01.

[18] Eppendorf Research Pro Electronic Pipette. http://www.eppendorf.com/int/index.php?pb=cd1a8795ac6b09e3&action=products&contentid=1&catalognode=9649. Accessed: 2014-08-01.

[19] Eppendorf-sized Wireless Temperature Sensor for Biological Experiments. https://github.com/charlesfracchia/eppenTemp. Accessed: 2014-08-01.

[20] Evernote. https://evernote.com/. Accessed: 2014-08-01.

[21] Gen9. https://www.gen9bio.com/. Accessed: 2014-08-01.

[22] Gilson Electronic Pipettes. http://www.gilson.com/en/Pipette/Categories/48/. Accessed: 2014-08-01.

[23] Gilson TRACKMAN. http://www.gilson.com/en/Pipette/Products/45.265/Default.aspx. Accessed: 2014-08-01.

[24] Ginkgo BioWorks. http://ginkgobioworks.com/. Accessed: 2014-08-01.

[25] Google Play Store. https://play.google.com/store?hl=en. Accessed: 2014-08-01.

[26] Integra ViaFlow Electronic Pipettes. http://www.integra-biosciences.com/sites/us/electronic_pipettes.html. Accessed: 2014-08-01.

[27] Integrated DNA Technologies. https://www.idtdna.com/. Accessed: 2014-08-01.

[28] International Genetically Engineered Machines Competition. http://igem.org/Main_Page. Accessed: 2014-08-01.

[29] JavaScript Object Notation. http://json.org/. Accessed: 2014-08-01.

[30] Joi Ito TED Talk: Want to innovate? Become a 'now-ist'. http://www.ted.com/talks/joi_ito_want_to_innovate_become_a_now_ist. Accessed: 2014-08-01.

[31] Labster 3d Virtual Laboratory for Education. http://www.labster.com/. Accessed: 2014-08-01.

[32] Maxim Integrated DS18B20 Digital Temperature Sensor. http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf. Accessed: 2014-08-01.

[33] MediaWiki. https://www.mediawiki.org/wiki/MediaWiki. Accessed: 2014-08-01.

[34] MIT Venture Mentoring Service. http://vms.mit.edu/. Accessed: 2014-08-01.

[35] Mongodb. http://www.mongodb.org/. Accessed: 2014-08-01.

[36] Mongodb package on node package manager. https://www.npmjs.org/package/mongodb. Accessed: 2014-08-01.

[37] MQ Telemetry Transport. http://mqtt.org/. Accessed: 2014-08-01.

[38] Nature Protocol Exchange. http://www.nature.com/protocolexchange. Accessed: 2014-08-01.

[39] Node package manager mqtt. https://www.npmjs.org/package/mqtt. Accessed: 2014-08-01.

[40] Node.JS. http://nodejs.org/. Accessed: 2014-08-01.

[41] Nordic Semiconductor nRF24L01+ Low Power 2.4 GHz Tranceiver. https://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01P. Accessed: 2014-08-01.

[42] NUPACK: Nucleic Acid Package. http://nupack.org/. Accessed: 2014-08-01.

[43] Open Computer Vision package. http://opencv.org/. Accessed: 2014-08-01.

[44] OpenCV Hough Circles Function Documentation. http://docs.opencv.org/modules/imgproc/doc/feature_detection.html?highlight=houghcircles#houghcircles. Accessed: 2014-08-01.

[45] OpenWetWare. http://en.wikipedia.org/wiki/OpenWetWare. Accessed: 2014-08-01.

[46] pyEDP3 Python Package GitHub Repository. https://github.com/charlesfracchia/pyEDP3. Accessed: 2014-08-01.

[47] Rainin E4 XLS+ Electronic Pipette. http://www.shoprainin.com/Ergonomic+Pipettes/Single+Channel+Electronic+Pipettes/E4+XLS%2B%2C+LTS.html?CatalogCategoryID=huWsEv2A6c4AAAEwGaMHVM6z. Accessed: 2014-08-01.

[48] RFM22 Industrial Scientific Medical Band (ISM) Transceiver Module. https://www.sparkfun.com/datasheets/Wireless/General/RFM22.PDF. Accessed: 2014-08-01.

[49] Roland Modela MDX20 3-axis Milling Machine. http://www.rolanddg.com/product/3d/3d/mdx-20_15/mdx-20_15.html. Accessed: 2014-08-01.

[50] Saleae Logic16 Logic Analyzer. https://www.saleae.com/logic16. Accessed: 2014-08-01.

[51] Sartorius eLINE Electronic Pipettes. http://www.biohit.com/us/liquid-handling/pipettes-electronic/products/9/eline-electronic-pipette. Accessed: 2014-08-01.

[52] ST Microelectronics STLM75 I^2C Temperature Sensor. http://www.st.com/web/catalog/sense_power/FM89/SC294/PF121768?sc=internet/analog/product/121768.jsp. Accessed: 2014-08-01.

[53] Stanford Online Courses. http://online.stanford.edu/courses. Accessed: 2014-08-01.

[54] Tecan Freedom EVO Liquid Handling Platform. http://www.tecan.com/platform/apps/product/index.asp?MenuID=2694&ID=5270&Menu=1&Item=21.1.8. Accessed: 2014-08-01.

[55] ThermoFisher Electronic Pipettes. http://www.thermoscientific.com/content/tfs/en/products/electronic-pipetting-systems.html. Accessed: 2014-08-01.

[56] Udacity. https://www.udacity.com/. Accessed: 2014-08-01.

[57] Reducing our irreproducibility. Nature Editorial, April 2013. http://www.nature.com/news/announcement-reducing-our-irreproducibility 10.1038/496398a.

[58] G[lenn] Anderson and E[nzo] A. Palombo. Microbial contamination of computer keyboards in a university setting. *American Journal of Infection Control*, 2009. 10.1016/j.ajic.2008.10.032.

[59] A[rnaud] Dechesnea, G[ang] Wangb, G[amze] Gleza, D[ani] Orb, and B[arth] F. Smets. Hydration-controlled bacterial motility and dispersal on surfaces. *PNAS*, 107(32):1436914372, 2010. 10.1073/pnas.1008392107.

[60] L[isa] Chong B[arbara] R. Jasny, G[ilbert] Chin and S[acha] Vignieri. Again, and again, and again... Science Special Issue, December 2011. http://www.nature.com/news/announcement-reducing-our-irreproducibility 10.1126/science.334.6060.1225.

[61] C. G[lenn] Begley and L[ee] M. Ellis. Drug development: Raise standards for preclinical cancer research. *Nature*, pages 531–533, 2012.

[62] S[teven] A. Benner and A. M[ichael] Sismour. Synthetic Biology. *Nature Reviews Genetics*, 2005. 10.1038/nrg1637.

[63] US Center for Disease Control. *Biosafety in Microbiological and Biomedical Laboratories.* US Department of Health and Human Services, Atlanta, Georgia, United States, fifth edition, December 2009.

[64] D[avid] E. Clapham and C[hristopher] Miller. A thermodynamic framework for understanding temperature sensing by transient receptor potential (TRP) channels. *PNAS*, 14(6):80–83, 2010. 10.1073/pnas.1117485108.

[65] J[oseph] DiStefano. *Dynamic Systems Biology Modeling and Simulation*, section 2.7, pages 20–22. Academic Press, Waltham, Massachusetts, first edition, 4 July 2014.

[66] M[ichael] B. Elowitz and S[tanislas] Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403:335–338, January 2000. 10.1038/35002125.

[67] T[homas] Schlange F[lorian] Prinz and K[husru] Asadullah. Believe it or not: how much can we rely on published data on potential drug targets? *Nature Reviews Drug Discovery*, 2011.

[68] C[harles] V. Fracchia and T[om] Ellis. Cyborg Reporter in Synthetic Biology. Undergraduate thesis, Imperial College London, Department of Biology, October 2011.

[69] Haruo Saruyama and Masatoshi Tanida. Effect of chilling on activated oxygen-scavenging enzymes in low temperature-sensitive and -tolerant cultivars of rice (Oryza sativa L.). *Plant Science*, 109(2):105–113, August 1995.

[70] D[avid] Hutchins. *Just in Time.* Gower Publishing, Ltd., Aldershot, England, Second edition, 1999.

[71] J[ohn] Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, June 1986.

[72] S[tephane] Leduc. *La biologie synthetique, etude de biophysique.* A. Poinat, Paris, France, 1912.

[73] Leland H. Hartwell. Macromolecule Synthesis in Temperature-sensitive Mutants of Yeast. *Journal of Bacteriology*, 93(5):1662–1670, 1967.

[74] A[nselm] Levskaya, A[aron] A. Chevalier, J[effrey] J. Tabor, Z[achary] B. Simpson, L[aura] A. Lavery, M[atthew] Levy, E[ric] A. Davidson, A[lexander] Scouras, A[ndrew] D. Ellington, E[dward] M. Marcotte, and C[hristopher] A. Voigt. Synthetic biology: Engineering escherichia coli to see light. *Nature*, 2005. 10.1038/nature04405.

[75] L[isa] Guernsey. M.I.T. Media Lab at 15: Big Ideas, Big Money. *New York Times*, 2000. `http://www.nytimes.com/2000/11/09/technology/mit-media-lab-at-15-big-ideas-big-money.html` Accessed: 2014-08-01.

[76] H[arley] H. McAdams and A[dam] Arkin. Simulation of prokaryotic genetic circuits. *Annual Review of Biophysics and Biomolecular Structure*, 27:199–224, 1998. 10.1146/annurev.biophys.27.1.199.

[77] M[onica] A. Hughes and M. A[lison] Dunn. The Molecular Biology of Plant Acclimation to Low Temperature. *Journal of Experimental Botany*, 47(3):679–698, November 1995.

[78] J[effrey] M. Perkel. Coding your way out of a problem. *Nature Methods*, 2011. 10.1038/nmeth.1631.

[79] C[arol] A. Rohl, C[harlie] E. M. Strauss, K[ira] M. S. Misura, and D[avid] Baker. Protein Structure Prediction Using Rosetta. *Methods in enzymology*, 383:66–93, 2004. 10.1016/s0076-6879(04)83004-0.

[80] A[lessandro] Rossi Fanelli, Renato Cavaliere, Bruno Mondov, and Guido Moricca. *Selective Heat Sensitivity of Cancer Cells*, volume 59 of *Recent Results in Cancer Research*. Springer, Berlin Heidelberg, first edition, 1977. 10.1007/978-3-642-81080-0.

[81] E[dward] Rosten, R[eid] Porter, and T[om] Drummond. Faster and better: A machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):105 – 119, 2010.

[82] M[aureen] Schultz, J[anet] Gill, S[abiha] Zubairi, R[uth] Huber, and F[red] Gordin. Bacterial Contamination of Computer Keyboards in a Teaching Hospital. *Infection Control and Hospital Epidemiology*, 2003. 10.1086/502200.

[83] S[tefan] Tilkov and S[teve] Vinoski. Node.js: Using Javascript to Build High-Performance Network Programs. *IEEE Internet Computing*, 14(6):80–83, 2010. 10.1109/MIC.2010.145.

[84] M[ark] Turner, D[avid] Budgen, and P[earl] Brereton. Turning Software into a Service. *IEEE Computer Society*, 2003. 10.1109/MC.2003.1236470.

[85] H. K. Yuen, J. Princen, J. Illingworth, and J. Kittler. A comparative study of hough transform methods for circle finding. In *Proc. 5th Alvey Vision Conf., Reading (31 Aug*, pages 169–174, 1989.

[86] D[ina] Zielinski, A[ssaf] Gordon, B[enjamin] L Zaks, and Y[aniv] Erlich. iPipet: sample handling using a tablet. *Nature Methods*, 11:784785, July 2014.