Final Report

Version 1.4

Document #100700

March 31st, 2025

WATER BADDIES - Microplastic, Heavy Metal and Inorganics Water Detection System for Environmental and Human Health

Team 25040

Brendan Bamberg

Jameson Brehmer

Daniel Knaus

Austin Medina

Alia Nichols

Aidan Talucci

Tyler Thursby

Revision History

1/23/2025 Rev(–) Created the initial draft of the SDD
1/27/2025 Rev(1.1) Added Verification Procedures
1/28/2025 Rev(1.11) Changes System requirements and verification methods to reflect design changes. No ECR was needed because the requirements did not change the form or function of the system
3/2/2025 Rev(1.2) Modified the SDD to update SystemWrapper and CSC's, updated: wire list, wire diagram, Water Baddies Control GUI, and system block diagram, added SR028 datasheet
3/31/2025 Rev(1.3) Updated SDD to include new diagrams and descriptions of the system
4/8/2025 Updated Electrical Requirements to require both test and analysis. Updated System Block Diagram removing Arsenic Module. Updated SDD (Look at SDD Rev. history for more details).

5/3/2025 Rev(1.4) Updated colorimetric detection section to include reflection mitigation procedure and data.

5/4/2025 Rev(1.5) Updated microplastic detection description to include plastic fluorescent spectrums and system design. Updated Verification process for MVP2 and SR001.

Table of Contents

1.0 Introduction & Project Description

1.1. Identification

This Technical Data Package describes the architecture, interfaces, and functions of the Water Baddies Software System. The system includes the Water Baddies App, the control GUI, the camera subsystem, and the mechanical functionality subsystem.

1.2. System Overview

The Water Baddies System shall be a lightweight, cheap, and portable system aimed at detecting microplastics, metals, and inorganics (collectively referred to as ***baddies)*** in water samples. This project is a continuation of the previous year's project where the team struggled with the size, ease of use, and detection of microplastics smaller than a certain size. The Water Baddies System aims to further the project by distinguishing between dust and microplastics at the micrometer level, detecting metals: lead and cadmium detecting inorganics: nitrate/nitrite and phosphates, and developing an app to provide a streamless delivery of information to the end user. These baddies, when ingested, have been found to cause kidney and liver problems, along with other major health problems. Once ingested, the baddies are also incredibly resilient, being incredibly difficult to remove from the body, causing long-term effects to those who didn't realize they were ingesting them in the first place. The system will allow users the ability to detect the baddies and some of their quantities, although ingesting the baddies in any quantity is potentially harmful to the user. While all of these materials are detectable in some form, a system doesn't exist to identify all 3 types of baddies and microplastics on a small level. The Water Baddies System also aims to display the results in a modern way through the use of a mobile app and long-term storage of the data in a database in order to track water quality over time and location.

1.3. System User Description

The Water Baddies System shall be an easy-to-use, accessible, and portable device. A new user shall be able to learn how to operate the system in 10 minutes. To begin, the user will turn on the device, open the Water Baddies app on their handheld device, and connect to the Water Baddies System. Next, they will open the door on either side and collect the empty syringe from the mount. They will then fill the syringe with the water sample they wish to test and replace the, now full, syringes on either side. They must turn on the microscope using the circular button on the microscope and confirm the blue light on the top of the microscope turns on. Now, each door can be closed and the user should hear a "click" when they have successfully closed the system. The user will press go using the button pertaining to the test they would like to run. Button A will run the microplastics detection side independently, button B will run the heavy metals and inorganics side independently, and button X will run both these tests simultaneously. The display on the side of the device will indicate the status of the test that they are running. The user can use the display to tell them when their test is complete, and check the app for the final results. Once the testing is complete, the user can safely dispose of the cartridges by opening the discard drawers on the side of the device and following their local hazardous waste disposal guidelines.

1.4. Document Overview

The Water Baddies Technical Data Package will cover the in-depth architecture, design solutions, and functions of the software behind the Water Baddies System. The document will first discuss our defined MVP as well as an overall system description including an architectural overview through the system block diagram, this will go with the following system verification plans that will present clear criteria for confirmation of success. Additionally, design documentation starting with our IDL including system requirements documentation, verification documentation, hardware drawings and models, and software documentation will all be presented to create a clear outline and understanding of the purpose and design of the system. It will be made clear the design considerations made for individual subsystems or functionalities and why those were chosen, the desired stimuli and responses from each of the subsystems or design considerations, the computer software components in the system, the requirements traceability, and any additional notes relating to the software note discussed earlier in the document.

2.0 Minimum Viable Product (MVP) 1 & 2

2.1. MVP 1

2.1.1. MVP 1 Description

We have identified the ability to colorimetrically detect concentrations of heavy metals as our Minimum Viable Product 1 (MVP1). We use pre-manufactured test pads that will be removed from existing strips for accuracy and replicability. These test pads will then be added to our in house paper fluidic.

2.1.2. MVP 1 Procedure

The new MVP1 procedure is to first remove our selected test pad from the premade test strip using scissors. We then drop a 10 µl drop of our water with our selected concentration of the appropriate contaminants. After at least 30 seconds an image is taken of the test pad and ran through ImageJ to receive the luminance value. After each drop, luminance is determined three times and averaged out at every concentration. This is repeated through every predetermined concentration. Each heavy metal concentration gradient will then be analyzed to form a curve which relates RGB to the concentration of that heavy metal using luminance. By utilizing a simple 3D printed tube to remove external light we are able to avoid light pollution which will impact the results. The same procedure will be performed with different combinations of the heavy metals with different concentrations. Following the establishment of this curve, an unknown concentration of a mixture of heavy metals will be created and analyzed using our curve to determine accuracy.

2.1.3. MVP 1 Verification

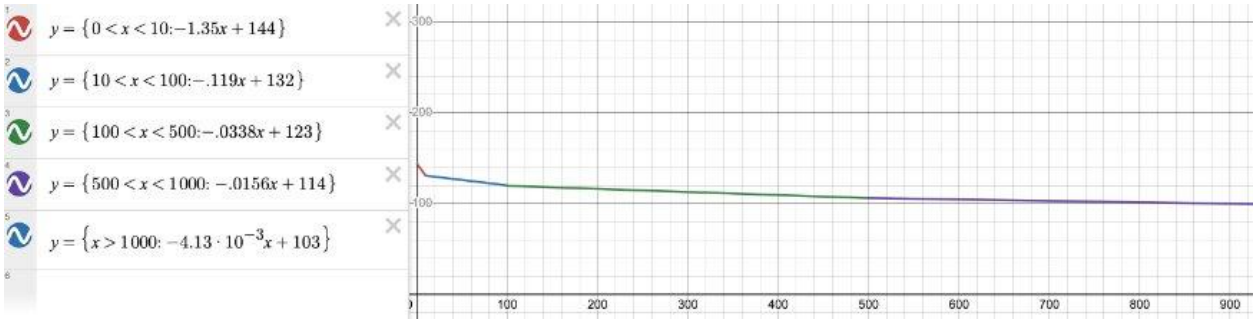
ID	System Requirement	
	Performance	Verification
SR001	The Water Baddies system shall detect the concentration of heavy metals, and inorganics in a water sample with 85% accuracy.	T
SR005	The Water Baddies system shall use a wired camera to perform colorimetry with at least a resolution of 1280 x 720.	I
SR014	The Water Baddies system shall utilize a 5x5 cm double-layer microfluidic paper-based device. [1]	I

2.1.4. MVP 1 Results

Below are the curves generated for Cadmium, Lead, Nitrate, Nitrite, and Phosphate. These were all generated using the same procedure as stated above.

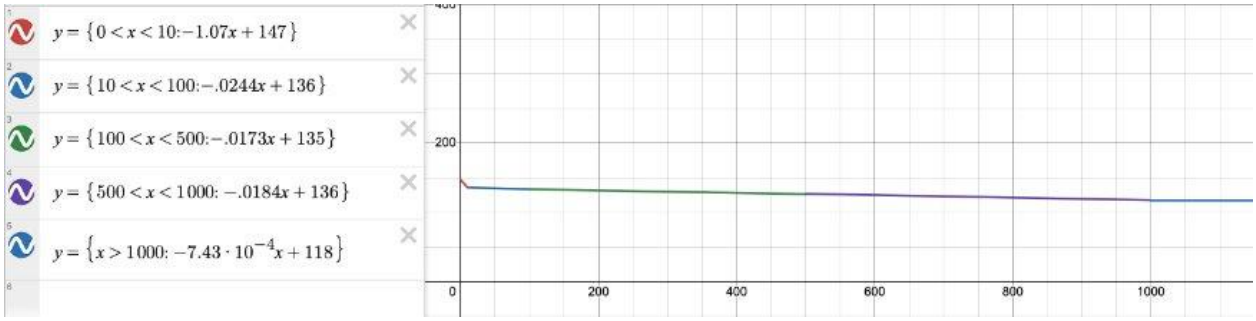
Cadmium Curve and Data

Cadmium Concentration in Nanomolar	ImageJ Value	Slope
0	143.9	-1.345
10	130.45	-0.1194444444
100	119.7	-0.03375
500	106.2	-0.0156
1000	98.4	-0.004125



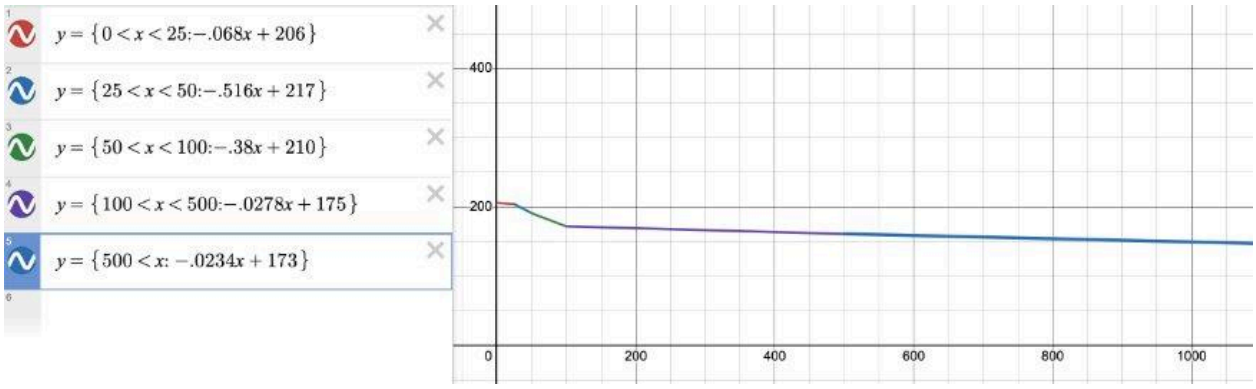
Lead Curve and Data

Lead Concentration	ImageJ Value	Slope
0	146.6	-1.07
10	135.9	-0.02444444444
100	133.7	-0.01725
500	126.8	-0.0184
1000	117.6	-0.00074285714



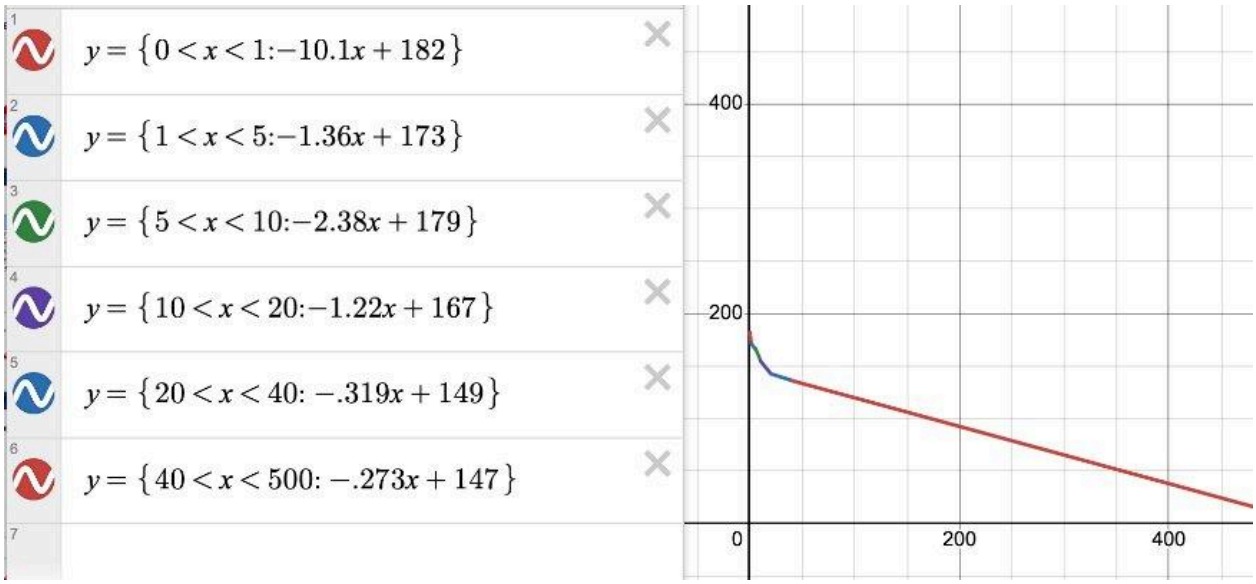
Nitrate Curve and Data

Nitrate Concentration	ImageJ Value	Slope
0	205.7	-0.068
25	204	-0.516
50	191.1	-0.38
100	172.1	-0.02775
500	161	-0.0234



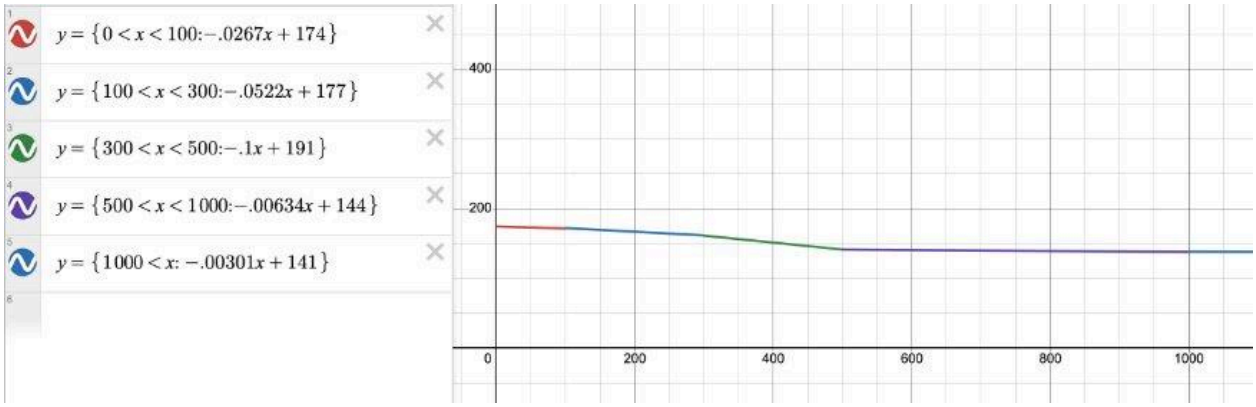
Nitrite Curve and Data

Nitrate Concentration	ImageJ Value	Slope
0	205.7	-0.068
25	204	-0.516
50	191.1	-0.38
100	172.1	-0.02775
500	161	-0.0234



Phosphate Curve and Data

Phosphate Concentration	ImageJ Value	Slope
0	174.09	-0.0267
100	171.42	-0.0522
300	160.98	-0.10015
500	140.95	-0.00634
1000	137.78	-0.00301333333



Following the successful implementation of our reflection rectification we were able to meet the accuracy standards we were hoping for. This allows us to pass MVP1 and move on to implementing these pre-made test pads into our own paper fluidic device. Specific data collection tables are present in the verification section, while average percent error and worst percent error are shown below in the data sheet for clear communication and easy to read formatting.

SR001			
Referenced VTP Paragraph Number:			
Analysis Referenced (for verification by T/A): Test			
Name of Test: Colorimetric Concentration Accuracy			
Unit Under Test: Paper Fluidic			
Name: Paper Fluidic Part Number:100100 Serial Number:			
Results (Pass / Fail): Pass		Date of Test: 4/2	
Recording of Test Measurement: Cadmium: Avg. Error: 7.13% Worst: 12.76% Lead: Avg. Error: 8.88% Worst: 13.67% Nitrate: Avg. Error: 2.03% Worst: 5.41% Nitrite: Avg. Error: 5.76% Worst: 9.28% Phosphate: Avg. Error: 5.5% Worst: 9%	Requirement (SR, with tolerances): Concentration accuracy must be within a 15% error.	Test Equipment Error:	Adjusted Test Limit:
Computations, (Include Analyses Results, if any): (55-62.52)/62.52 = -.1367			
Signatures: Tester____ Jameson Brehmer_____ Customer_____			

2.2. MVP 2

2.2.1. MVP 2 Description

The MVP2 is described as the successful detection of microplastic concentration in a water sample. The goal is to assess the initial performance of the optical assembly and code base. Testing will consist of the creation of a Python script for image analysis of fluorescence microscopy and water samples with known microplastic concentrations. These known concentrations will be created, placed on the optical assembly, imaged, and processed by the Python script.

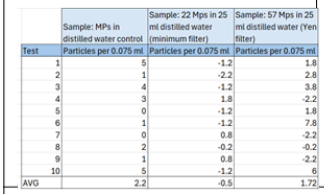
2.2.2. MVP 2 Procedure

The microplastics will be created using sandpaper from a cup made of entirely PET plastic. We chose to test using PET plastic because it is most commonly produced for household products likely to leach into drinking water, thus replicating a similar outcome expected with the use of the device. These microplastics will be counted out by the algorithm while they are dry, and then placed into distilled water. It is important to use distilled water to minimize potential microplastics already present in the water. However, throughout testing we discovered the difficulty in eliminating the presence of microplastics in the air and distilled water since this water is stored in plastic containers. Once the samples are created using the pre counted microplastics and the distilled water, they will be placed into the optical subassembly and the Python script will run its analysis. Once complete, the number

identified by the software will be compared to the known number of microplastics placed into the sample and an error analysis will be performed to determine the effectiveness of our image analysis.

2.2.3. MVP 2 Verification

The goal of MVP2 is to verify the accuracy of the quantification of microplastics present in the test sample. Initially the microplastics will be created, classified, and counted by hand. The known quantity sample will be placed in the optical subassembly and images will be captured. The images will be inputted into the Python script and the output quantification analysis will be compared to the known quantification completed by hand.

<table>
<tr><td colspan="4">Water Baddies Data Sheets</td></tr>
<tr><td colspan="4">SR001</td></tr>
<tr><td colspan="4">Referenced VTP Paragraph Number:</td></tr>
<tr><td colspan="4">Analysis Referenced (for verification by T/A): Test</td></tr>
<tr><td colspan="4">Name of Test: Concentration of microplastics in a water sample with 85% accuracy.</td></tr>
<tr><td colspan="4">Unit Under Test: Optical Subassembly</td></tr>
<tr><td colspan="4">Name: Microscope Fluorescence Assembly
Part Number: N/A
Serial Number: N/A</td></tr>
<tr><td colspan="2">Results (Pass / Fail):
Fail</td><td colspan="2">Date of Test: 2/24/2025</td></tr>
<tr><td>Recording of Test Measurement:
  </td><td>Requirement (SR, with tolerances):
85%</td><td>Test Equipment Error:
Contaminated distilled water</td><td>Adjusted Test Limit: 85% accuracy</td></tr>
<tr><td colspan="4">Computations, (Include Analyses Results, if any):

1.72particles/ 0.075ml = 22.93 particles/mL
From software: 33.56 particles/mL</td></tr>
<tr><td colspan="4">Signatures:
Tester: Alia Nichols
Brendan Bamberg
Customer_____</td></tr>
</table>

Test	Sample: MPs in distilled water control Particles per 0.075 ml	Sample: 22 Mps in 25 ml distilled water (minimum filter) Particles per 0.075 ml	Sample: 57 Mps in 25 ml distilled water (Yen filter) Particles per 0.075 ml
1	5	-1.2	1.8
2	1	-2.2	2.8
3	4	-1.2	3.8
4	3	1.8	-2.2
5	0	-1.2	1.8
6	1	-1.2	7.8
7	0	0.8	-2.2
8	2	-0.2	-0.2
9	1	0.8	-2.2
10	5	-1.2	6
AVG	2.2	-0.5	1.72

Fig 2.2.3.1 Datasheet of initial MVP 2 testing

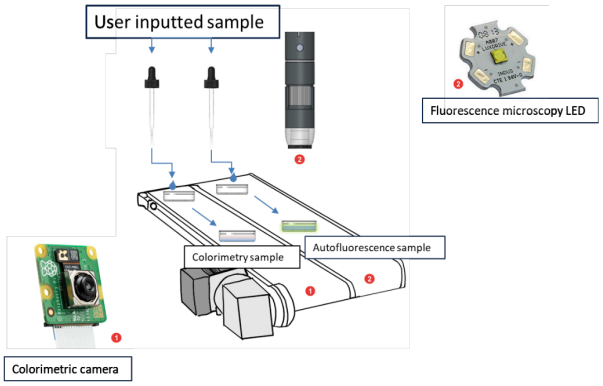
The datasheet above shows our failed initial testing of MVP 2 with our goal of getting a 2.28 particles per milliliter concentration, though there were problems with illumination and reflections in the water that contributed to the failed test. To rectify these errors, we offset the LED and retested to find a conversion factor consistent in our testing. The conversion factor is used to account for experimental errors such as miscellaneous particles in the water sample that happen to fluoresce similar to microplastics and uneven distribution of microplastics in a water sample that would otherwise result in a skewed concentration value.

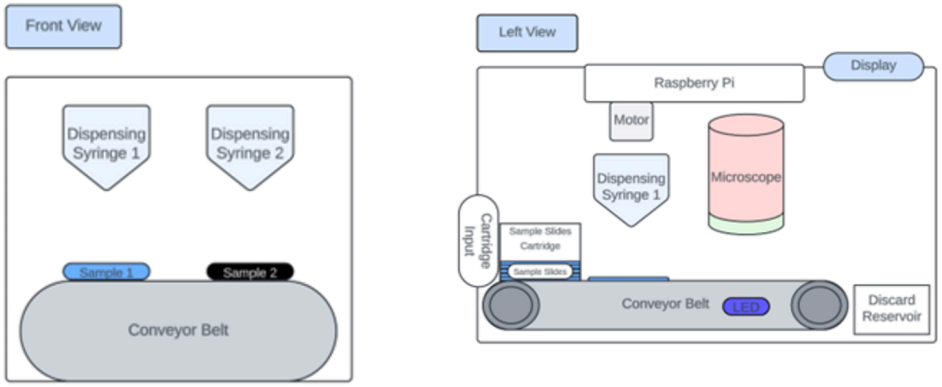
3.0 System Description & System Block Diagram

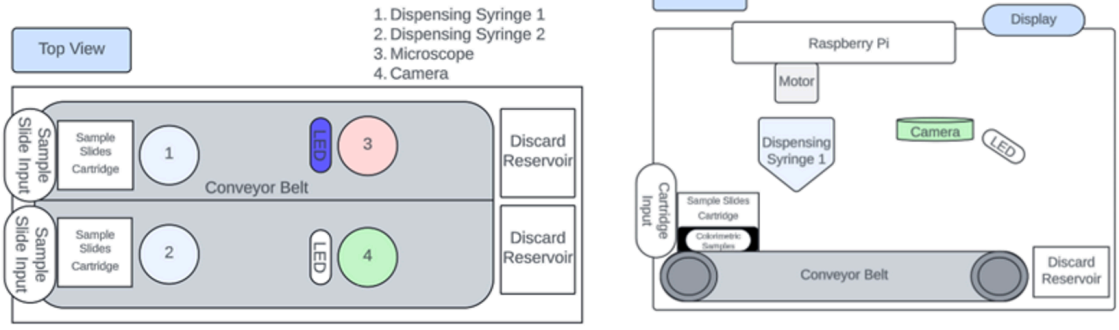
3.1. Description of System and Subsystem (Including Pictures)

3.1.1. System Description

The system can be described and divided into five subassemblies illustrated in the system block diagram. There is the electrical aspect depicted in the system block diagram in green, the optical subassembly in pink, the mechanical parts in orange, the microfluidics in red, and the software in blue. Our system works by having water poured in the top of the device. From there the water flows into syringes that will allow it to be either dropped onto the optical slide or the paper fluidic. These two test cartridges will move parallel to each other on corresponding conveyor belts. On the paper fluidic side of this belt, additional chemicals will be dropped onto the device in order to have the full chemical reaction. After that the chemicals will be allowed to incubate before being photographed by a camera that will upload the photo to our code and allow for colorimetric analysis to occur giving us a quantity necessary for comparison to the EPA standards. Meanwhile, the simultaneous microscope test will light the water sample and through a highly specific filter be able to excite the microplastic particles and generate a concentration. Information from both tests can then be sent through Bluetooth to our mobile app.



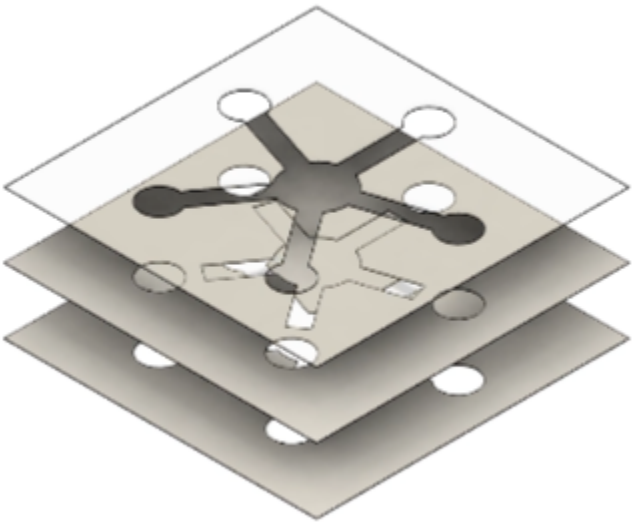




3.1.2. Paperfluidic Subsystem Description

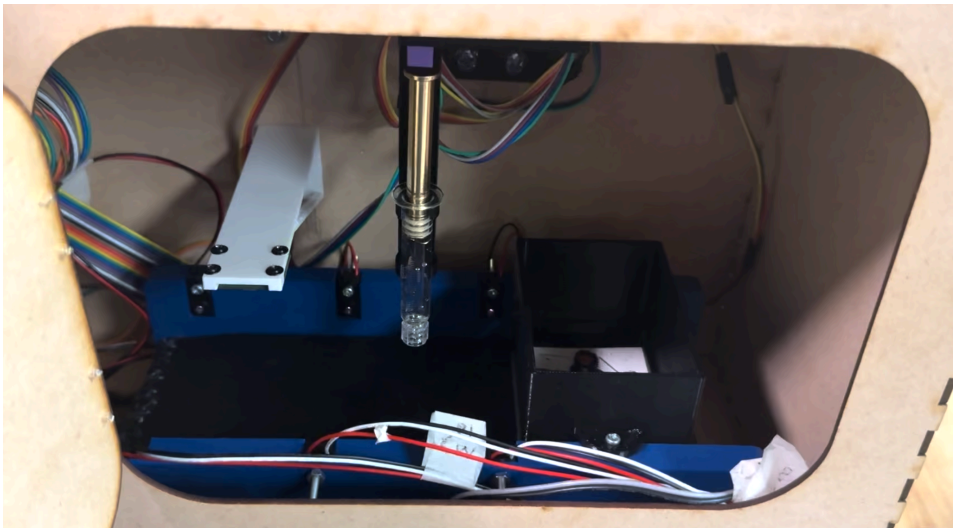
The paperfluidic subsystem consists of the paperfluidic device along with the associated chemicals to elicit a colorimetric reaction. Colorimetry is a method of qualitative, quantitative, or semi-quantitative analysis of a specific color-specific reaction. For our purpose, we have utilized pre made and pre tested chemicals pads due to their documented success and increased replicability. These pads will be adhered to the paperfluidic using the same wax that already lines the channels and will not impact the results. This elicited color change will allow for both identification and quantification of our targeted Water Baddies. The paper fluidic device is designed on AutoCAD where the channels will be hydrophilic and the surrounding surface area will be hydrophobic. We accomplish this by using

paraffin wax to act as a barrier surrounding the detection zones and channels. By tracing our desired channels with the max, and then melting said wax using a hot iron we can outline our channels and control the flow. Due to the nature of paper and capillary flow (similar to how paper towels absorb), the flow is passive and does not require external energy (i.e., a motor). This system will allow us to drop a sample of water (see 3.1.3 Mechanical Subsystem Description) onto the middle of the paper fluidic device and the sample will passively flow to areas which will drive colorimetric chemical reactions. These color changes can be quantified by taking a photo and analyzing the color change.



3.1.3. Mechanical Subsystem Description

The mechanical subsystem will consist of the dropper mechanisms which will include the uptake of the customers water as well as the dispersal of said water onto both the paper fluidic as well as the optical slide. Due to the nature of calculating concentrations a precise drop must be accurately and consistently dropped in the desired area. This will be done using linear actuator motors precisely aligned to trigger a release of a drop when initiated by the raspberryPi. The mechanical subsystem also consists of the double, parallel conveyor belt system that will be powered by stepper motors and will be tasked with moving both the optical slides under the necessary droppers before moving it under the microscope for analysis. Additionally, the other belt will move the paper fluidic under its droppers before positioning it underneath the camera for an image to be taken for analysis. Finally, these systems will need to be fully enclosed, while also including the electrical subsystem among other subsystems components. This enclosure will be necessary to make an efficient, working, safe device that can be transported while remaining functional and accurate.



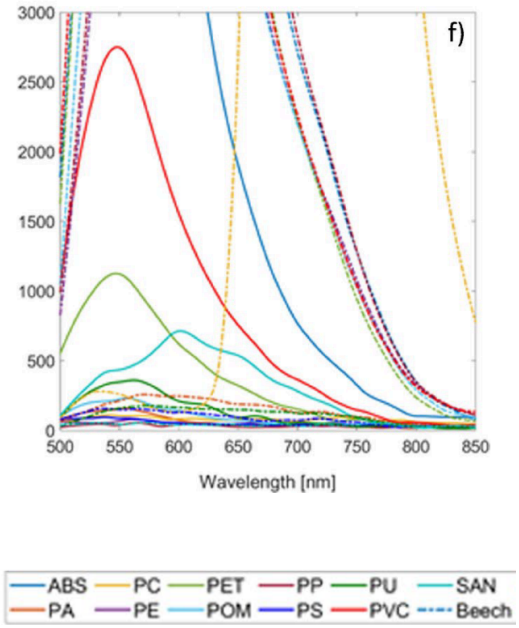
Dropper and Conveyor Subassemblies

Full Frame

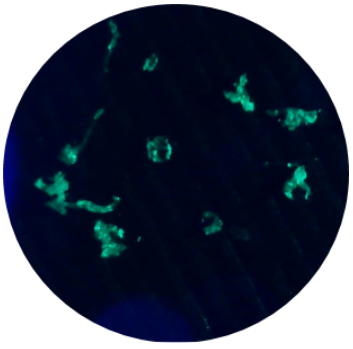
3.1.4. Optical Subsystem Description

The optical subsystem consists first of the optical well slides that will be filled with a 0.075 ml sample of the imputed water by the dropper system. This slide will then be moved by the conveyor belt to be positioned above a blue LED placed underneath the conveyor belt. The conveyor belt

will have a small hole the diameter of the well divot in the glass slide so that light can illuminate the sample from below to eliminate reflection. A microscope equipped with an emission filter is in place above the slide's position and will be set to focus on the slide during system assembly. hen the sample is illuminated by the blue LED, plastic microparticles within the water emit autofluorescent light in the 500–540 nm range. This specific wavelength range passes through the microscope's filter, allowing only the fluorescent signal from plastics to be captured.

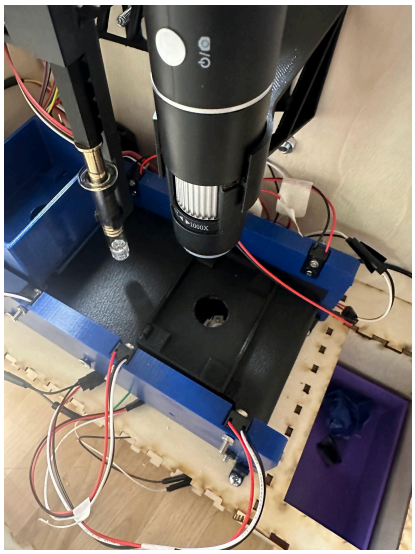


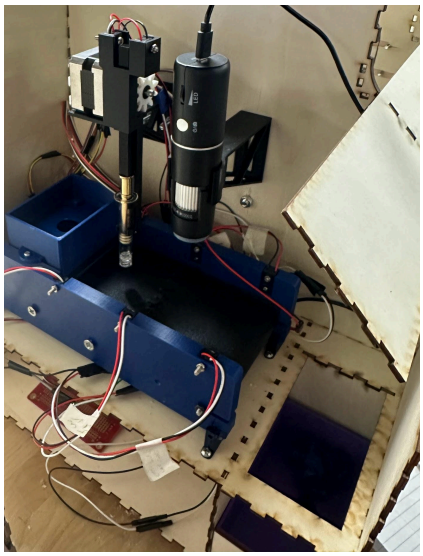
Based on the 11 plastic types tested, Wohlschläger et. al were able to identify their fluorescent range after exciting the particles with a 445nm excitation wavelength. The range for all plastic types encompassed the peak relative intensity within the 500 - 540 nm range that the bandpass filter allows. It is important to accurately filter excited wavelengths so that there is confidence in the systems ability to discern between microplastics and other materials present in water that may fluoresce under 445 nm light. Non-plastic particles like dust emit outside this range and are effectively filtered out.The microscope transmits the image to a Raspberry Pi, which analyzes the number of visible fluorescent particles. Based on this count, the system calculates the concentration of microplastics in the sample.



Fluorescent emission of microplastics

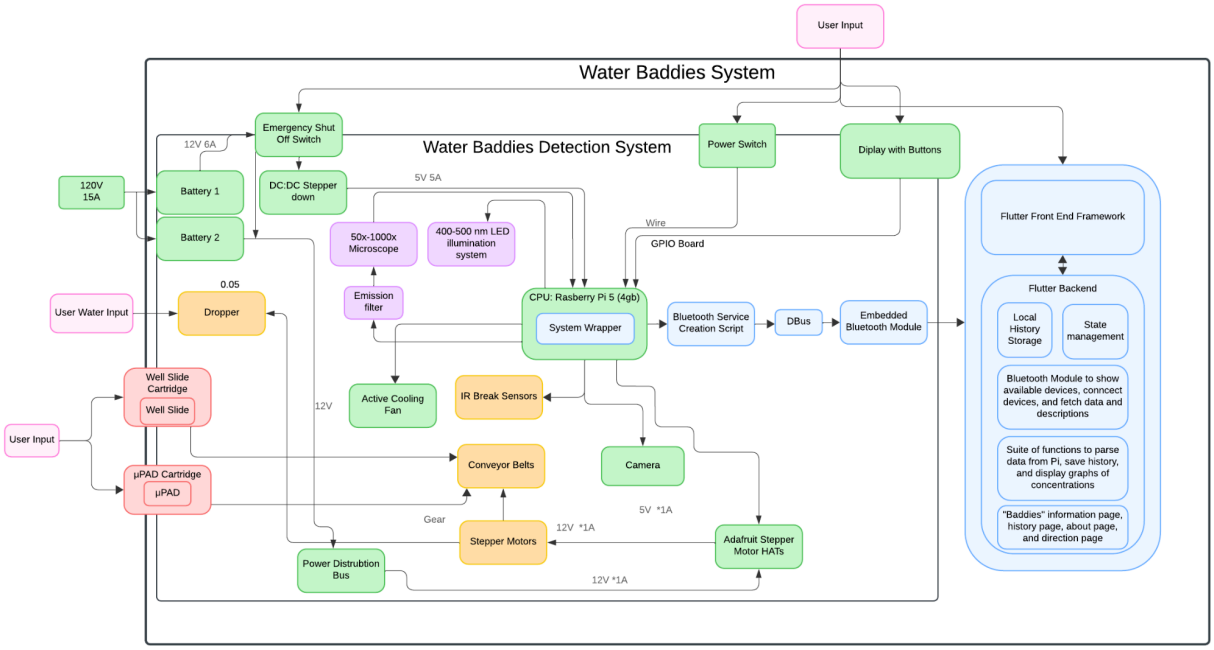
The bandpass filter is fitted to the edge of the microscope in front of the sensor using a snap fit printed part that can be removed for cleaning or replacement. There is a hole in the conveyor belt for bottom illumination which is placed at the slight offset to mitigate reflection errors. The Microscope is mounted on a 3d printed adjustable mount with the on button and focusing mechanism facing outward for recalibration and ease of use since the user will have to turn on the microscope manually before testing. Once the image is collected the cartridge will be discarded in the discard bin on the right of the microscope for safe disposal.





.

System Block Diagram



3.2. Global, Cultural, Social, Environmental, and Economic Context.

Clean water is a finite resource that many of us take for granted. The Water Baddies' initiative is the development of accessible systems designed to inform and educate users about the rapid increase in contaminants in water systems. Not only will the design identify the 'baddies', but it will also serve as a source for understanding the detrimental impacts these pollutants have on those who consume them. The global impact of our design resides in the accessibility to information concerning water systems that directly affect the well-being of individuals and their loved ones. Access to clean water, education, and knowledge are privileges the team believes are human rights and there should not be restrictions to resource access based on social class, location, nor education level. We hope that the development of the Water Baddies detection system will be a step in the process of improving the quality of life around the world.

While we feel that microplastics are all around us, a survey for statista by YouGov expressed that only 52 percent of adults in the US have heard of microplastics. It is estimated that in the US there are as many as 22 million people drinking water that passes through lead pipes. Additionally, the BBC approximates 23 million people worldwide who live on floodplains contaminated by concentrations of toxic waste from metal mining activity. And with climate change and increased flooding, the contamination is projected to spread. Policies are only as good as their enforcement, and if people aren't aware of the contamination, there is no motivation to demand systemic change toward increased access to uncontaminated water. The Water Baddies device aims to 'empower with information' by allowing users to identify contaminants in their personal water supply so that they may take steps to

gain agency over their health. Our device also considers the 43 million Americans who rely on private well water. This is an important consideration because there are no federal laws that govern well water quality, making the confidence in their cleanliness decrease significantly. Using a point of care device to monitor the contamination of private wells allows users to monitor contamination and proactively protect themselves from a range of naturally and human-produced toxins present in groundwater. The portability of the device makes it ideal to be used in areas with limited access and resources to water sanitation. Tackling the monstrous global issue of clean drinking water is only achieved through collaborative efforts and continued awareness. Although our device does not remove contaminants from water samples, it quantifies the issue, empowering users with cognizance to take action and drive change. The Water Baddies team approached this challenge with an ambitious vision, leveraging interdisciplinary expertise to develop a solution that prompts necessary action. We have created a device that not only advances scientific frontiers but also provides a practical and sustainable solution to one of the world's most pressing issues.

3.3. Standards Used

[1] Bluetooth SIG, "GATT Specification Supplement," Version 2025-01-15. [Online]. Available: https://www.bluetooth.com/specifications/gss/. [Accessed January 27, 2025].

[2] International Electrotechnical Commission. (2008). *IEC/EN 62471:2008 - Photobiological safety of lamps and lamp systems*. International Electrotechnical Commission. Available:chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://smartvisionlights.com/wp-content/uploads/IEC_62471_summary.pdf

[3] National Electrical Manufacturers Association. (n.d.). *NEMA 5-15P plug standard*. Retrieved from https://www.nema.org

[4] USB Implementers Forum, *Universal Serial Bus 3.0 Specification*, Revision 1.0, USB-IF, 2008. [Online]. Available: https://www.usb.org

[5] NXP Semiconductors, *I²C-Bus Specification and User Manual*, Rev. 6, NXP Semiconductors, Jan. 2014. [Online]. Available: https://www.nxp.com

[6] IEEE Standards Association, *IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications—Amendment 5: Enhancements for Very High Throughput for Operation in Bands below 6 GHz*, IEEE 802.11ac-2013, 2013. [Online]. Available: https://ieeexplore.ieee.org

[7] "IEEE Standard for Information Technology--Systems Design--Software Design Descriptions," in IEEE STD 1016-2009 , vol., no., pp.1-35, 20 July 2009, doi: 10.1109/IEEESTD.2009.5167255.

[8] W3C. (2018). WCAG 2.1 (Web Content Accessibility Guidelines). World Wide Web Consortium. https://www.w3.org/TR/WCAG21/

[9] International Organization for Standardization. (2020). *ISO/TR 21960:2020 - Plastics — Environmental aspects — State of knowledge and methodologies*. Retrieved from https://www.iso.org/standard/72300.html

[10]Standard Practice for Calculation of Color Tolerances and Color Differences from Instrumentally Measured Color Coordinates. ASTM International, West Conshohocken, PA, 2021.
[11]Standard Guide for the Characterization of Paper-Based Microfluidic Systems. ASTM International, West Conshohocken, PA, 2023.

4.0 System Verification

4.1. System Verification Plan

Detailed Verification Plans and Data Sheets can be found in the appendix under Verification Plans.

The system will be verified using the following methods:
T - Test: Application of procedures in order to determine the functional capabilities of a component or process.

A - Analysis: The use of technical and/or mathematical models or simulations to provide evidence that a component or process meets requirements.

D - Demonstration: Physical performance of a component or process which displays proper adherence to requirements.

I - Inspection: A process of confirming whether a product, system, or process meets specified requirements by visually examining it, using one or more senses (like sight, touch, sometimes smell) to identify any discrepancies or non-compliance with the defined standards, without necessarily needing to actively test its functionality

4.2. System Requirements Verification Matrix

ID	System Requirement	Verification Method			
		T	A	D	I
	Performance				
SR001	The Water Baddies system shall detect the concentration of microplastics, metals, and inorganics in a water sample with 85-90% accuracy.	x			
SR002	The Water Baddies system shall use a microscope with 50-1000x magnification capabilities for fluorescence microscopy.			x	x
SR003	The Water Baddies system shall use blue LEDs peak at 445 nm to fluoresce microplastics.	x			

SR004	The Water Baddies system shall use an emission filter between 500-540 nm to detect fluorescence peaks of microplastics.	x			
SR005	The Water Baddies system shall use a wired camera to perform colorimetry with at least a resolution of 1280 x 720.		x		
SR007	The Water Baddies App shall have a data loss rate of less than 1%		x		
SR014	The Water Baddies system shall utilize a 5x5 cm double-layer microfluidic paper-based device.				x
SR015	The System shall have 4gb of ram to be able to process and analyze data.				x
SR016	The Water Baddies' water dispensing system shall dispense water in increments of 120 µL with a standard deviation of ± 20µL.	x			
SR017	The Water Baddies system shall analyze one paper fluidic and one microplastic sample per cycle.				x
SR018	The Water Baddies system shall have a white spectrum LED.				x
SR024	The battery life of the system shall last at least 1 hour.	x	x		
SR022	The Water Baddies system shall make use of 50ml reloadable water reservoirs to accept test species into the system.				x
	Environment				
SR010	The Water Baddies system will connect to a standard American outlet, using 120V 60Hz. Plug type A and B. Standard (NEMA 5-15P)			x	
SR027	The Water Baddies System shall have an operational temperature between 15-35C.		x		
	Safety				
SR020	In an emergency shutoff, the system shall abort the previous test after being rebooted and give a visible alert from the mobile application.			x	
SR023	The Water Baddies systems shall have chamfered edges and moving components covered/shielded from users.				x
SR028	The Water Baddies System's components shall not exceed 12 volts and 6 Amps.	x	x		
	Design Features				
SR006	The Water Baddies system shall weigh less than 25 lbs.	x			

SR008	The Water Baddies System shall take a new user 10 minutes to learn.			x	
SR009	The Water Baddies GUI shall have an instruction set for using the system.			x	
SR011	The Water Baddies system will generate a report outlining the concentration of microplastics within the sample, the concentrations of lead and cadmium, as well as the concentration of both nitrates and nitrites in parts per ml.			x	
SR012	The Water Baddies App shall connect using Bluetooth 5/Bluetooth Low Energy			x	
SR013	The Water Baddies system report shall display one particle concentration per carousel slide			x	
SR019	The Water Baddies system shall make use of reloadable cartridges containing 5 microfluidic pads and 5 optical slides to load the system with testing.			x	
SR021	The Water Baddies App will give an audible and haptic alert if detection levels of each detectable are above the thresholds established by the EPA in their report on National Primary Drinking Water Regulations.			x	
SR026	The Water Baddies system shall operate fully autonomously once accepting user input of the water reservoir, test material cartridges, and test selection via touchscreen.			x	
SR025	The Water Baddies App shall have a single connection page to aid the user in connecting to the device.			x	
SR029	The Water Baddies App shall download the report as a PDF.			x	

4.3. Verification Table

System Requirement ID	Subsystems						
	Paper Fluidics	Optical Assy.	Mobile App	Electrical Assy.	Conveyor Belt Assy.	Dropper Assy.	Software Assy.

SR001 (T): The Water Baddies system shall detect the concentration of microplastics, metals, and inorganics in a water sample with 85% accuracy.	1. (T) Colorimetric change calculation will generate a concentration that is 85% accurate of a controlled concentration of heavy metals. 2. (T) Colorimetric change calculation will generate a concentration that is 85% accurate to a controlled concentration of inorganics.	(T) Will be able to detect fluorescent microplastics ≥ 5 µm in size using a microscope, a laser, and a bandpass filter. Using a controlled quantity analysis test, accuracy can be calculated.		(T) The subsystem will power the LEDs and Camera for the correct amount of time	Will deliver the water sample to the microscope accurately and reliably	Will drop appropriate amounts of water in each location with precision	(T) Will analyze the images of 'water baddies' and calculate concentrations.
SR002 (D/I): The Water Baddies system shall use a microscope with 50-1000x magnification capabilities for fluorescence microscopy.		(I) Magnification verification will be done by inspection using the microscope specifications sheet					
SR003 (D/A): The Water Baddies system shall use blue LEDs at 445nm to fluorescent microplastics.		(T) Verify the Blue LEDs emit wavelength of 445nm for excitation bndpass source		The Blue and White LEDs will use the Raspberry Pi GPIO pins.			(The Blue LEDs will turn on when the microplastic sample is ready to be analyzed
SR004 (T): The Water Baddies system shall use an emission filter between 480-560 nm to detect fluorescence peaks of microplastics.		(D/A) Filter shall be verified using a spectrometer and a large bandpass source					
SR005 (I): The Water Baddies system shall use a wired camera to perform colorimetry with at least a resolution of 1280 x 720.		(I) Camera shall have a display resolution of 1280 x 720 verified using the camera specifications sheet		(T) The camera will turn on and off at the Raspberry Pis command. The camera and the Pi will communicate with each other.			
SR007 (T): The Water Baddies App shall have a data loss rate of less than 1%.			(D) The App will use Bluetooth Low Energy	(D) The camera and microscope are going to be hard-wired to the Raspberry Pi			(I/T) The software shall verify validity of the data it receives
SR014 (I): The Water Baddies system shall utilize a 5x5 cm double-layer microfluidic paper-based device.	(I) Paper fluidic will be measured upon completion and be under the dimension limit.				(D/I) The conveyor system shall reliably accept and precisely locate 5x5 cm double-layer microfluidic paper-based device via cartridge.		

SR015 (I): The System shall have 4gb of ram to be able to process and analyze data.				(I) The Pi will have 4 GB of ram for Processing			(D) The software must run using a maximum of 4gb of ram
SR016 (T): The Water Baddies' water dispensing system shall dispense water in increments of 70 µL with a standard deviation of ± 20µL.	(T) Channel length will be able to maintain full drop without additional flowing solution while getting equal amount to each reagent well when using a 120 µL drop from a micropipette			(D) The subsystem shall supply enough power to the motors to dispense water		(T) The dropper system will utilize micro syringes to accurately dispense small amounts of water within range	(T) The software shall spin the dropper for enough time for120 µL of water to be dispensed
SR017 (I): The Water Baddies system shall analyze one paper fluidic and one microplastic sample per cycle.	(I) One µPAD will be dispensed and analyzed per sample cycle.				(T) The conveyor belt shall reliably and accurately pass one paper fluidic and one microplastic sample past the camera and microscope per cycle	(D) The dropper system shall reliably and accurately dispense fluids on one paper fluidic and one microplastic sample per cycle	(I/T) Every cycle, the software will analyze two images, one of the paperfluidics and one photo of microplastics
SR018 (T): The Water Baddies colorimetry subsystem shall have a white spectrum LED.				(D) The White LEDs will use the Raspberry Pi GPIO pins.			(D) The White LEDs will turn on when the heavy metals and inorganics samples is ready to be analyzed
SR024 (A/T): The battery life of the system shall last at least 1 hour.				(T) The battery will have enough power for it to last 1 hour			
SR022 (I): The Water Baddies system shall make use of 2 ml reloadable syinges to accept test species into the system.						(A/T) The dropper system will be easily reloadable and refillable by the user and will retain required capacity	
SR010 (D): The Water Baddies system will connect to a standard American outlet, using 120V 60Hz. Plug type A and B. Standard (ANSI C84.1)(NEMA 1-15) (NEMA 5-15)				(I) The components shall be rated to 120V 60Hz, and the battery will be compatible with Type A and B outlets			

SR027 (A): The Water Baddies System shall have an operational temperature between 15-35ºC.	(A) Reagents will maintain proper functionality under operational temperatures of 15-35ºC			(A) All components will run efficiently in this temperature range. The active cooling system will turn on when the Pi is overheated		(T) The dropper system shall remain accurate (± 20µL) within the operational temperature range.	
SR020 (D): In an emergency shutoff, the system shall abort the previous test after being rebooted and give a visible alert from the mobile application.			(T) The mobile application shall display when the system shuts off	(T/D) There will be a cutoff switch to cut off power to All components (except battery).	(D) Once test is aborted, partially used microplastic samples and microfluidic devices will be sent to the disposal tray and the belt will reset for the next test		(T/D) The software shall cycle power to all components and move them to their starting positions after a shutoff
SR023 (I): The Water Baddies systems shall have chamfered edges and moving components covered/shielded from users.					(I) Conveyor belt will be covered so as not to be reachable during operation.		
SR028 (A): The Water Baddies System's components shall not exceed 5 volts and 3 Amps.		(T) LEDs shall operate within this range verified through threshold testing			(T) Required motors will remain in required voltage and amperage range	(T) Required motors will remain in required voltage and amperage range	
SR006 (T): The Water Baddies system shall weigh less than 25lbs.		(T) The optical assy shall weigh <=3lbs		(T) The electrical assy shall weigh 5lbs	(T) The conveyor system and frame shall weigh <8lbs	The dropper assy. Shall weigh 3lbs	
SR009 (D): The Water Baddies GUI shall have an instruction set for using the system.			(D) The mobile application shall have instructions for using the app	(D) The Display will contain the Raspberry Pi GUI.			(D) The software shall load a list of instructions onto the touchscreen
SR011 (D): The Water Baddies system will generate a report outlining the concentration of microplastics within the sample, the concentrations of lead, cadmium, as well as the concentration of both nitrates and nitrites in parts per ml.	(I) Concentration of lead, and cadmium, as well as nitrates and nitrites will be featured in the report.	(A/I) The fluorescence microscope will be able to detect microplastics <500 µm with a concentration accuracy of 85-90%	(D) Shall be able to download a report of the system outputs. Display the graphs of each concentrations				(T/A) The Software will analyze data from the camera and microscope in order to generate the report.

SR021 (D): The Water Baddies App will give an audible and haptic alert if detection levels of each detectable are above the thresholds established by the EPA in their report on National Primary Drinking Water Regulations.			(D) Shall be built using haptic feedback and accessibility features				
SR026 (D): The Water Baddies system shall operate fully autonomously once accepting user input of water reservoir, test material cartridges, and test selection via touchscreen.			(D) The mobile application shall collect result data once the cycle is finished	(D) Coordinate the actions of motors, droppers, lights, cameras, and sensors	(D) The conveyor belt system shall automatically locate the slides on the belt once the cartridge is loaded and the test is started		(D) The software shall run all devices in sequence, and process all data one detecting input from the user
SR025 (D): The Water Baddies App shall have a single connection page to aid the user in connecting to the device.			(D) Shall have a bluetooth button that opens a page and displays all available devices				
SR029 (D): The Water Baddies App shall download the report as a PDF.			(D) Shall have a button to download the PDF				
SR013 (D): The Water Baddies system report shall display one particle concentration per carousel slide			(I) Shall show one group concentration on each page/slide				
SR008 (D): The Water Baddies System shall take a new user 10 minutes to learn.	(I) User's Manual will have an in depth explanation and instructions describing how to load µPADs and perform µPAD analysis.	The microscope will be entirely internal with no need for user input or refocusing	(I/D) Shall have intuitive interface design and instructions of how to use the app		(A) Loading cartridges shall be an intuitive and quick process requiring <1 minute to complete	(A) Loading reservoirs shall be an intuitive and quick process requiring <3 minutes to complete	(D) The GUI for both the App and Raspberry will have detailed instructions, showing step by step how to use it
SR012 (D/I): The Water Baddies App shall connect using Bluetooth 5/Bluetooth Low Energy			(I/D) Shall use a BLE package	(I) The Raspberry Pi will have an integrated Bluetooth Component			
SR019 (D): The Water Baddies system shall make use of reloadable cartridges containing 5 microfluidic pads and 5 optical slides to load the system with testing materials.	(D) Single pad can be removed from cartridge and loaded into system				(D) Cartridges will be of appropriate size to accept 5 of each testing material		

5.0 IDL

Part #	Document	Revision
100100	**Paperfluidic Assembly**	**1**
100110	Chromatography Paper	
100120	Chemical Biomarkers	
100130	Pi Camera	
100140	White Spectrum LED	
100150	Cartridge/Holder	
100160	Minimum Viable Product Definition	1
100170	Paperfluidic Assembly Drawing	1
100200	**Optical Assembly**	**1**
100210	Illumination LED	
100220	Emission Filter	
100230	Microscope	
100240	Test Microplastics	
100250	Meniscus Microscope Slide	
100260	Optical Assembly Drawing	1
100300	**Software Assembly**	**4**
100310	Software Design Document	4
100320	Software System Wrapper	
100330	Colorimetry CSC	
100340	Microplastic and Illuminator CSC	
100350	Dropper CSC	
100360	Conveyor Control CSC	
100370	Conveyor Verification CSC	
100370	Water Baddies Control GUI CSC	
100390	Mobile Application	
100391	Bluetooth Connectivity CSC	
100392	Data Display CSC	
100393	User Navigation CSC	
100400	**Electrical Assembly**	**1**
100410	Battery	

100420	Raspberry Pi 5	
100430	Mini Display	
100440	Active Cooler	
100450	IR Break Sensor	
100460	Adafruit Motor HAT	
100470	Adafruit Motor	
100480	DC-DC Step Down Converter	
100480	Power Distributor	
100490	Electrical Assembly Drawings	1
100491	Wire Schematic	
100492	Circuit Drawing	
100493	Wire List	
100500	**Conveyor Belt Assembly**	**1**
100510	Motor	
100520	Belt and Drive Gear	
100530	Mounts and Framing	
100540	Micro µPADs Cartridge	
100550	Optic Slide Cartridge	
100560	Discard Tray	
100570	Conveyor Belt Assembly Drawing	1
100600	**Dropper Assembly**	**1**
100610	Micro / Macro Syringes	
100620	Stepper Motors	
100630	Pushrod Gearing	
100640	Mounts and Framing	
100650	Dropper Assembly Drawing	1
100700	**Technical Data Package**	**1**
100710	Verification Procedures	1
100720	SKYBASIC Microscope User Manual and Product Specifications	
100730	Pi Camera Data	
100740	Raspberry Pi 5 Datasheet	
100800	**Top Assembly**	**1**
100810	Top Assembly Drawing	1

6.0 Hardware Design

6.1. Wire List

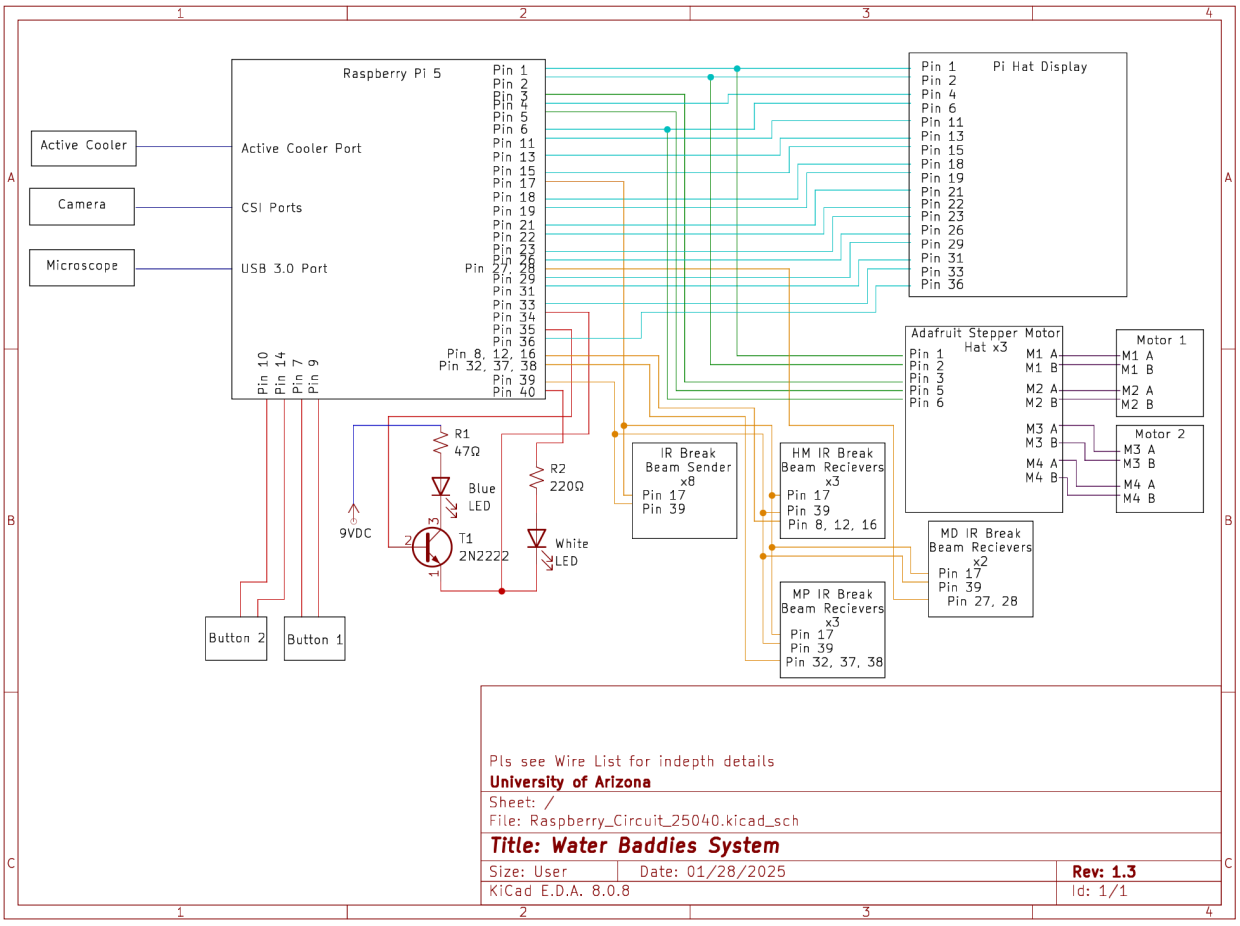
Raspberry Pi Port	Associated Part	Type	Communication	Description	Color
Pin 1	Pi Hat Display/ Adafruit Motor HAT	Single Wire Power	N/A	3.3 Volt Supply	Cyan
Pin 2	Pi Hat Display/ Adafruit Motor HAT	Single Wire Power	N/A	5 Volt Supply	Cyan/ Green
Pin 3	Adafruit Motor HAT	Single Wire Power	I2C (SDA)	I2C Communication	Green
Pin 4	Pi Hat Display	Single Wire Power	N/A	5 Volt Supply	Cyan
Pin 5	Adafruit Motor HAT	Single Wire Power	I2C (SLC)	I2C Communication	Green
Pin 6	Pi Hat Display/ Adafruit Motor HAT	Single Wire Ground	N/A	Ground	Cyan/ Green
Pin 11	Pi Hat Display	Single Wire Power	N/A	LED Red	Cyan
Pin 13	Pi Hat Display	Single Wire Power	N/A	LED Green	Cyan
Pin 15	Pi Hat Display	Single Wire Power	N/A	LED Blue	Cyan
Pin 17	IR Break Beam Sender 1/ IR Break Beam Reciever 1/ IR Break Beam Sender 2/ IR Break Beam Reciever 2/ Button	Single Wire Power	N/A	3.3 Volt Supply	Orange
Pin 18	Pi Hat Display	Single Wire Power	N/A	Button Y	Cyan
Pin 19	Pi Hat Display	Single Wire Power	SPI	LCD SPI MOSI	Cyan
Pin 21	Pi Hat Display	Single Wire Power	SPI	LCD DATA/Command	Cyan
Pin 22	Pi Hat Display	Single Wire Power	SPI	LCD TE	Cyan
Pin 23	Pi Hat Display	Single Wire Power	SPI	LCD SPI SCLK	Cyan
Pin 26	Pi Hat Display	Single Wire Power	SPI	LCD SPI CS	Cyan
Pin 29	Pi Hat Display	Single Wire Power	N/A	Button A	Cyan
Pin 31	Pi Hat Display	Single Wire Power	N/A	Button B	Cyan
Pin 32	Button	Single Wire Power	N/A	Receive Button Signal	Red

Pin 34	Blue LED/ White LED	Single Wire Ground	N/A	Ground	Cyan
Pin 35	Blue LED	Single Wire Power	N/A	Powering Blue LED	Cyan
Pin 36	Pi Hat Display	Single Wire Power	N/A	Button X	Cyan
Pin 37	IR Break Beam Reciever 1	Single Wire Power	N/A	Receive IR Signal	Orange
Pin 38	IR Break Beam Reciever 2	Single Wire Power	N/A	Receive IR Signal	Orange
Pin 39	IR Break Beam Sender 1/ IR Break Beam Reciever 1/ IR Break Beam Sender 2/ IR Break Beam Reciever2 / Button	Single Wire Ground	N/A	Ground	Orange
Pin 40	White LED	Single Wire Power	N/A	Power White LED	Cyan
USB 3.0	Microscope	USB	USB 3.0	Send Data from Microscope	Dark Blue
CSI Port	Camera	15 Pin Ribbon Cable	N/A	Send Data from Camera	Dark Blue
Active Cooler Connector	Active Cooler	Ribbon Cable	N/A	Control Cooler Fan	Dark Blue
Adafruit Stepper Motor HAT Port	Associated Part	Type	Communication	Description	Color
M1 A	Motor 1	Single Wire Power	PWM	Power the Stepper Motor	Magenta
M1 B	Motor 1	Single Wire Ground	N/A	Power the Stepper Motor	Magenta
M2 A	Motor 1	Single Wire Power	PWM	Power the Stepper Motor	Magenta
M2 B	Motor 1	Single Wire Ground	N/A	Power the Stepper Motor	Magenta
M3 A	Motor 2	Single Wire Power	PWM	Power the Stepper Motor	Magenta
M3 B	Motor 2	Single Wire Ground	N/A	Power the Stepper Motor	Magenta
M4 A	Motor 2	Single Wire Power	PWM	Power the Stepper Motor	Magenta
M4 B	Motor 2	Single Wire Ground	N/A	Power the Stepper Motor	Magenta

Pin #	Type of Use	State
Pin 1	3.3V	Used
Pin 2	5V	Used
Pin 3	GPIO 2	Used
Pin 4	5V	Used
Pin 5	GPIO 3	Used
Pin 6	Ground	Used
Pin 7	GPIO 4	Button
Pin 8	GPIO 14	IR
Pin 9	Ground	Button
Pin 10	GPIO 15	Button 2
Pin 11	GPIO 17	Used
Pin 12	GPIO 18	IR
Pin 13	GPIO 27	Used
Pin 14	Ground	
Pin 15	GPIO 22	Used
Pin 16	GPIO 23	IR
Pin 17	3.3V	Used
Pin 18	GPIO 24	Used
Pin 19	GPIO 10	Used
Pin 20	Ground	Button 2
Pin 21	GPIO 9	Used
Pin 22	GPIO 25	Used
Pin 23	GPIO 11	Used
Pin 24	GPIO 8	White LED
Pin 25	Ground	White LED
Pin 26	GPIO 7	Used
Pin 27	GPIO 0	IR Droper
Pin 28	GPIO 1	IR Dopper
Pin 29	GPIO 5	Used
Pin 30	Ground	Used
Pin 31	GPIO 6	Used
Pin 32	GPIO 12	IR
Pin 33	GPIO 13	Used
Pin 34	Ground	Blue LED
Pin 35	GPIO 19	Blue LED
Pin 36	GPIO 16	Used
Pin 37	GPIO 26	IR
Pin 38	GPIO 20	IR
Pin 39	Ground	Used
Pin 40	GPIO 21	Broken

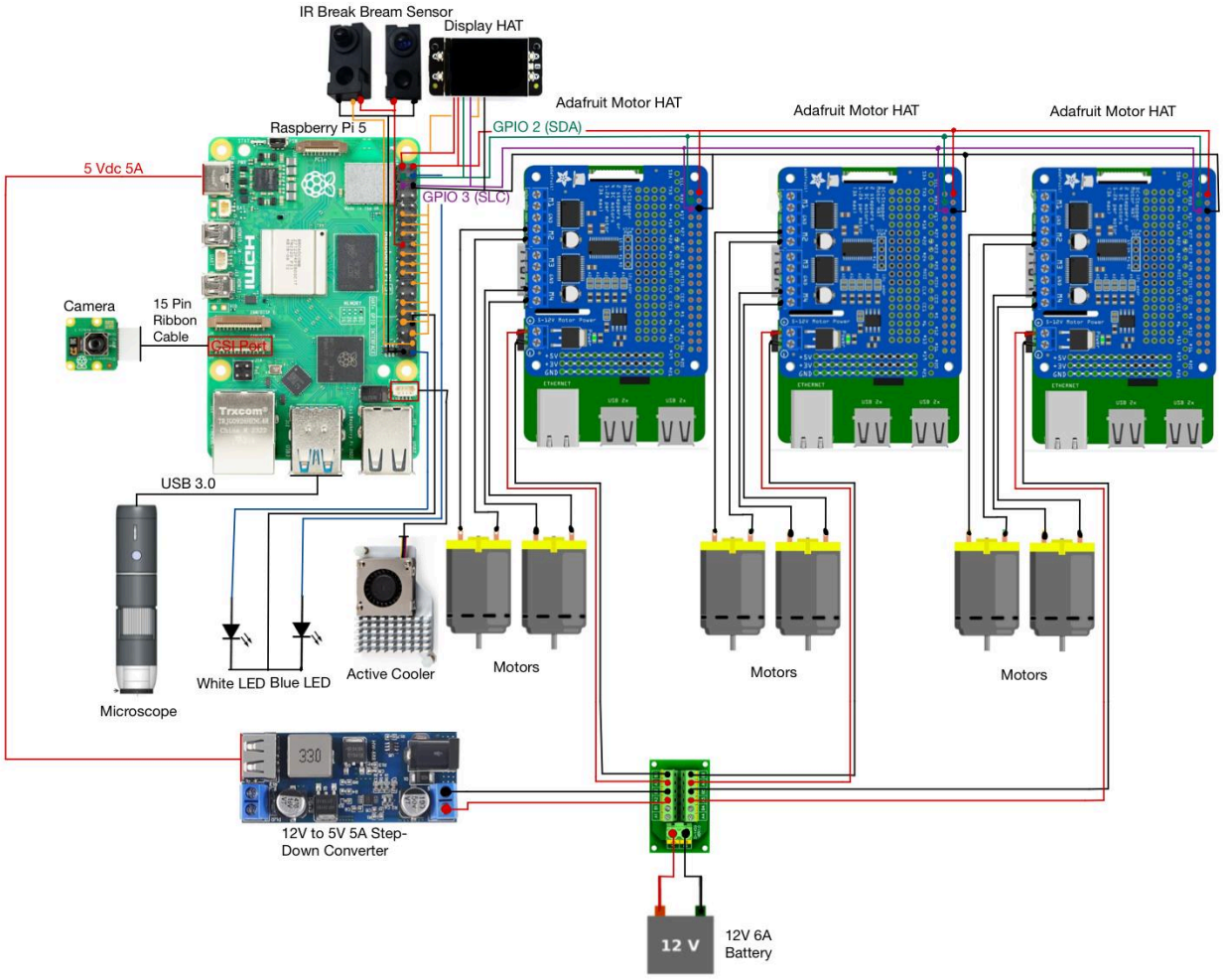
Part 100493: Wire List

6.2. Wire Schematic



Part 100491: Wire Schematic

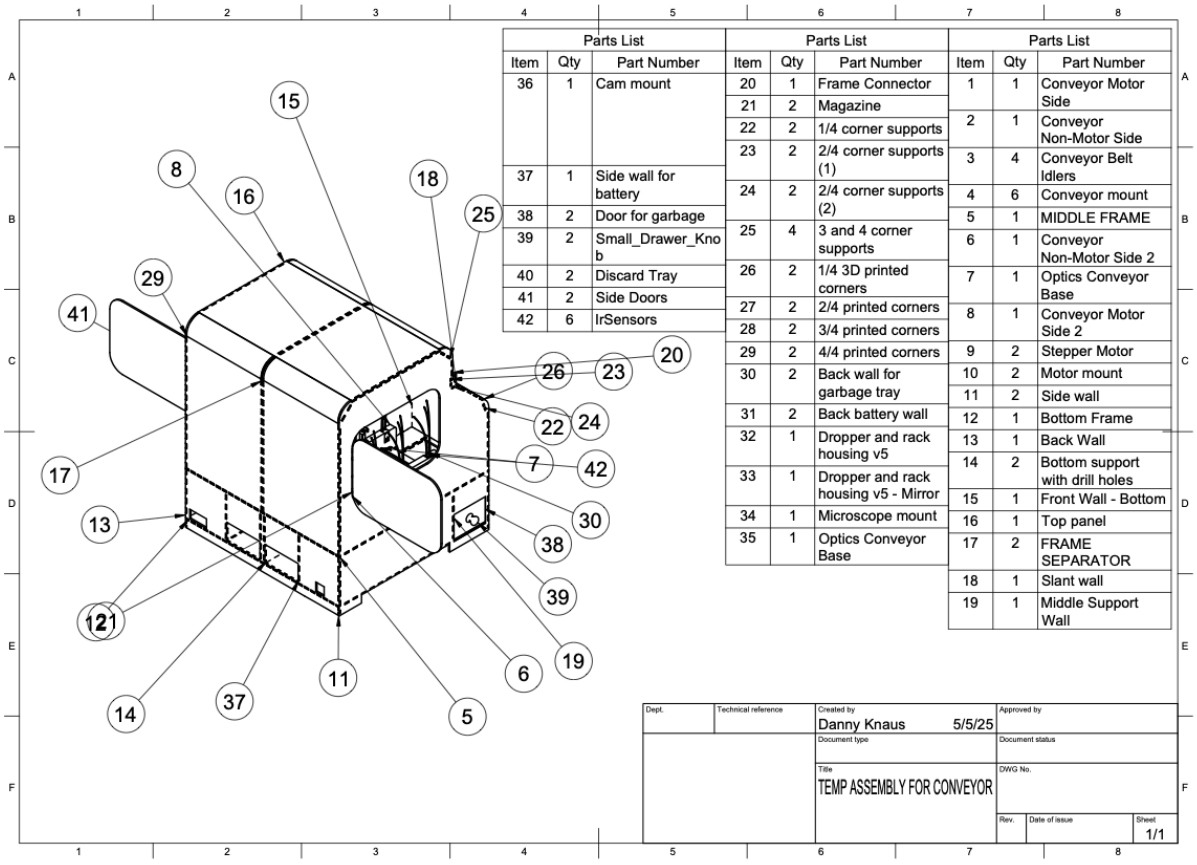
6.3. Circuit Drawing



Part 100492: Circuit Drawing

7.0 Drawings, Schematics, and Data Sheets

7.1. Top Assembly



Parts List		
Item	Qty	Part Number
36	1	Cam mount
37	1	Side wall for battery
38	2	Door for garbage
39	2	Small_Drawer_Knob
40	2	Discard Tray
41	2	Side Doors
42	6	IrSensors

Parts List		
Item	Qty	Part Number
20	1	Frame Connector
21	2	Magazine
22	2	1/4 corner supports
23	2	2/4 corner supports (1)
24	2	2/4 corner supports (2)
25	4	3 and 4 corner supports
26	2	1/4 3D printed corners
27	2	2/4 printed corners
28	2	3/4 printed corners
29	2	4/4 printed corners
30	2	Back wall for garbage tray
31	2	Back battery wall
32	1	Dropper and rack housing v5
33	1	Dropper and rack housing v5 - Mirror
34	1	Microscope mount
35	1	Optics Conveyor Base

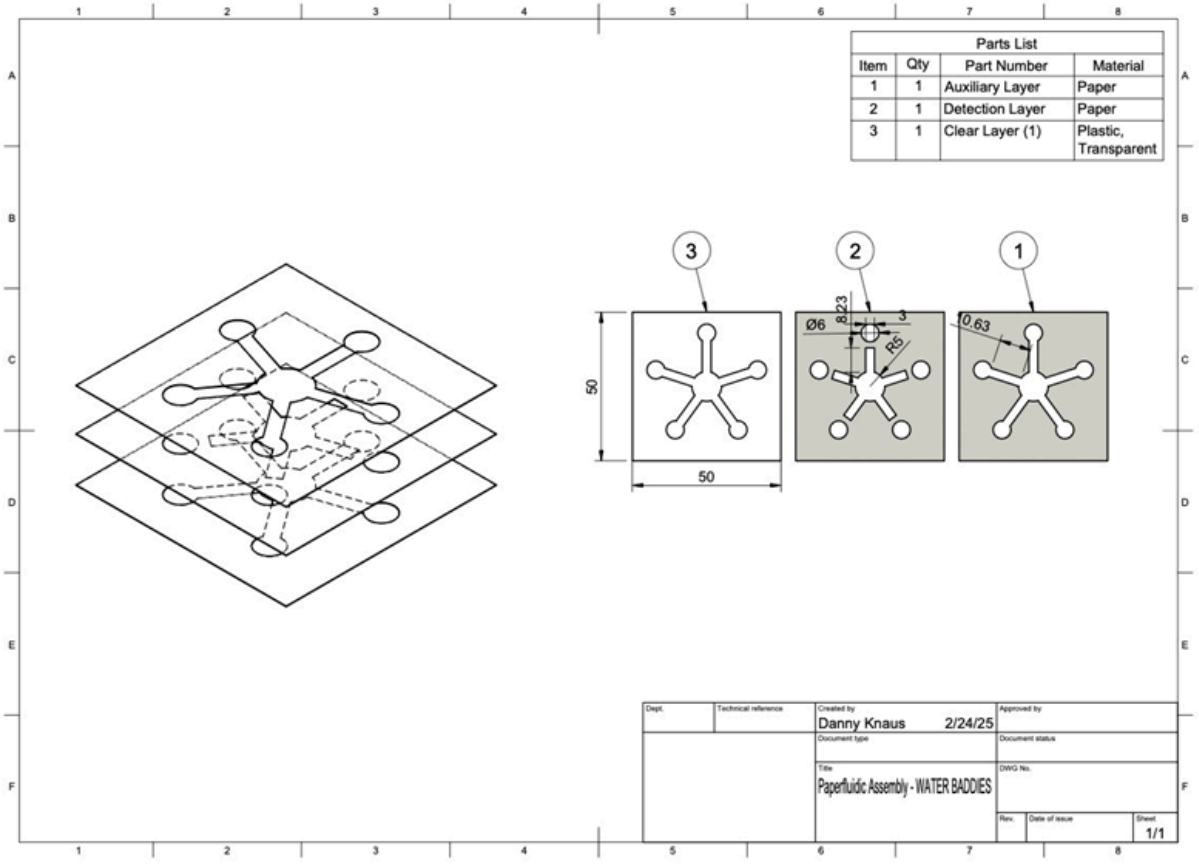
Parts List		
Item	Qty	Part Number
1	1	Conveyor Motor Side
2	1	Conveyor Non-Motor Side
3	4	Conveyor Belt Idlers
4	6	Conveyor mount
5	1	MIDDLE FRAME
6	1	Conveyor Non-Motor Side 2
7	1	Optics Conveyor Base
8	1	Conveyor Motor Side 2
9	2	Stepper Motor
10	2	Motor mount
11	2	Side wall
12	1	Bottom Frame
13	1	Back Wall
14	2	Bottom support with drill holes
15	1	Front Wall - Bottom
16	1	Top panel
17	2	FRAME SEPARATOR
18	1	Slant wall
19	1	Middle Support Wall

Dept.	Technical reference	Created by: Danny Knaus 5/5/25	Approved by
		Document type	Document status
		Title: TEMP ASSEMBLY FOR CONVEYOR	DWG No.
			Rev. / Date of issue / Sheet: 1/1

Part 100810: Top Assembly Drawing

7.2. Paperfluidic Assembly

7.2.1. Paperfluidic Subassembly



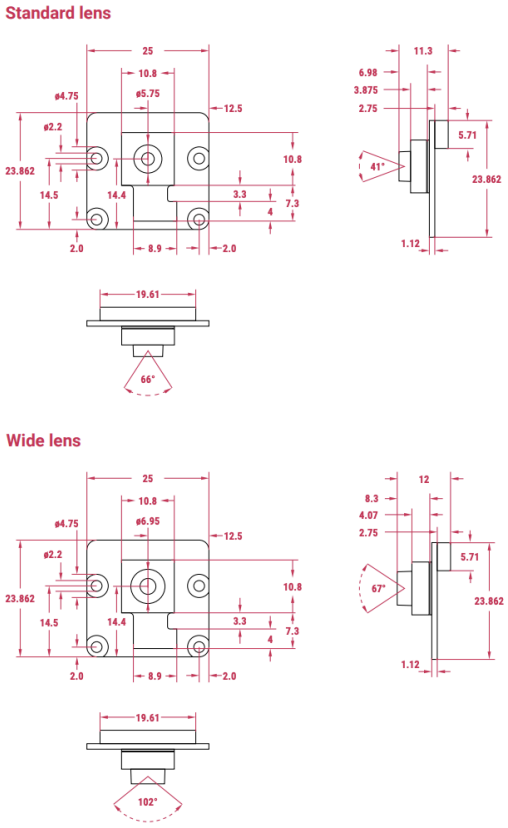
Part 100170: Paperfluidic Assembly Drawing

7.2.2. Chemical Subassembly (MVP1)

Note: There is no drawing available for MVP1 due to the nature of the subassembly being experimentation with chemicals.

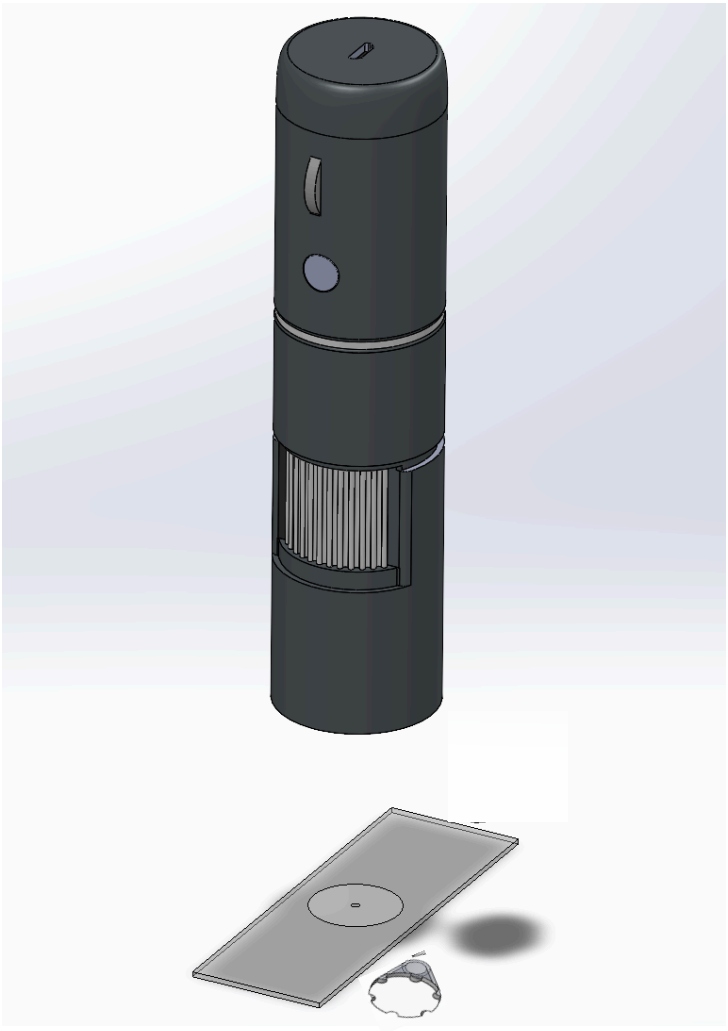
7.3. Optical Assembly

7.3.1. Camera Assembly



Part 100130: Pi Camera Module 3

7.3.2. Microscope Assembly



Part 100210: Illumination LED
Part 100220: Emission Filter
Part 100230: Microscope
Part 100250: Meniscus Microscope Slide

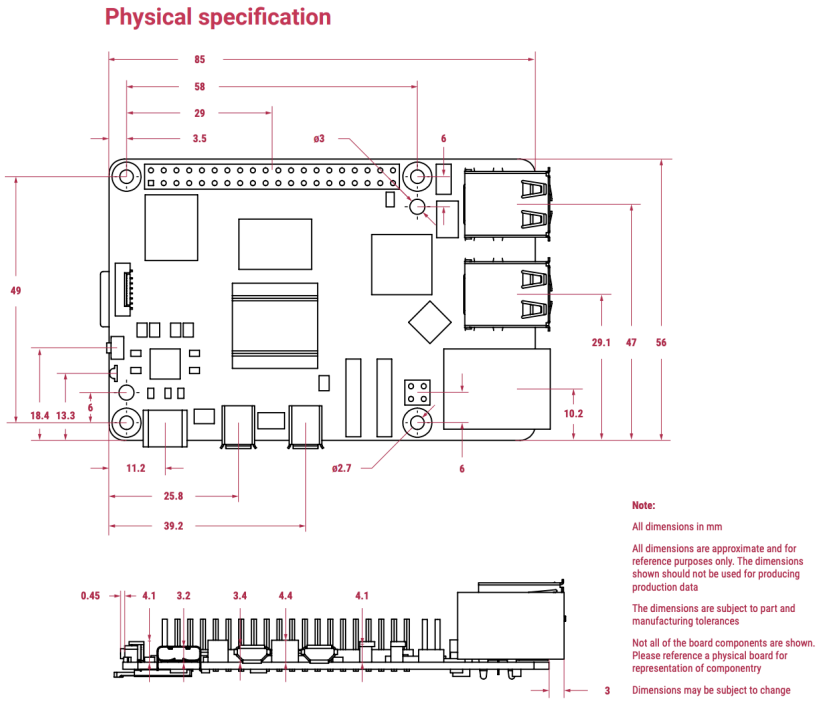
7.3.3. Camera Resolution Datasheet

Sensor:	Sony IMX708
Resolution:	11.9 megapixels
Sensor size:	7.4mm sensor diagonal
Pixel size:	1.4µm × 1.4µm
Horizontal/vertical:	4608 × 2592 pixels
Common video modes:	1080p50, 720p100, 480p120

Part 100750: Camera Specs

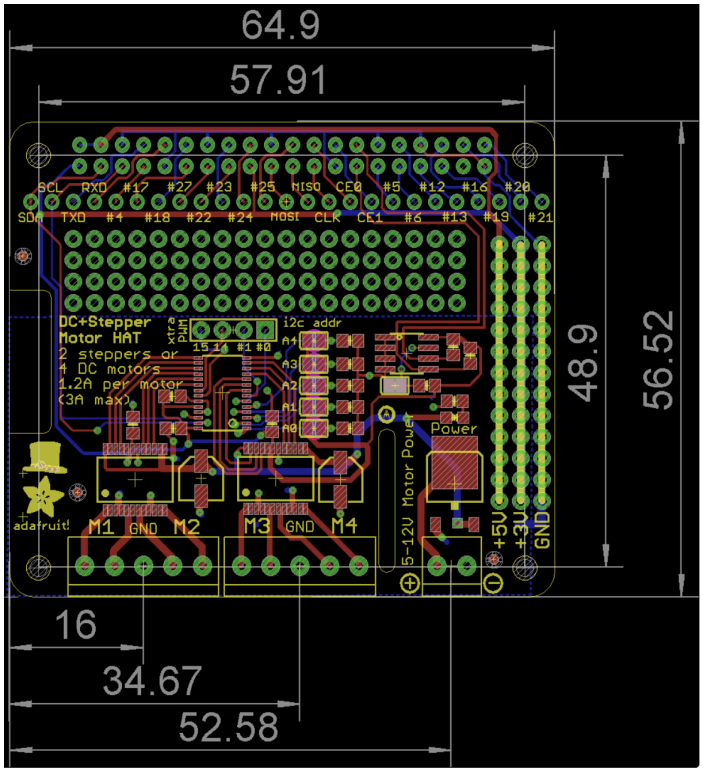
7.4. Electrical Assembly

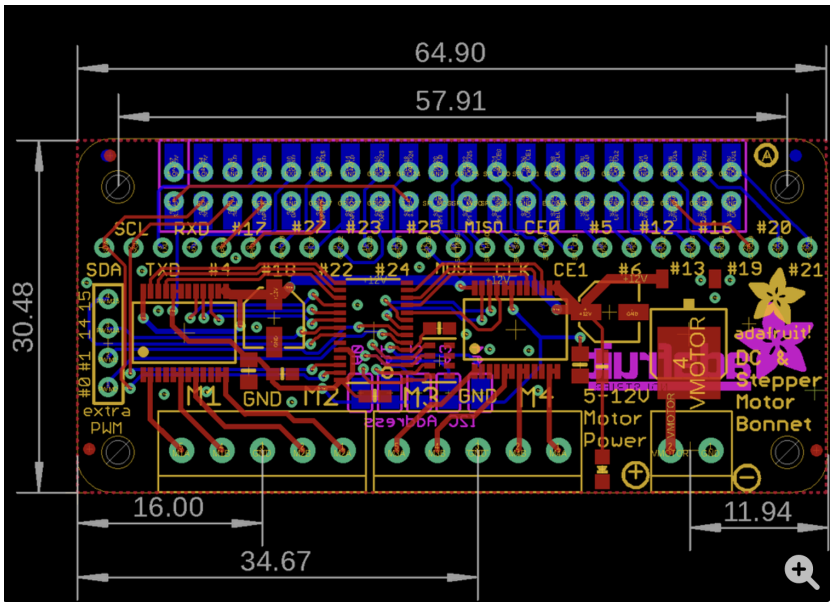
7.4.1. Raspberry Pi 5 Assembly



Part 100420: Raspberry Pi 5

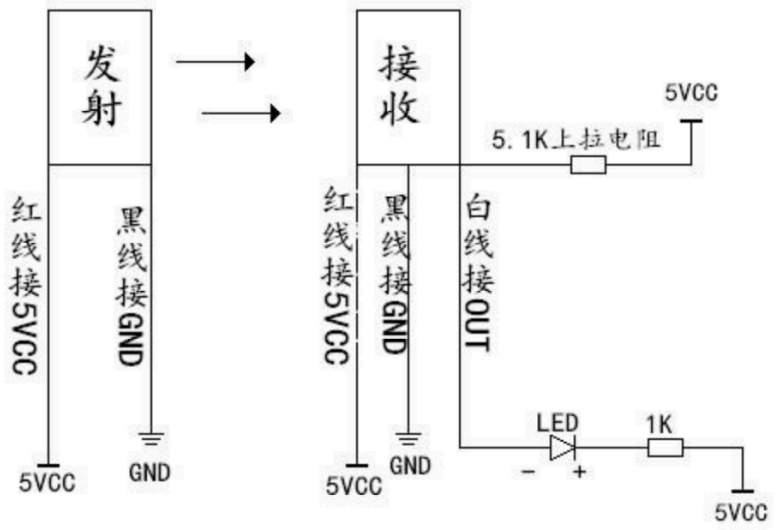
7.4.2. Adafruit Motor HAT Assembly





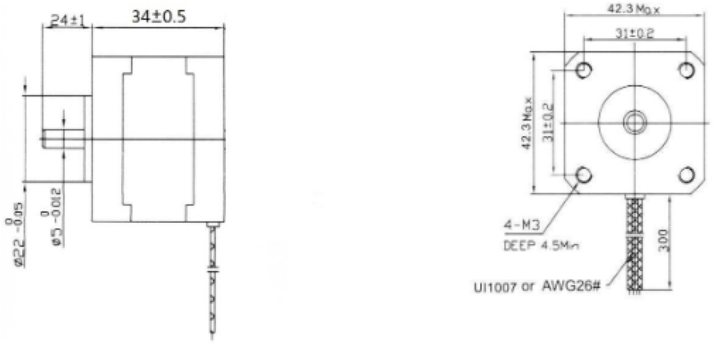
Part 100460: Adafruit Motor HAT

7.4.3. IR Break Sensor Assembly

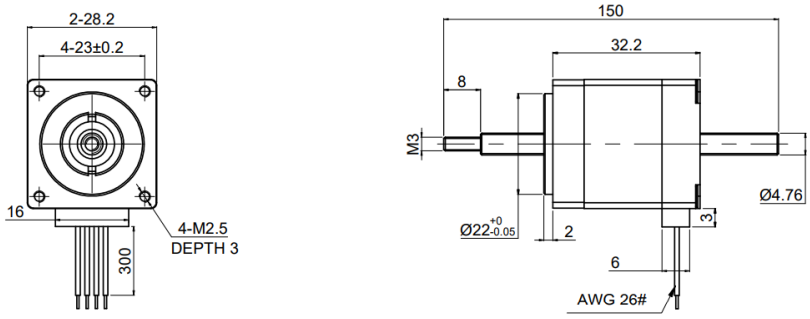


Part 100450: IR Break Sensor

7.4.4. Motor Assembly

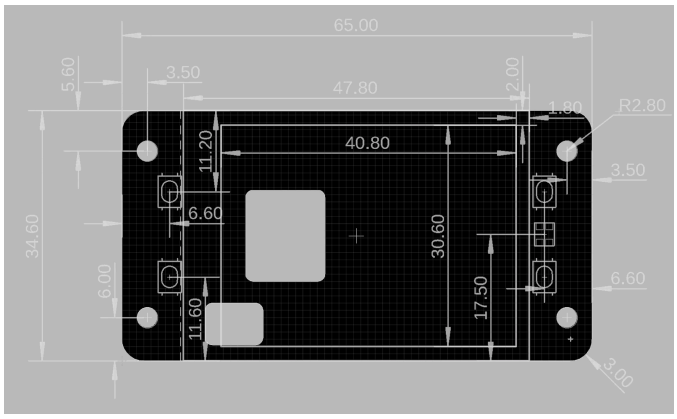


Part 100470: Stepper Motor



Part 100620: Linear Actuator Stepper Motor

7.4.5. Pi HAT Mini Display Assembly



Part 100430: Mini Display

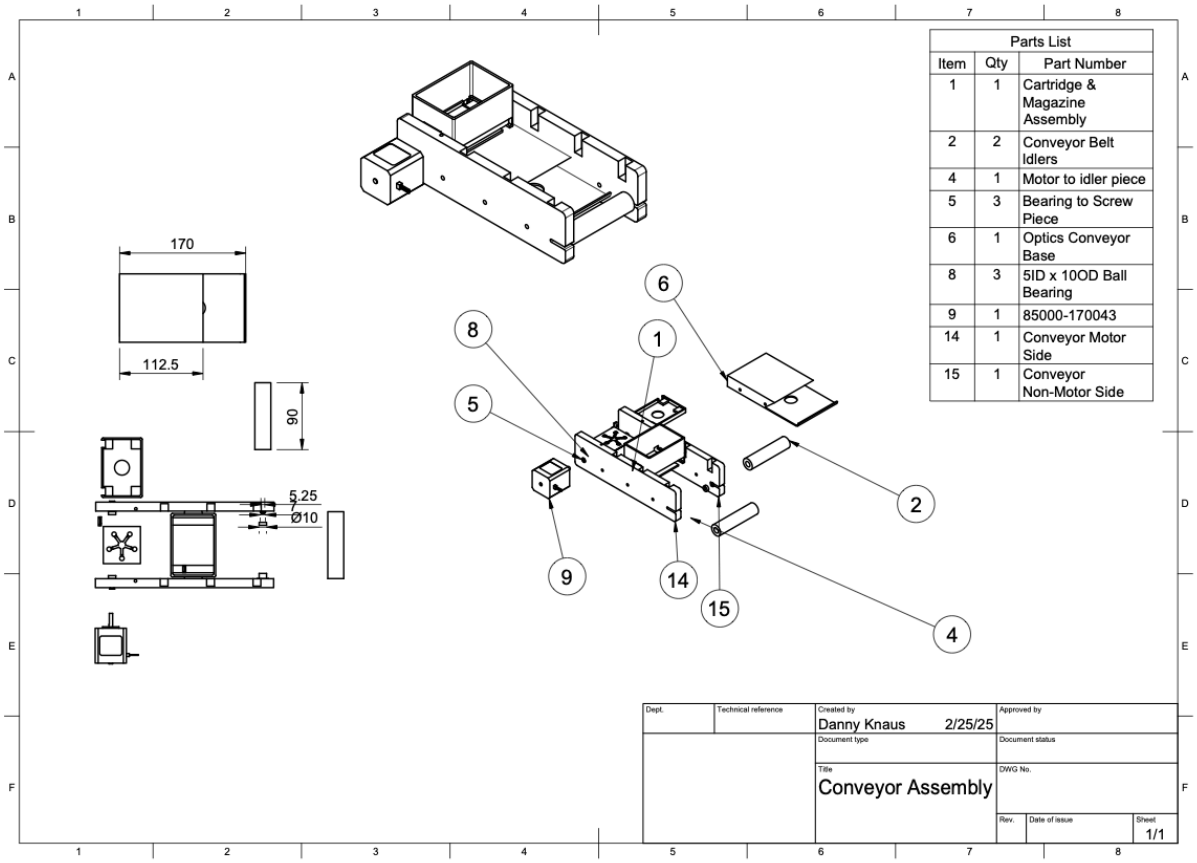
7.4.6. Image of Raspberry Pi 5 Datasheet 4gb

List price:

2GB	**$50**
4GB	**$60**
8GB	**$80**
16GB	**$120**

Part 100760: Raspberry Pi 5 Datasheet

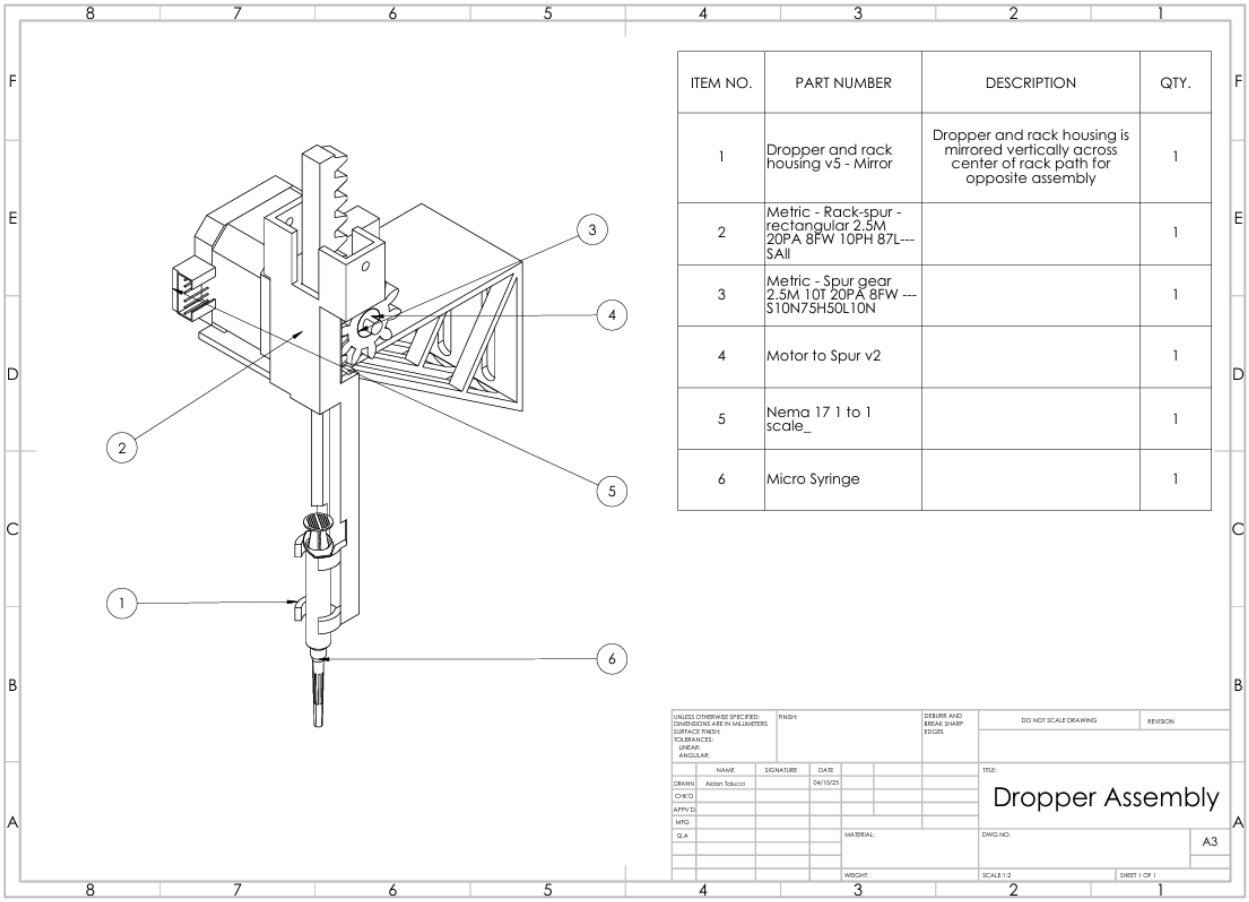
7.5. Conveyor Belt Assembly



Part 100500: Conveyor Belt Assembly

Note: There are two conveyor belts with mirrored side walls. Shown above is the optics side. The paperfluidics version will have an altered version of the base without any holes for lights.

7.6. Dropper Assembly



ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	Dropper and rack housing v5 - Mirror	Dropper and rack housing is mirrored vertically across center of rack path for opposite assembly	1
2	Metric - Rack-spur - rectangular 2.5M 20PA 8FW 10PH 87L---SAll		1
3	Metric - Spur gear 2.5M 10T 20PA 8FW --- S10N75H50L10N		1
4	Motor to Spur v2		1
5	Nema 17 1 to 1 scale_		1
6	Micro Syringe		1

	NAME	SIGNATURE	DATE
DRAWN	Aidan Tolucci		04/10/25
CHK'D			
APPV'D			
MFG			
Q.A			

DO NOT SCALE DRAWING | REVISION

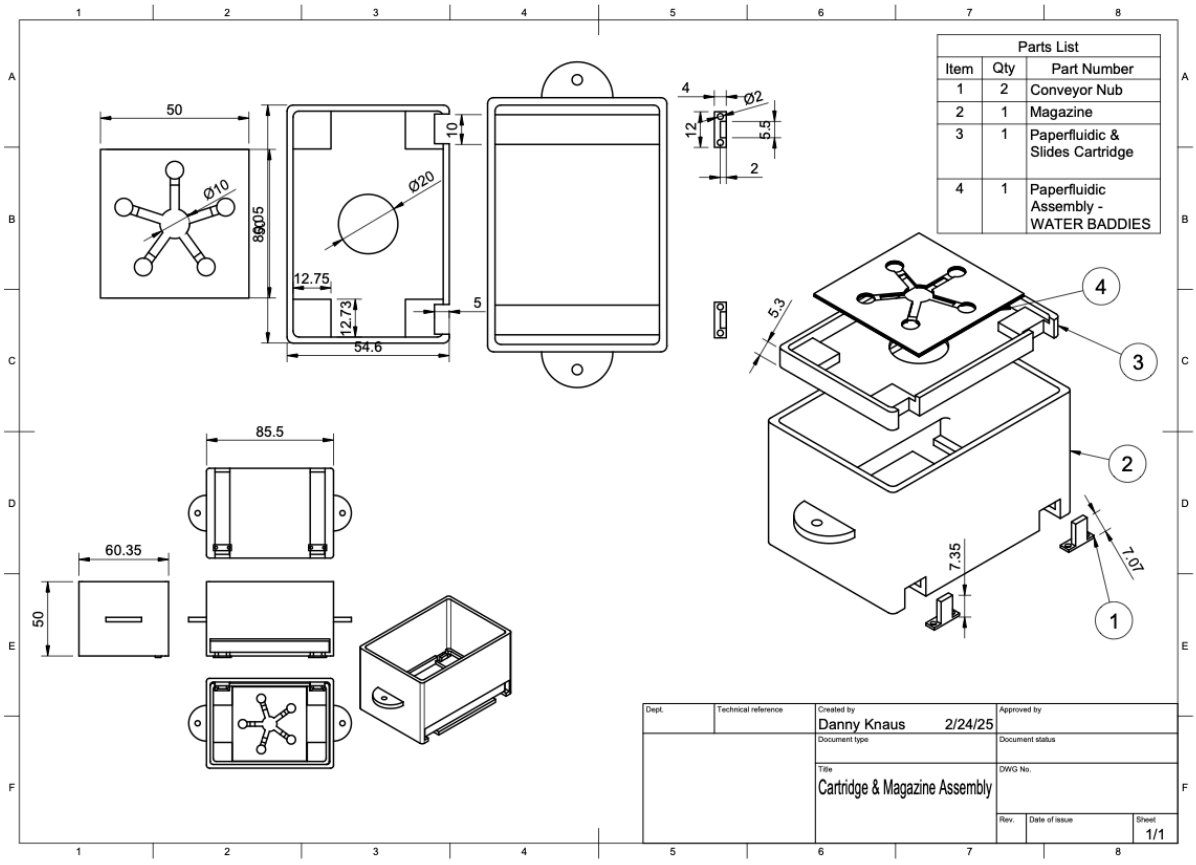
TITLE: Dropper Assembly

MATERIAL: | DWG NO. | A3

WEIGHT: | SCALE:1:2 | SHEET 1 OF 1

Part 100600: Dropper Assembly

7.7. Cartridge & Magazine Assembly

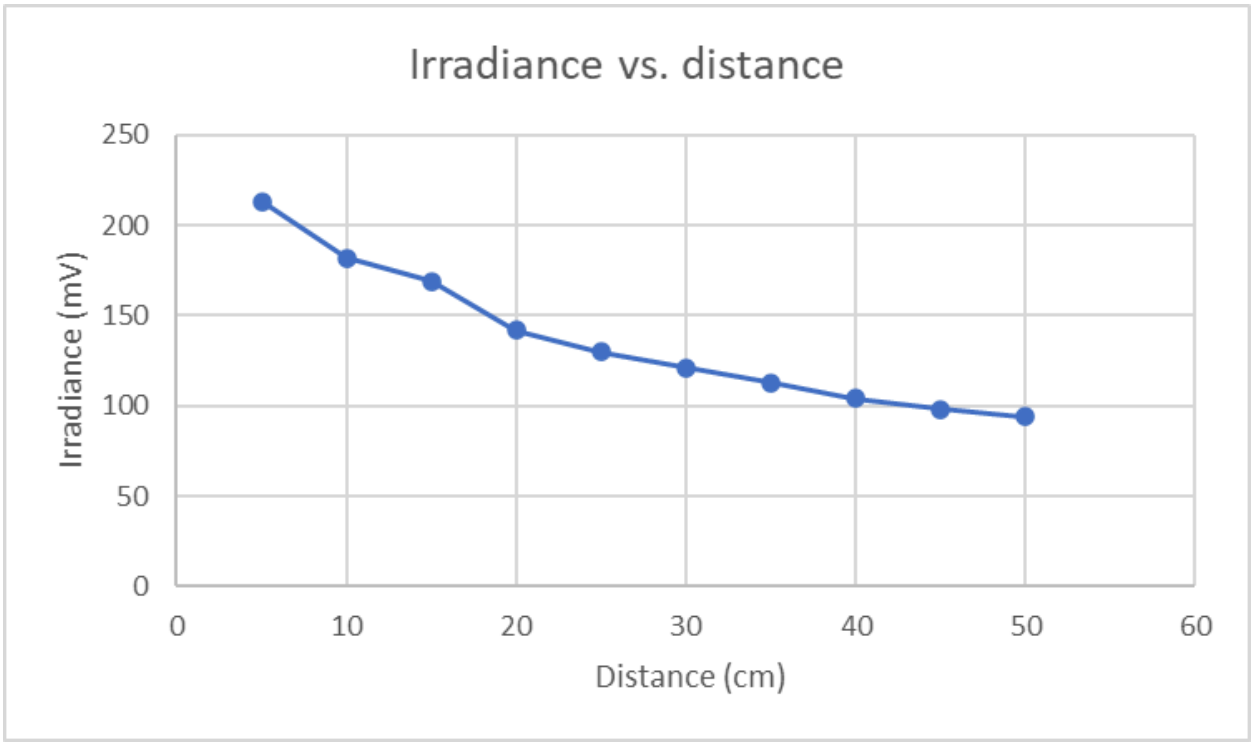


8.0 Models

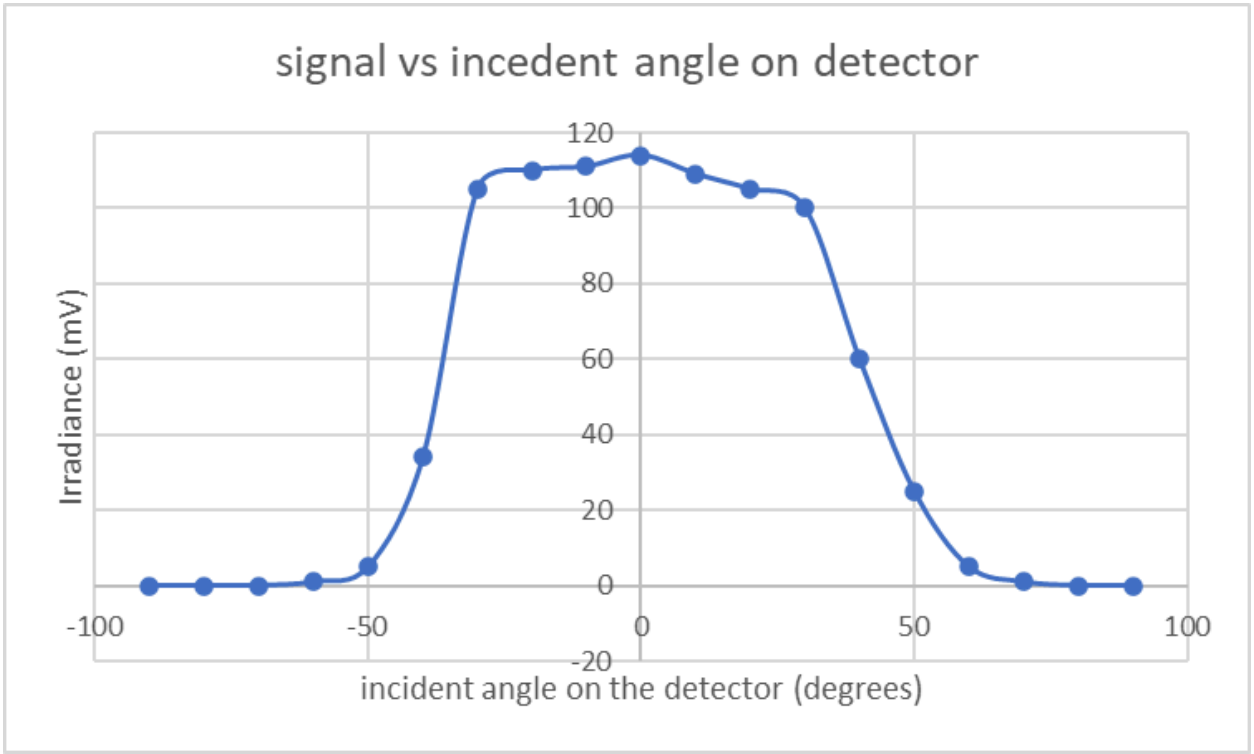
Analysis: Detection System Voltage

The Water Baddies System's components shall not exceed 12 volts and 6 Amps.

Components	Quantity	Voltage(V)	Current(mA)
Raspberry Pi 5	1	5	3000
Display	1	5	150
Camera	1	3.3	300
Active Cooler	1	5	300
White LED	1	3.3	16
Blue LED	1	5	16
Microscope	1	5	900
Adafruit Motor Hat	3	5	20
Motors	6	12	400
IR Beam Sensor	2	3.3	120
Total:		12	5742



Irradiance of white LED as a function of distance: Demonstrating the importance of proximity for best illumination.

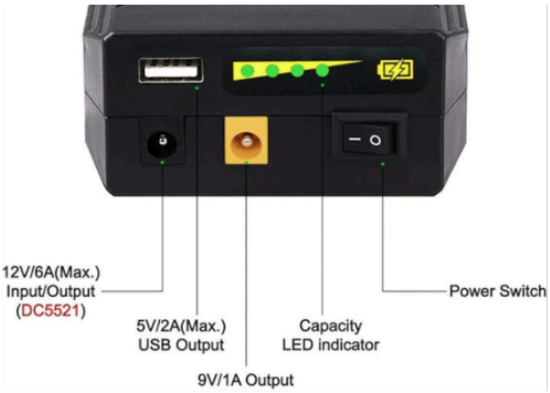


Signal collected as a function of angle: Indicating the importance of maintaining perpendicularity between source LED and sample being illuminated.

Analysis: Detection System Battery Life

The battery life of the Water Baddies Detection System shall last at least 1 hour.

We need a battery that has a minimum of 6000mAh at 12V. We have found a battery that gives us 8300mAh at 12v. Giving us a conservative estimate of a runtime of 1hr and 23mins. Testing is needed to see how the system performs in real life.



9.0 Final Budget and Bill of Materials

The final bill of materials is below showing the final spending. In total we spent $2,751.88 for development and testing of the Water Baddie device. Order number 23 never arrived and a refund was not able to be acquired. Additionally, removing items for testing and unused equipment, the system cost for one unit would be $729.15 making out device affordable for its future use in design projects as well as a point of care device.

Bill of Mat

Number	Date	Part Number	Assy.	Vendor	Status	Name/Description	Qty	Price (Including Shipping)
1	10/02/2024	4GB-9024	Electrical	PiShop	Arrived	Raspberry Pi	1	$107.51
2	10/2/2024	46-1	Electrical	PiShop	Arrived	Pi Camera	1	$25.00
3	10/2/2024	374-1	Electrical	PiShop	Arrived	Cooler	1	$6.40
4	10/2/2024		Optical	Amazon	Arrived	Wireless digital Microscope	1	$32.60
5	10/29/2024	18470	Biomedical	Sigma Aldrich	Arrived	Bromothymol Blue	1	$41.10
6	10/29/2024	C7352	Biomedical	Sigma Aldrich	Arrived	Cystine	1	$33.73
7	10/29/2024	U5128	Biomedical	Sigma Aldrich	Arrived	Urea	1	$45.80
8	10/29/2024	WHA303	Biomedical	Sigma Aldrich	Arrived	Whatman Paper	1	$100.60
9	10/6/2024	FBH520-40	Optical	thorlabs	arrived	Bandpass FIlter	1	$182.77
10	10/6/2024	1824	Electrical	PiShop	Arrived	1 x Display HAT Mini	1	$31.20
11	10/6/2024	324	Mechanical	Adfruit	Arrived	Stepper Motor (2x)	2	$33.83
12	10/6/2024	2348	Mechanical	Adfruit	Arrived	AdaFruit Motor HAT	1	$28.33
13	10/6/2024	2167	Mechanical	Adfruit	Arrived	Beam Sensor (2x)	6	$29.72
14	11/6.2024		Optical	PiShopUS	Arrived	Hard-Coated Bandpass FIlter	1	$196.27
15	11/8/2024	CS-PID-301	Software	PiShop	Arrived	Micro-HDMI to HDMI cable for P	1	$11.63
16	11/19/2024	PUV-14	Optical	Amazon	Arrived	UV light safety glasses	1	34.54
17	12/2/2024	BS-C12	Optical	AmScope	Arrived	Microscope slides 15 Pack	1	$22.28
18	1/16/2025		Biomedical	Amazon	Arrived	Black Crayons (12 Pack)	1	$5.42
19	1/21/2025		Electrical	Amazon	Arrived	Talentcell 12V Lithium Ion Battery	2	$104.34
20	1/21/2025		Mechanical	Amazon	Arrived	10 Pack Glass 2ml Syringes	1	$12.82
21	1/21/2025		Mechanical	StepperOnline	Arrived	Nema 17 Non-captive 34mm Stack	1	$53.74
22	1/22/2025		N/A	UA Bookstore	Arrived	Polo Shirts	7	$304.36
23	1/27/2025	17N13S0404AF4-150RS	Mechanical	StepperOnline	Ordered	Nema 17 Non-captive 34mm Stack	2	$100.88
24	2/6/2024		Biomedical	Amazon	Arrived	Varisy 17 in 1 Drinking Water Test	1	$26.99
25	2/6/2024		Biomedical	Amazon	Arrived	Phosphate Test Kit	1	$19.99
26	2/6/2024		Biomedical	Amazon	Arrived	Liberty Gold Standard Heavy Metal Water	1	$15.99
27	2/6/2024		Biomedical	Amazon	Arrived	System Quick Arsenic Water Test	1	$42.82
28	2/10/2025	11N13S0424ED6-150RS	Mechanical	StepperOnline	Arrived	Nema 11 Non-captibe 32.2mm Stack	4	$177.60
29	2/10/2025		Mechanical	Home Depot	Arrived	24 in x 24 in - 16 Gauge Steel Sheet Metal	2	$127.86
30	2/10/2025		Mechanical	Home Depot	Arrived	1/4 in Zinc Flat Washer (100-Pack)	1	$7.97
31	2/10/2025		Mechanical	Home Depot	Arrived	1/4 in. -20 Stainless Cap Nuts (25-pack)	1	$6.87
32	2/10/2025		Mechanical	Home Depot	Arrived	1/4 in X 1in Zinc Plated Hex Bolt(100-pack	1	$15.30
33	2/10/2025		Mechanical	Home Depot	Arrived	1/4 in Zinc Plated Hex Nut (100-Pack)	1	$8.98
34	2/10/2025		Mechanical	Home Depot	Arrived	1/4 in X 3 ft Zinc Platted Steel Thread Rod	3	$22.58
35	2/12/2025		Optical	Amazon	Arrived	ONiLAB Lab Micropipette	1	$29.34
36	2/12/2025		Optical	Amazon	Arrived	Micropipette Tips ONiLAB	1	$30.98
37	2/17/2025	2348	Electrical	Adafruit	Arrived	Adafruit DC & Stepper Motor HAT for Rasp	1	$34.79
38	2/17/2025	737	Electrical	Adafruit	Arrived	Power Distribution Bus	2	$3.90
39	2/21/2025	2167	Electrical	Adafruit	Arrived	IR Break Beam Sensors	3	$19.61
40	2/21/2025		Biomedical	Sigma Aldrich	Arrived	Sodium Nitrite	1	$112.37
41	2/21/2025		Biomedical	Sigma Aldrich	Arrived	Sodium Nitrate	1	$84.16
42	3/18/2025		Mechanical	Amaon	Arrived	Borosilicate Glass Syringe	1	$27.82
43	3/19/2025		Mechanical	Adafruit	Arrived	Stepper motor - NEMA-17	2	$57.11
44	4/1/2025		Electrical	Amazon	Arrived	Dupont Wires	1	$21.72
46	4/4/2025		Electrical	Amazon	Arrived	Electrical Tape Black	1	$6.90
47	4/4/2025		Electrical	Amazon	Arrived	Zipties Black	1	$6.90
48	4/10/2025		Electrical	Amazon	Arrived	Flat Ribbon Cable 2.54mm Pitch 40 Pin	1	$15.39
49	4/10/2025		Mechanical	Amazon	Arrived	50x Magnets 10x3 mm	1	$9.97
50	4/10/2025		Electrical	Amazon	Arrived	UCTRONICS Male to Female GPIO Ribbon	1	$7.99
51	4/10/2025		Electrical	Amazon	Arrived	120pcs 15cm Dupont Wire Male to Female Bre	1	$7.49
52	4/10/2025		Electrical	Amazon	Arrived	120pcs 10cm Dupont Wire Male to Femal	1	$6.98
53	4/25/2025		Mechanical	Amazon	Arrived	16 Pack Baltic Birch Bords 24x36x1/8	1	$145.64
54	4/25/2025		N/A	Fast Copy	Arrived	Poster	1	$105.00
						Total:		$2,751.88

10.0 Software Documentation (SDD) Overview

The Water Baddies Software Description Document will cover the in-depth architecture, design solutions, and functions of the software behind the Water Baddies System. The document will discuss the reference documents used to design and develop the software subsystems, the design considerations made for individual subsystems or functionalities and why those were chosen, the desired stimuli and responses from each of the subsystems, the computer software components in the system, the requirements traceability, and any additional notes relating to the software note discussed earlier in the document. The full SDD can be found in the appendix below.

The software architecture of the Water Baddies System is a component based architecture. Every functional element needed for the functionality of the Water Baddies Detection System and the Water Baddies App, is broken up into its respective functional component. The components then either communicate with each other through listeners, bluetooth, or through a wrapper script. Listeners are used on the Water Baddies App to communicate between bluetooth functions and data display functions. Bluetooth is used to communicate between the Raspberry Pi and the Bluetooth component of the Water Baddies App. For the software that controls physical components on the Water Baddies Detection System, those are controlled by a system wrapper which serves to coordinate functions across the entire system.

The Water Baddies App Contains 4 individual components which work together to serve up the app. The Bluetooth Connectivity CSC contains the logic for connecting to the Raspberry Pi over bluetooth low energy. The Data Display CSC creates bar charts for the data received from the Water Baddies Detection System, and serves up information on each of the water baddies and their harmful affects. The User Navigation CSC controls the navigation between the different pages of the app: data display, history, and about. Finally the Cloud Database shown in the Software Architecture Diagram refers to the google firebase connectivity used to store all user data and do big data analytics.

The Water Baddies Detection System includes the Raspberry Pi, all of the hardware components, and software components to control the hardware. The Colorimetry Capture CSC works with the Pi Camera Module 3 to take photos of the paper fluidics and determine the color change and associated concentration of metals and/or inorganics in the water sample. The Microplastic Illuminator and Capture CSC controls the LED to floresce the microplastic, saves and image from the microscope of the florescence, and analyzes that image to determine a concentration. The Dropper CSC controls the motors which push out water and chemicals onto the slides. The Conveyor Control CSC moves the conveyor belt and with it the optical and paper fluidic slides to their necessary locations. The Conveyor Verification Sensor CSC confirms the slides made it to their correct locations when needed. The Water BAddies COntrol GUI CSC creates the GUI and button functionalities for the inlay display. FInally the System Wrapper coordinates all of the functions of the Water Baddies Detection System from the time the device is turned on to the time its turned off.

Overall the software of the Water Baddies System works together to seamlessly provide all of the functionalities necessary. These functions meet all of the requirements for the system and aim to give users a better sense of what is in their water. For a deeper look into the software and each component see the full Software Design Document in the appendix.

11.0 Lessons Learned and Recommended Next Steps

One of the first things we noticed was the difficulty in securing necessary lab space for chemical testing. Due to the need for chemical testing equipment, like a fume hood, we were already limited in which labs we could work in. Securing lab space was difficult and we quickly learned that the second you realize you will be conducting chemical testing it is important to start the process immediately. Additionally, we learned that the entire team should be involved in designing the main framing/mounting even if that would traditionally be thought of as a mechanical engineer responsibility. When developing an autonomous integrated system with moving parts involving multiple detection methods, extensive communication is required so nothing is built incorrectly causing complete redesign. An iterative process was useful, but for a time-limited project, limiting long design processes is key. For our device, we need cameras and microscopes in very precise spots that make it vital for everyone to be working together on the framing of the device. Another lesson we quickly learned is that microplastics are so abundant it makes testing difficult. Even distilled water that was held in a plastic bottle had microplastics making it extremely hard to test our concentration scripts when we couldn't be sure how much microplastic was actually in the sample. Additionally, project planning and timing is important because integration would help with concentration verification. Verifying the concentration script through testing with the enclosed system would ease the discrepancies and increase our ability to conduct trials quickly.

For recommended next steps, we would first expand the mobile applications to include more "big data" processing capabilities to better provide more revealing information across larger populations. Additionally, the creation of an arsenic unit that can safely and accurately detect arsenic in water would be a next step. Additionally arsenic detection was in the scope of this project, but concerns about arsine gas generation lead to this no longer being an option. In the same vein, mercury detection could also be a nice addition to the paper fluidic but testing concerns inhibited us from pursuing that in this project. A quicker and easier way to make the paper fluidics would also be nice, as it would allow for more consistent quality and remove potential sources of human error. The next steps would include simplifying the design and decreasing the size of the device. Wire management would decrease the height of the device because the male to female pins have a higher clearance than soldering them. The conveyor belts could be incorporated into the sides of the device to decrease the width so the system is more compact and marketable. Overall we learned the importance of project timeline and predicting potential limitations so that there can be risk mitigation plans in place before problems arise. Collaborating with multiple disciplines has potential for knowledge gaps, so we learned the importance of effective communication within a cross-disciplinary team.

12.0 Appendix

1. Software Description Document
2. Verification Plans
3. Part Drawings

Software Design Document

Version 1.4

Document #100610

March 31st, 2025

WATER BADDIES - Microplastic, Heavy Metal and Inorganics Water Detection System for Environmental and Human Health

Team 25040

Brendan Bamberg

Jameson Brehmer

Daniel Knaus

Austin Medina

Alia Nichols

Aidan Talucci

Tyler Thursby

Revision History

10/15/2024 Rev(–) Created the initial draft of the SDD
1/23/2025 Rev(1.1) Added in design decisions and architecture discussions
1/25/2025 Rev(1.2) Formatted document and finished the design details section
1/27/2025 Rev(1.21) Added figure identifiers
3/2/2024 Rev(1.3) Modified the SDD to update SystemWrapper and CSCs, updated the wire list, wire diagram, Water Baddies Control GUI, and system block diagram, and added SR028 datasheet
3/31/2025 Rev(1.4) Updated architecture diagrams and added Analytics description

Table of Contents

1.0 Scope

1.1. Identification

This Software Design Document describes the architecture, interfaces, and functions of the Water Baddies Software System. The system includes the Water Baddies App and the Water Baddies Pi..

1.2. System Overview

The Water Baddies System shall be a lightweight, cheap, and portable system aimed at detecting microplastics, metals, and inorganics (collectively referred to as *baddies)* in water samples. This project is a continuation of the previous year, when the team struggled with the size, ease of use, and detection of microplastics smaller than a certain size. The Water Baddies System aims to further the project by distinguishing between dust and microplastics at the micrometer level, detecting metals like lead and cadmium; detecting inorganics like nitrate, nitrite, and phosphate; and developing an app to provide a streamless delivery of information to the end user. These baddies, when ingested, have been found to cause kidney and liver problems, along with other major health problems. Once ingested, the baddies are also incredibly resilient, being incredibly difficult to remove from the body, causing long-term effects to those who didn't realize they were ingesting them in the first place. The system will allow users to detect the baddies and some of their quantities, although ingesting the baddies in any quantity is potentially harmful to the user. While all of these materials are detectable in some form, a system doesn't exist to identify all 3 types of baddies on a small level while only using one device to do so. The Water Baddies System also aims to display the results in a modern way through the use of a mobile app and local history on the app to track water quality over time and location.

1.3. Document Overview

The Water Baddies Software Description Document will cover the in-depth architecture, design solutions, and functions of the software behind the Water Baddies System. The document will discuss the reference documents used to design and develop the software subsystems, the design considerations made for individual subsystems or functionalities and why those were chosen, the desired stimuli and responses from each of the subsystems, the computer software components in the system, the requirements traceability, and any additional notes relating to the software note discussed earlier in the document.

2.0 References Documents

The SDD references the following documents:
https://docs.flutter.dev/
https://pub.dev/packages/flutter_blue_plus

3.0 Design Decisions

The design decision section will discuss a general overview of the system, providing context for the architecture of the system. The section will also discuss a few major design decisions for the software of the system and discuss the reasons why specific design decisions were made.

3.1. Physical Design

The software produced for the Water Baddies Detection System is a direct product of the physical architecture and design as indicated by the physical needs of the system. Once the team had determined the hardware needed to achieve our project goal, we then crafted a software design that would support all hardware and satisfy any remaining requirements. Below is the system architecture diagram with all major physical components of the system and how they interact with other portions of the system, shown to provide context to the design decisions discussed below.

One key feature and goal of the Detection System is minimal user intervention and a focus on the system's autonomy, strengthening the need for more complex software systems. At the center of our system is a Raspberry Pi 5, which is responsible for controlling all incoming user input, system input, app input, and distributing all system output. Some of the Raspberry's responsibilities include but are not limited to: controlling the motors to move the paper fluidics and slides, collecting and processing images from the camera and microscope, sensing the levels of water, paper fluidics, and slides in the system, and sending processed data to the mobile app via Bluetooth.

The system uses the newest version of Raspberry Pi, the Pi 5. Our specific CPU is the Raspberry Pi 5 (4GB), as we determined this would be the optimal amount of RAM needed to process our images and data without incurring too much cost. The Pi 5 has the following specifications:

- Broadcom BCM2712 2.4GHz quad-core 64-bit Arm Cortex-A76 CPU, with cryptography extensions, 512KB per-core L2 caches and a 2MB shared L3 cache
- VideoCore VII GPU, supporting OpenGL ES 3.1, Vulkan 1.2
- Dual 4Kp60 HDMI® display output with HDR support
- 4Kp60 HEVC decoder
- LPDDR4X-4267 SDRAM (2GB, 4GB, and 8GB)
- Dual-band 802.11ac Wi-Fi®
- Bluetooth 5.0 / Bluetooth Low Energy (BLE)
- microSD card slot, with support for high-speed SDR104 mode
- 2 × USB 3.0 ports, supporting simultaneous 5Gbps operation
- 2 × USB 2.0 ports
- Gigabit Ethernet, with PoE+ support (requires separate PoE+ HAT)
- 2 × 4-lane MIPI camera/display transceivers
- PCIe 2.0 x1 interface for fast peripherals (requires separate M.2 HAT or other adapter)
- 5V/5A DC power via USB-C, with Power Delivery support
- Raspberry Pi standard 40-pin header
- Real-time clock (RTC) powered by an external battery
- Power button

3.2. Design Decision: Mobile Application Language/Framework

The Water Baddies App must be a cross-platform application that works seamlessly on Android and iOS. Both systems use their own coding languages, Kotlin and Swift, respectively, which result in development time being doubled if we decide to program the app in each system's native language. The team then researched cross-platform languages, which would allow us to program in one

language/framework, reducing rework. It was decided to use Flutter, a cross-platform framework built by Google and programmed in Dart. Flutter allows users to create components and widgets that can be switched based on the app's state. The stateful nature of the widgets in Flutter allows us to build reactive apps, increasing the end-user experience.

3.3. Design Decision: Integrated System GUI

In keeping with the majority autonomous nature of the Water Baddies Detection System, we aim to require minimal user input. However, some cases may arise where redundancy in our systems is necessary. In these cases, it is beneficial for the detection system to have an integrated GUI with 4 primary buttons. The interface will be located on the outer face of the system and will display the current state of the system, the previous state of the system, the battery percentage, and the Bluetooth status. The primary purpose of the buttons in the display will be to reset the Bluetooth, start and stop the detection sequence, and turn the system on and off. Beyond that, the integrated GUI's primary purpose is the redundancy of information display and to serve as the main controls on the system.

3.4. Design Decision: Component-Based Architecture

When designing the software, there were many different approaches we could have taken with the architecture. Considering the team is using a Raspberry Pi, it's simplest to implement as much embedded code as possible with Python. Python is also an incredibly easy language to utilize and learn, specifically for members of the team who have never programmed before. With the team needing so many different functionalities spread across so many different disciplines (ie, Mechanical, Optical, Software), the team felt it was best to adopt a component-based architecture. Component-Based Architecture will allow the team to work on individual functions of the software in siloed development spaces. Each component will serve one major objective, such as moving the conveyor belt motor, taking and processing images from the microscope, etc. We can test these components individually to ensure compliance with our requirements and needs and then create a wrapper to call the individual component functions when needed. Overall, the design provides us with modular software that can be more easily developed by the team as the needs of our project become more defined or change over time.

4.0 Architectural Design

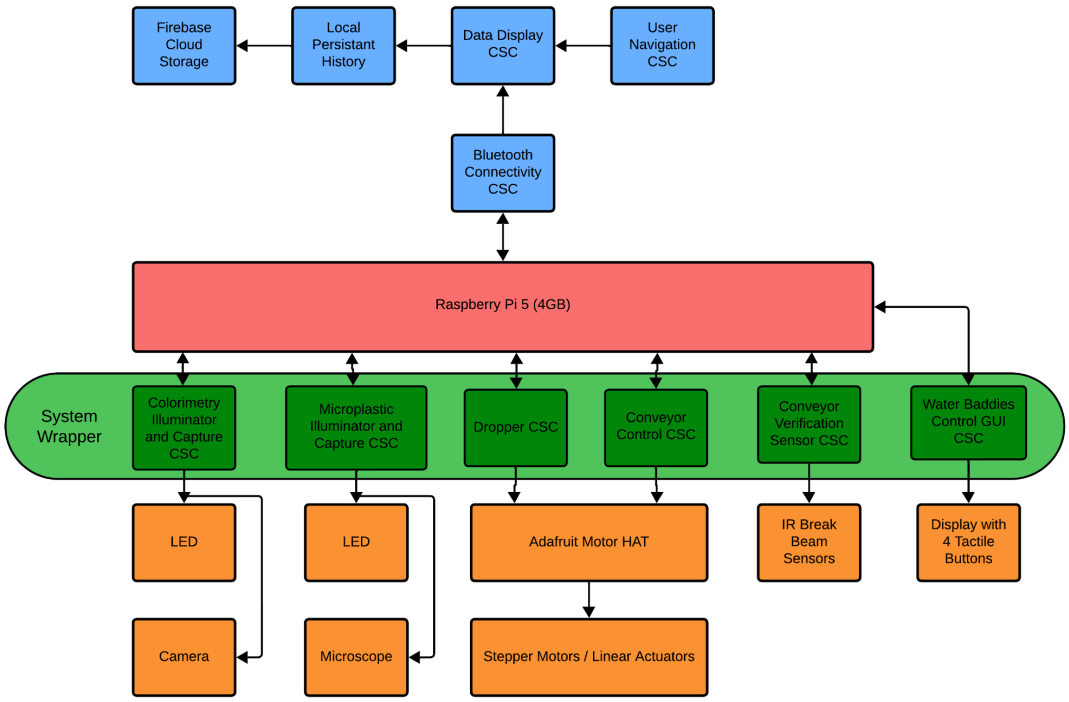


Figure 4.1: Software Architecture Diagram

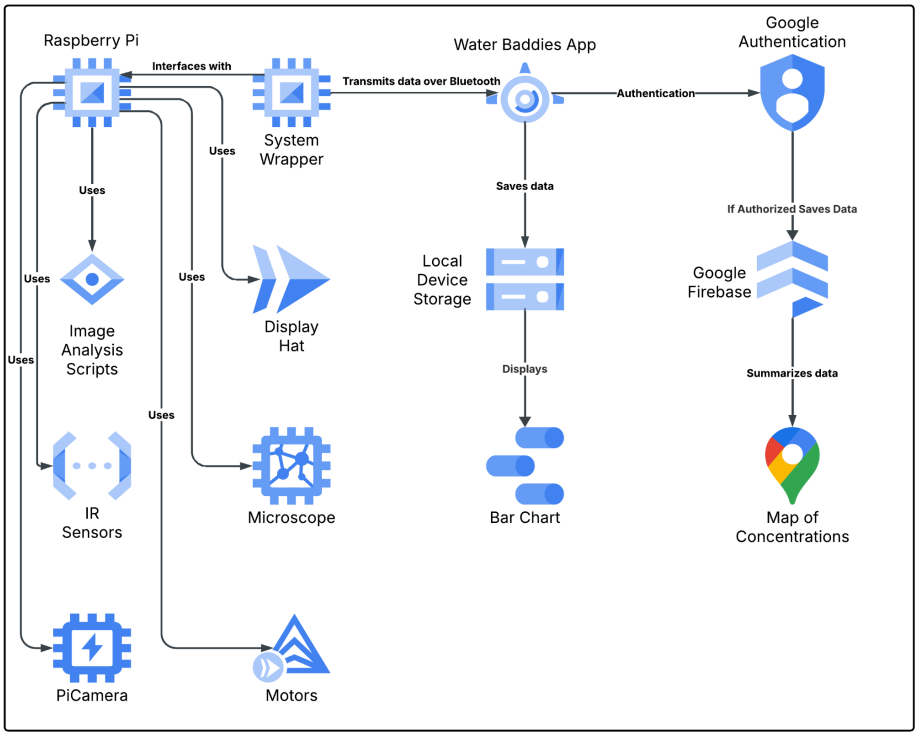


Figure 4.2: Software Interface Flow Diagram

4.1. Overall Architecture

As discussed in Section 3.4, the software for the Water Baddies Detection System is designed using component-based architecture. Figure 1.1 shows all of the software components, grouped by color, and their interfaces and flow of data. The software for the Water Baddies System is split into two distinct groups.

The first is the group in blue: the Bluetooth Connectivity CSC, Data Display CSC, Local Persistent History, and User Navigation CSC, which comprise the Water Baddies App. The goal of the app is to provide users with a seamless and modern interface to view the data coming from the detection system. The app also serves as a source of information on water baddies and documentation for the entire project. That data received from the Raspberry Pi on the mobile application is then sent to a secure Google Firebase Cloud database and used for big data processing and mapping of water quality over an area.. The mobile application receives data from the Pi over Bluetooth, the logistics of which will be discussed more in section 5. The concentration information collected from the Pi is associated with the MVP2 capability.

The second major component is the hardware inside the Water Baddies Detection System, including the Raspberry Pi, colored salmon, and the Pi Camera Module 3, LEDs, Microscope, Adafruit Motor HAT, Stepper Motors, Linear Actuators, IR Break Beam Sensors, and Display, all colored in orange. While the diagram makes it seem like the CSCs in green are directly communicating between the Pi and the hardware, the software is embedded and running on the Pi, which has a direct physical connection with the rest of the hardware.

The final major section of components are the computer software components run on the Pi located inside the Water Baddies Detection System, marked in green. These software components are responsible for running all of the hardware and calculations responsible for collecting data, processing concentrations and images, and communicating between subsystems and the mobile app. Each CSC, which will be discussed in more depth in Section 5, is responsible for one major task and, for orchestration purposes, is wrapped in a system wrapper. The system wrapper will be activated when the user starts the Water Baddies Detection System and will control the flow and execution of the Bluetooth service, the embedded GUI, and each individual CSC, providing periodic updates back to the user on the state of the system. All of these components work coherently to perform all vital functions for the Water Baddies Detection System and ensure the system meets its requirements.

5.0 File Architecture

5.1. Water Baddies App

The Water Baddies App is broken down into 14 files and 8 folders. The purpose of breaking the app into multiple files is to allow maintainers and readers an easier experience digesting the many functions of the app. Each file attempts to contain only one major element or process, but results vary depending on the page selected. The base directory lib, is broken down into 4 parts:

---

* Any name ending in "/" is a directory or folder

- lib/
  - main.dart
  - firebase_options.dart
  - utils/
  - screens/

Main.dart is the central application for the water baddies app. Here, the app is initialized, basic states are created for timers, Bluetooth listeners are created, and other booleans are set. The database connection with Firebase, a Google Cloud service, is also initialized for the application. The main.dart then goes on to set up the initial page infrastructure, including the drawer side bar, nav page, title bar, and page area. A change notifier is also created, which allows access to commonly used functions and variables that will need to be used across classes.

Firebase_options.dart is a computer-generated Flutter initializer file for the Firebase database. The options file includes the configurations for integrating firebase to an android app, and my api keys and identifiers for my specific firebase database. We should note that in the Firebase Cloud application, security rules have been set up to require authentication and authorization for pushing new data to the database and for viewing the data inside.

The utils folder contains commonly used functions and data used across the entire app. Inside the utils folder is a file called Utils.dart. Specifically, this file contains a Google sign-in function that is used on multiple pages, constants for EPA-allowed values for the water baddies, and a custom Boolean class that allows for easier handling of booleans across pages and reloads.

The screens folder is where all of the dart files for each screen and the components of each screen are located. By screens, we are referring to the different pages accessed from the navigation bar setup in main.dart. The screens folder structure is as follows:

---

* Any name ending in "/" is a directory or folder

- screens/
  - about/
    - about.dart
  - analytics/
    - analyticsPage.dart
  - Bluetooth/
    - bluteoothBar.dart
  - history/
    - history.dart

- info/
  - baddiesInfo.dart
  - barChart.dart
  - heavyMetalsInfo.dart
  - infoCard.dart
  - infoUtils.dart
  - inorganicsInfo.dart
  - microplasticsInfo.dart

All of these files, and more specifically, the classes inside of them, are included in the detailed design section below. The following section will be an overview of each.

Inside the About section is about.dart, which contains a basic FAQ, directions on how to use the water baddies system, and information about the team.

Inside analytics in analytics.dart. This page is responsible for authenticating the user and checking their authorization for the Firebase database. If the user is authorized, the page will display a summary of all of the data in the cloud. The summary includes but is not limited to average concentrations, percentages of healthy water, and a map showing the location of each water trial and a green or red marker depending on if the water was deemed healthy to drink or not. The process of authentication and display is discussed more in the Analytics CSC section.

Inside Bluetooth is the BluetoothBar.dart file, which created the drawer side bar I discussed earlier. The Bluetooth bar contains the general structure for the Bluetooth bar listing out available devices, if the device is scanning, if the device is connected, and buttons to disconnect from the device. Used within the drawer are utility functions that handle the connection, disconnect, search, and status updates for the Bluetooth connection. This process is discussed more in the Bluetooth CSC section.

The history folder contains history.dart, which is simply a display module to list out each entry of the water baddies system. The history section only stores local history, so any data that was fetched on the current device from the Water Baddies System and not from the cloud, like the analytics page. In-depth details are discussed more in the Data Display CSC section.

Finally, in the info directory are all of the components and files that allow the user to view their current water results from the Water Baddies System, along with information about each water baddie and why it's important we are able to detect them. The baddiesInfo.dart is the main file that orchestrates the calling of each of the components and contains all of the utility functions used when fetching data, saving data offline and uploading it when online, and providing the user with error messages. Inside its build function contains the logic for partial rebuilds for specific status related graphics, and calls to the other components. The barChart.dart contains two functions used in creating the bar chart that is shown on the info page, which tells the user how many of each water baddie was in their water and what the safe amount to drink is. The heavyMetalsInfo.dart, inorganicsInfo.dart, and microplasticsInfo.dart are a collection of text and padding fields that provide the user with information on each baddies. The infoCard is the wrapper for each water baddie, including Microplastics, Heavy Metals, and Inorganics. The purpose of the info card is to be a reusable element so that more water baddies can be included in the future. Lastly, the infoUtils.dart includes three functions to reduce lines of code when building sections for text and create consistency across the app.

5.2. Water Baddies Pi

The Water Baddies Pi contains the embedded code on the Raspberry Pi 5 inside of the Water Baddies Detection System. The code is broken down into different files containing classes and functions that are then imported into the systemWrapper.py. The file structure is as follows:

---

* Any name ending in "/" is a directory or folder

- BluetoothCreation/
  - tools/
    - advertisement.py
    - bletools.py
    - service.py
  - baddiesDetection.py
- logs/
- paperfluidicsImages/
- plasticImages/
- test_images/
- tests/
  - test_analysis.py
  - test_breakpoint.py
  - test_display.py
  - test_motor_breakpoint.py
  - test_motors.py
  - test_picamera.py
- wbe/
- breakpointSensor.py
- DisplayHat.py
- microscope_analysis.py
- paperfluidic_analysis.py
- systemWrapper.py

BluetoothCreation contains all of the files related to creating a Bluetooth GATT service on the Raspberry Pi. This includes creating characteristics, descriptors, and advertisements for the Bluetooth. Inside the tools folder of BluetoothCreation are classes for a Bluetooth advertisement, which tells other devices that the Raspberry Pi is able to be connected to, and a service class, which is a wrapper for the characteristics and descriptor classes. Bletools.py contains basic functions for interacting with the dbus and Raspberry Pi hardware. Finally, inside BluetoothCreation is baddiesDetection.py, which combines all of the tools into the specific GATT Bluetooth service for the Water Baddies Detection System. This creates specific characteristics for microplastics, lead, cadmium, etc, and creates a BaddiesDetectionService and BaddiesAdvertisement class that the system wrapper utilizes to control the Bluetooth on the Raspberry Pi for our specific GATT service.

The logs folder contains all of the logs while running the system and are used for debugging and troubleshooting errors.

The paperfluidicImages and plasticImages are where images from the piCamera and microscope are saved during the detection sequences. Meanwhile, the test_images folder stores known quantity images to ensure our analysis scripts work.

The tests folder contains pytests that can be run to ensure that the software is still working and the hardware is connected correctly. Test_analysis.py contains two functions to test the paperfluidic analysis and the microplastic analysis algorithms. Test_breakpoint.py contains a simple hardware to test to ensure the breakpoint sensors are wired correctly and connected to the correct pins. Test_motor_breakpoint.py is an evolution of the breakpoint test to test that each breakpoint sensor is correctly hit and that once hit they trigger the correct motor. Test_motors.py is used to spin each motor to make sure they are wired correctly. Finally, test_picamera.py ensures that the picamera is wired correctly and can take and save a photo. All of these tests combined ensure extensive coverage of hardware and software integration.

Wbe is the virtual environment for the Raspberry Pi. This ensures that if the Raspberry Pi breaks or needs to be reset, all of the required packages can be easily transplanted along with the code to a new code base for easy spin up for the system.

BreakpointSensor.py creates an IRSensor class and a simple function called isObjectDetected(), which allows the systemWrapper to easily create new breakpoint sensors and check if they detected an object, the object being the slides where water was dispensed on.

DisplayHat.py creates a display object that contains all of the logic for the DisplayHatMini to work. The DisplayHat object creates a messaging queue that allows for the system wrapper to write messages to the display safely without overloading it. The class also handles button clicks, changes in the display, and the timer for the current batch.

Microscope_analysis.py and plastic_analysis.py contain the two image processing algorithms for the detection system that are responsible for producing an accurate and quantifiable number of particles in each water sample.

Finally is the systemWrapper.py, the overarching orchestrator for the entire Water Baddies Detection System. This script contains a class called SystemWrapper, which initializes every class used in the system, threads for button listeners, starts the Bluetooth service, and creates functions and the logic for each detection sequence to ultimately make the Water Baddies Detection System special.

6.0 Detailed Design

6.1. Bluetooth Connectivity CSC

An embedded CSC used in the Water Baddies App to facilitate the detection of available Bluetooth low-energy devices, the connection handshake between the Raspberry Pi and the app, and data transfer from the Pi to the App over Bluetooth. The CSC is split between *BluetoothBar.dart* and the class *WaterBaddiesState* within *main.dart.*

WaterBaddiesState:

For purposes of simplistic data transfer with the rest of the app, and the proper state management the WaterBaddiesState holds the device the app is currently connected to, the concentration values received from the Pi, the list of characteristics found from the Bluetooth service, the subscriptions to listen for new concentrations, a function to initially fetch the Bluetooth characteristics (which contain

the concentration information) upon connection of a new device, and utility functions to support the variables. Having this information stored in the WaterBaddiesState and not the BluetoothBar drawer allows for data persistence once the side drawer is closed and allows for other states (pages) within the app to update upon a change in one of the WaterBaddiesState variables.

```dart
class WaterBaddiesState extends ChangeNotifier {
  BluetoothDevice? _device;
  String changeKey = "";
  String _connectionMessage = "Please Connect a Bluetooth Device";
  bool newDataAvailable = false;
  StreamSubscription<BluetoothConnectionState>? _connectionSub;

  Map<String, double> _characteristicsData = {};
  Map<BluetoothCharacteristic, StreamSubscription<List<int>>> subscriptions = {};
  List<BluetoothCharacteristic> characteristics = [];
  List<dynamic> readingSubs = [];

  BluetoothDevice? get device => _device;

  set device(BluetoothDevice? newDevice) {
    _device = newDevice;
    if (newDevice != null) { // Only fetch characteristics if newDevice is not null
      createConnectionSubscription();
      fetchCharacteristics(_device!);
      startFetchingCharacteristics();
    }
    notifyListeners();  // Notify listeners when the device is updated
  }

  StreamSubscription<BluetoothConnectionState>? get connectionSub => _connectionSub;

  set connectionSub( StreamSubscription<BluetoothConnectionState>? newSub) {
    _connectionSub = newSub;
  }

  String get connectionMessage => _connectionMessage;

  set connectionMessage(String? newMessage) {
    // Check if the newMessage is null, if so, assign a default value.
    _connectionMessage = newMessage ?? "Please Connect a Bluetooth Device";
  }

  Map<String, double> get characteristicsData => Map.from(_characteristicsData);
```

Figure 5.1: WaterBaddiesState device functions

The fetchCharacteristics() function is responsible for reading the available characteristics over Bluetooth from the Raspberry Pi and storing them for easy access later. The function first detects all available services and loops through them to find the specific characteristics we are looking for. Fetching characteristics is only done once because the characteristics don't change in the Bluetooth service, so fetching new data becomes faster with the saved characteristics.

```dart
class WaterBaddiesState extends ChangeNotifier {
  void fetchCharacteristics(BluetoothDevice device) async {
    try {
      final services = await device.discoverServices();

      final service = services.where((s) => s.uuid.toString() == targetServiceUuid).firstOrNull;

      if (service != null) {
        for (final characteristic in service.characteristics) {
          if (targetCharacteristics.contains(characteristic.uuid.toString())) {
            characteristics.add(characteristic);
          }
        }
      }
    } catch (e) {
      print("Error discovering services: $e");
    }
  }
```

Figure 5.2: WaterBaddiesState fetchCharacteristics() logic to loop through characteristics

Each characteristic pertains to one “water baddies”: lead, cadmium, nitrate, nitrite, and microplastics. Each characteristic contains the value of concentration for the associated “baddie” and also contains a descriptor, allowing us to categorize each concentration. The data is fetched using the fetchNewData function, which is called every 3 seconds. Inside of the function, all of the characteristics saved earlier are looped through, and their value is saved and decoded:

```dart
void fetchNewData() async {
  for (final characteristic in characteristics) {
    try{
      final charValue = await characteristic.read();
      final charValueString = String.fromCharCodes(charValue);
      final charValueDouble = double.tryParse(charValueString) ?? 0.0;

      for (final descriptor in characteristic.descriptors) {
        if (descriptor.uuid.toString().toUpperCase() == "2901") {
          final descValue = await descriptor.read();
          if (descValue.isNotEmpty && !descValue.every((v) => v == 0)) {
            final descString = String.fromCharCodes(descValue);
            _characteristicsData[descString] = charValueDouble;
          }
        }
      }
    } catch(e){
      print("Error reading characteristic: $e");
    }
  }
}
```

Figure 5.3: fetchNewData() logic to read Bluetooth value from characteristic and store it

One of the characteristics in the GATT service is a changeKey. This changeKey is updated by the Raspberry Pi at the end of a detection sequence to tell listeners that the values have been changed. In our program, we run the fetchNewData function every 3 seconds to ensure the most updated values at all times. To ensure the values being read are new, the didKeyChange function is used to find the broadcasted key and compare it to our stored key. If the key is changed, the fetchNewData function is called, and the newDataAvailable variable is set to true. In the WaterBaddiesInfoState there is a listener for that boolean which will update the display, telling the user new data is available:

```
void startFetchingData() {
  _fetchTimer?.cancel(); // Cancel any existing timer
  _fetchTimer = Timer.periodic(Duration(seconds: 3), (timer) {
    didKeyChange();
    if (newDataAvailable) {
      fetchNewData();
    }
    notifyListeners();
```

Figure 5.4: Timer to check if new data is available

```
void didKeyChange() async {
  for (final characteristic in characteristics) {
    if (characteristic.uuid.toString() == "0000000
      try{
        final charValue = await characteristic.rea
        final charValueString = String.fromCharCod
        if (changeKey != charValueString) {
          newDataAvailable = true;
          changeKey = charValueString;
        }
```

Figure 5.4: Check for key change

Finally, the WaterBaddiesState creates a listener on the connection of the device. The subscription serves to update the message displayed and notify the user if the connection to the Baddies Detection System drops for an unknown reason:

```
void createConnectionSubscription() {
  if (device != null) {
    connectionSub = device!.connectionState.listen((BluetoothConnectionState state) {
      if (state == BluetoothConnectionState.disconnected) {
        connectionMessage = "Please Connect a Bluetooth Device";
        device = null;
        notifyListeners();
      } else if (state == BluetoothConnectionState.connected) {
        connectionMessage = "Connected";
        notifyListeners();
      }
    });
  } else {
    connectionMessage = "Please Connect a Bluetooth Device";
  }
}
```

Figure 5.5: Connection subscription

BluetoothBar.dart

The BluetoothBar is a widget containing a suite of objects and functions to facilitate the connections between the Raspberry Pi and the Water Baddies App using Bluetooth-Low-Energy. The visible portion of the widget is a drawer or side panel that slides into the screen and displays the current connection status, connected devices, previously connected devices, available devices, and methods to connect and disconnect from a device. The separation between the BluetoothBar and WaterBaddiesState is necessary because once the BluetoothBar drawer is closed, which occurs after the user taps somewhere on the screen that's not the bar, all of the data and active running processes are destroyed from memory. This means that if we stored the device information or characteristics in the BluetoothBar, the data would be inaccessible when the bar is closed. Separating the CSC into two classes also serves to provide more separation of concerns and have the BluetoothBar only contain functions necessary to the connection of devices.

When the user opens up the BluetoothBar, the program attempts to fetch any currently connected device from the *WaterBaddiesState*, since the variables in the WaterBaddiesState are persistent. If the app is connected to a device, it sets the private variable "_device" to the device, and then creates a listener for the status of the connection. The *BluetoothBar* will then create other subscriptions to fetch scan results of available Bluetooth devices and statuses. Finally, the bar will load any device the user has previously connected to and begin scanning for devices:

```dart
class _BluetoothBarState extends State<BluetoothBar> {
  void initState() {
    super.initState();

    getConnectedDevice();

    _scanResultsSubscription = FlutterBluePlus.scanResults.listen(onData:
    (List<ScanResult> results) {···

    _isScanningSubscription = FlutterBluePlus.isScanning.listen(onData:
    (bool state) {···

    _statusMessageSubscription = FlutterBluePlus.adapterState.listen(onData:
    (BluetoothAdapterState state) {···

    loadPreviouslyConnectedDevices();

    scanDevices();
  }
```

Figure 5.6: Subscriptions initialized upon opening the Bluetooth sidebar

The BluetoothBar creates a button to start and stop scanning for available Bluetooth Low Energy Devices and creates clickable cards for each device. When the card is clicked, the BluetoothBar will attempt to connect to that device, and once connected, the device information will be saved on the local storage device. Once connected, the device will appear at the top of the BluetoothBar, and the user will have the option to disconnect from the device. Once connected, the user can click back to the main screen of the Water Baddies App and begin fetching data if connected to the Pi.

```dart
IconButton(
  icon: Icon(icon: _isScanning ? Icons.stop : Icons.
  start,
  color: _isScanning ? Colors.red : Colors.white),
  onPressed: _isScanning ? stopScan : scanDevices,
), // IconButton
```

Figure 5.7: Button to start and stop scanning of Bluetooth devices

```
return Card(
  elevation: 2,
  child: ListTile(
    title: Text(data: data.device.platformName),
    subtitle: Text(data: data.device.remoteId.str),
    trailing: Text(data: data.rssi.toString()),
    onTap: () {
      scrollCont.jumpTo(value: 0.0);
      connectDevice(device: data.device);
    },
  ), // ListTile
); // Card
```

Figure 5.8: Card that displays available Bluetooth device information and connection function

6.2. Data Display CSC

The data display is the main page of the Water Baddies App. This page directs the user to connect to the Pi, when to fetch new data from the Pi, contains information on all of the water baddies, and has dynamically updatable bar charts showing the concentration of water baddies from the most recent batch of the Water Baddies Detection system. The Data Display CSC consists of many classes and states, mainly the *WaterBaddiesInfoState,* which is composed of multiple instances of the *InfoCard* class, which in turn has an instance of the *barChart* class. The data display CSC also contains the *History, Analytics,* and *About* classes.

WaterBaddiesInfoState

The class contains a suite of functions and variables that continuously listen for changes in the incoming data from the Raspberry Pi and display when new data is available, allow for past data to be displayed in the bar chart, store data from the raspberry Pi in the local storage of the phone whenever new data is fetched, and serves as a scaffold for the information on the page. Within the scaffold, there is a section listening for updates from the *WaterBaddiesState,* and when the characteristic information changes, the display updates to notify the user that new data is available. The state will update and give the user the option to fetch the new data. Once the user selects to fetch the data, it is stored in the history, saved to the cloud, and the variable that feeds the *InfoCard* is updated. There is also an option for the user to upload data that may have been saved to the localHistory but was unable to be saved to the cloud at that time.

```
Selector<WaterBaddiesState, bool>(
  selector: (context, state) => state.newDataAvailable && state.characteristicsData.isNotEmpty,
  builder: (context, hasNewData, child) {
    if (hasNewData) {
      return Column(children: [
        Padding( // Padding ···
        ElevatedButton(
          onPressed: () {
            _updateDisplayedData(context);
          },
          child: Text("Fetch New Data"),
        ), // ElevatedButton
      ]); // Column
    } else {···
    }
  }
), // Selector
Selector<WaterBaddiesState, bool>(
  selector: (context, state) => internetConnected.value && offlineData.isNotEmpty,
  builder: (context, internetConnectedAndData, child) {
    if (internetConnectedAndData) {
      if (offlineData.isNotEmpty) {
        return TextButton(
          child: const Text("Upload Data"),
          onPressed: () async {
            await _uploadOfflineData(); // Upload offline data
```

Figure 5.9: Section that updated to show new data or offline data available to save

When new data is "fetched," it has actually already been fetched from the Bluetooth service and saved to the device. The function below, updateDisplayedData creates a copy of the concentration information, adds it to history (both local and cloud), then shows a warning dialog if any of the water is unhealthy to drink, and finally updates the state variable to tell the app that no new data is available.

```
void _updateDisplayedData(BuildContext context) {
  final newData = wbState.characteristicsData;
  wbState.newDataAvailable = true;
  addHistory(newData);
  setState(() {
    List<String> warningMessages = _checkData(newData);
    if (warningMessages.isNotEmpty) {
      _showWarningDialog(context, warningMessages);
      Vibration.vibrate(pattern: [500, 1000, 500, 2000]);
      String fullMessage = warningMessages.join(". ");
      _speak(fullMessage);
    }
    _displayedData = Map.from(newData);
    wbState.newDataAvailable = false;
  });
}
```

Figure 5.10: Function to save new data and update the data being displayed

When new data is saved to the history, it is saved on the local storage of the device as well as the cloud storage available to the application. The local store is a simple key-value string that is used to populate the history page. Similarly, the cloud storage is also a key-value database, but the authorization of a user must first be checked, and then the data must be saved to the cloud. If the user is unable to be saved to the cloud, it will be stored alongside history in a variable called offlineData, which can be uploaded at a later time:

```
Future<void> addHistory(Map<String, double?> newData) async {
  try {
    final prefs = await SharedPreferences.getInstance();
    List<Map<String, dynamic>> historyInfo = [];

    // Retrieve any previously saved data
    if (prefs.containsKey('history')) {
      String? savedData = prefs.getString('history');
      if (savedData != null) {
        historyInfo = List<Map<String, dynamic>>.from(jsonDecode(savedData));
      }
    }

    Map<String, dynamic> newEntry = {
      "Date": DateFormat('yyyy-MM-dd HH:mm:ss').format(DateTime.now()).toString(),
    };

    if (newData.containsKey("Lead") && newData["Lead"] != null) {
      newEntry["Lead"] = newData["Lead"];
    }
    if (newData.containsKey("Cadmium") && newData["Cadmium"] != null) {
      newEntry["Cadmium"] = newData["Cadmium"];
    }
    if (newData.containsKey("Mercury") && newData["Mercury"] != null) {
      newEntry["Mercury"] = newData["Mercury"];
    }
    if (newData.containsKey("Phosphate") && newData["Phosphate"] != null) {
      newEntry["Phosphate"] = newData["Phosphate"];
    }
    if (newData.containsKey("Nitrate") && newData["Nitrate"] != null) {
      newEntry["Nitrate"] = newData["Nitrate"];
    }
    if (newData.containsKey("Microplastic") && newData["Microplastic"] != null) {
      newEntry["Microplastic"] = newData["Microplastic"];
    }

    newEntry["Location"] = await _getLocation();
    newEntry["Healthy"] = _getHealthy(newData.cast<String, double>());

    historyInfo.add(newEntry);

    await prefs.setString('history', jsonEncode(historyInfo));
```

Figure 5.11: New data being added to local storage history

```dart
class _WaterBaddiesInfoState extends State<WaterBaddiesInfo> {
  Future<void> addHistory(Map<String, double?> newData) async {
    // Check for internet connectivity
    var connectivityResult = await (Connectivity().checkConnectivity());
    if (connectivityResult[0] == ConnectivityResult.none) {
      // No internet connection, store data locally
      if (prefs.containsKey('offline_history')) {
        String? offlineDataString = prefs.getString('offline_history');
        if (offlineDataString != null) {
          offlineData = List<Map<String, dynamic>>.from(jsonDecode(offlineDataString));
        }
      }
      offlineData.add(newEntry);
      await prefs.setString('offline_history', jsonEncode(offlineData));
      if (mounted) {
        _showNoInternetPopup(context);
      }
    } else {
      // Internet connection available, upload data to Firebase
      try {
        UserCredential userCredential = await signInWithGoogle(); // Authenticate
        if (userCredential.user != null) {
          await FirebaseFirestore.instance.collection('history').add(newEntry);
          await _uploadOfflineData(); // Upload any previously stored offline data
        } else {
          throw Error();
        }
      } catch (e) {
        if (prefs.containsKey('offline_history')) {
          String? offlineDataString = prefs.getString('offline_history');
          if (offlineDataString != null) {
            offlineData = List<Map<String, dynamic>>.from(jsonDecode(offlineDataString));
          }
        }
        offlineData.add(newEntry);
        await prefs.setString('offline_history', jsonEncode(offlineData));
        if (mounted) {
          _showCloudError(context);
        }
```

Figure 5.12: Data being saved to the cloud or offline history

InfoCard:

The class is a custom template class meant to minimize the use of repetitive code structures when making the expandable cards for each of the “baddies”. Each card has a key that listens for updates to the displayedData dictionary and updates their respective cards, specifically the bar chart within, whenever the data is changed. The InfoCard takes two booleans, showChart and showInfo, which allow for the state of the card (expanded or not expanded) to be kept even when the infoCard is rebuilt upon new data changes. The card also takes in barChartData, which is the actual concentration for each “baddie”.

```
InfoCard(
  key: ValueKey("Inorganics${_displayedData["Nitrite"]}${_displayedData["Nitrate"]}"),
  showChart: showInorganicsChart,
  showInfo: showInorganicsInfo,
  cardTitle: "Inorganics",
  barChartData: _displayedData.isEmpty
      ? []
      : [
          if (_displayedData.containsKey("Nitrite"))
            {
              'name': 'Nitrites',
              'maxQuantity': maxQuantities['Nitrite'],
              'quantity': _displayedData["Nitrite"],
            },
          if (_displayedData.containsKey("Nitrate"))
            {
              'name': 'Nitrates',
              'maxQuantity': maxQuantities['Nitrate'],
              'quantity': _displayedData["Nitrate"],
            },
        ].where((element) => element.isNotEmpty).toList(),
), // InfoCard
```

Figure 5.13: Info Card for Inorganics

Each info card is broken down to show information about heavy metals, inorganics, and microplastics. These classes encompass stateful widgets, meaning the state of the app changes as the user interacts with it. They are indicated as such within their own class so they can be individually customized. They have a string with initial information and a photo and may have additional buttons for the sub-section drop-down tiles.

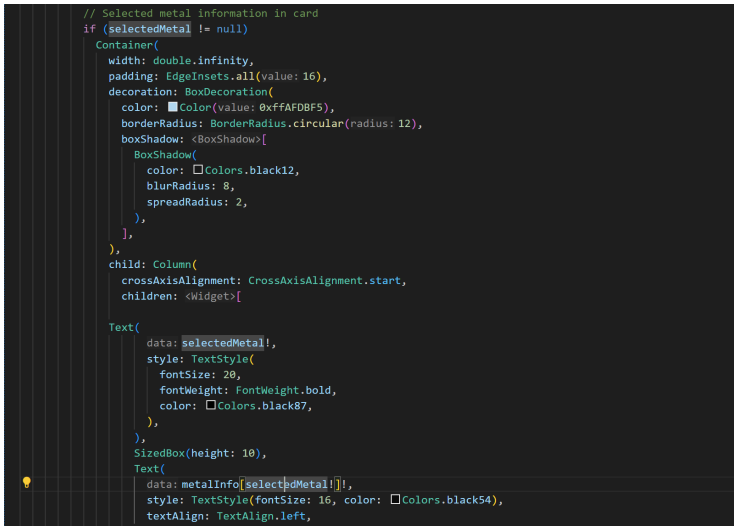
```dart
class HeavyMetalsInfo extends StatefulWidget {
  @override
  // ignore: library_private_types_in_public_api
  _HeavyMetalsInfoState createState() => _HeavyMetalsInfoState();
}

class _HeavyMetalsInfoState extends State<HeavyMetalsInfo> {
  String? selectedMetal;

  final Map<String, String> metalInfo = <String, String>{
    'Lead': "EPA Standard: 5.0 mg/L \nLead exposure can cause neurological damage, especially
    'Cadmium': "EPA Standard: 1.0 mg/L\nCadmium exposure is linked to kidney damage and lung d
    'Mercury': "EPA Standard: 0.2 mg/L\nMercury exposure can affect the nervous system, especi
  };

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: EdgeInsets.symmetric(horizontal: 20, vertical: 20),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.center,
        children: <Widget>[
          // Image
          ClipOval(
            child: Image.asset(
              name: "images/HeavyMetals.jpg",
              width: 300,
              height: 300,
              fit: BoxFit.cover,
            ), // Image.asset
          ), // ClipOval
          SizedBox(height: 10),
```

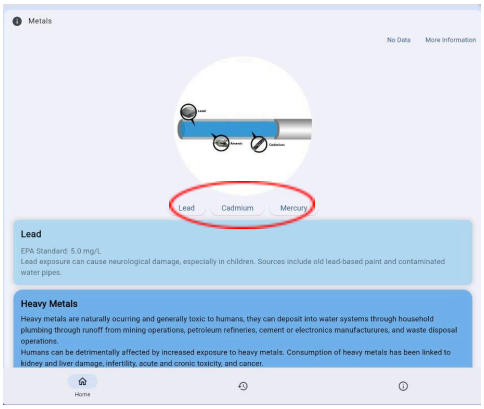
Figure 5.14: HeavyMetalsInfo class with string with initial information and widget display customizations.



```
// Selected metal information in card
if (selectedMetal != null)
  Container(
    width: double.infinity,
    padding: EdgeInsets.all(value: 16),
    decoration: BoxDecoration(
      color: Color(value: 0xffAFDBF5),
      borderRadius: BorderRadius.circular(radius: 12),
      boxShadow: <BoxShadow>[
        BoxShadow(
          color: Colors.black12,
          blurRadius: 8,
          spreadRadius: 2,
        ),
      ],
    ),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: <Widget>[

    Text(
          data: selectedMetal!,
          style: TextStyle(
            fontSize: 20,
            fontWeight: FontWeight.bold,
            color: Colors.black87,
          ),
        ),
        SizedBox(height: 10),
        Text(
          data: metalInfo[selectedMetal!]!,
          style: TextStyle(fontSize: 16, color: Colors.black54),
          textAlign: TextAlign.left,
```

```
// Always Visible Buttons
Wrap(
  spacing: 10.0, // Space between buttons
  children: metalInfo.keys.map<ElevatedButton>(toElement: (String metal) {
    return ElevatedButton(
      onPressed: () {
        setState(fn: () {
          selectedMetal = (selectedMetal == metal) ? null : metal;
        });
      },
      child: Text(
        data: metal,
        style: TextStyle(fontSize: 16), // Set font size here
      ),
    );
  }).toList(),
),
SizedBox(height: 10),
```




```
class _HeavyMetalsInfoState extends State<HeavyMetalsInfo> {
  String? selectedMetal;

  final Map<String, String> metalInfo = <String, String>{
    'Lead': "EPA Standard: 5.0 mg/L \nLead exposure can cause neurological damage, especially in children. Sources inc
    'Cadmium': "EPA Standard: 1.0 mg/L\nCadmium exposure is linked to kidney damage and lung disease. It is commonly d
    'Mercury': "EPA Standard: 0.2 mg/L\nMercury exposure can affect the nervous system, especially in unborn babies an
  };
```

Figure 5.15: Stateful widget interfaces

barChart:

The barChart is a class built to encompass the logic and utility functions for creating the bar chart visual representation of the concentration of each “baddies" relative to their respective EPA limit. The bar chart utilizes the barChart package from Syncfusion, and most of the code is slightly modified versions of their use case example. The main modifications are calculations for the max height of the bar chart and logic to draw a red line where the EPA threshold is.

```dart
if (widget.barChartData.isNotEmpty && widget.showChart.value)
  BarChart(
    key: ValueKey<List<Map<String, dynamic>>>(value: widget.barChartData),
    barData: widget.barChartData,
    showChart: widget.showChart,
), // BarChart
```

Figure 5.16: Creation of bar chart inside InfoCard

History:

The history.dart file and accompanying *historyState* are a separate page from the *WaterBaddiesInfoState* discussed above. The history page serves to display a list of past “water baddies" data collected from the raspberryPi and their associated datetime, individual baddies values, and the location of the user when the data was received. The class has a simple utility function, fetchHistory, which runs when the user opens the page. The fetchHistory function uses the SharedPreferences class, which is the Flutter class for small, local, and persistent data storage. When the page is initially loading, the fetchHistory function will grab the data stored in SharedPreferences under the key ‘history’ and parse the data, converting it to a list of maps. The list is fed to the build function, looping through the list and outputting expandable list tiles to display each entry's data.

```dart
Future<List<Map<String, dynamic>>> fetchHistory() async {
  final prefs = await SharedPreferences.getInstance();
  String? savedData = prefs.getString('history');

  if (savedData != null) {
    return List<Map<String, dynamic>>.from(jsonDecode(savedData));
  }
  return [];
}
```

Figure 5.17: History being fetched from local storage

In the build function, there is also a buildKeyValueRow function, which is a simple function that puts the key and its value next to each other in text. The function is used to minimize repetitive code when building each card.

One of the elements built into the card is a generated PDF function. The function takes whichever history entry you selected and creates and downloads a table containing the concentrations, maximum values, and an indicator of whether the water is drinkable:

```
Future<void> createPDF(Map<String, dynamic> data) async {
        pw.TableHelper.fromTextArray(
          headers: [
            'Contaminant',
            'Concentration (mg/L)',
            'Max Allowed (mg/L)',
            'Status'
          ],
          data: concentrations.entries.map((entry) {
            final epaLimit = epaLimits[entry.key] ?? 0.0;
            final bool hasValue = entry.value != null && entry.value != 0.0;
            final concentration =
                hasValue ? entry.value.toStringAsFixed(3) : 'No Value';
            final status = hasValue
                ? (entry.value > epaLimit ? 'Exceeded' : 'Safe')
                : 'Safe';
```

Figure 5.18: Creation of PDF table headers and values

Analytics:

The analytics page is located in the analyticsPage.dart file. The purpose of the page is to integrate the cloud into the Water Baddies App. Before accessing the page, the user is prompted to sign in with Google by entering in their Google account information. Once the user is authenticated, the application checks if the user has the allowed authorization to access the cloud. For enhanced security, authorization is checked on the client side, as well as through security rules set up on the cloud:

```
Future<UserCredential> signInWithGoogle() async {

  final GoogleSignInAccount? googleSignInAccount = await googleSignIn.signIn();
  final GoogleSignInAuthentication googleSignInAuthentication = await googleSignInAccount!.authentication;
  final OAuthCredential credential = GoogleAuthProvider.credential(
    accessToken: googleSignInAuthentication.accessToken,
    idToken: googleSignInAuthentication.idToken,
  );
  return await FirebaseAuth.instance.signInWithCredential(credential);
```

Figure 5.19: Sign in with Google function

Once the user is authorized, they can access the summary of data from the cloud. This includes a summary of the water quality, a percentage of healthy water, and a map showing all

of the tests done by the detection system, and a color indicating if the water is healthy or not:

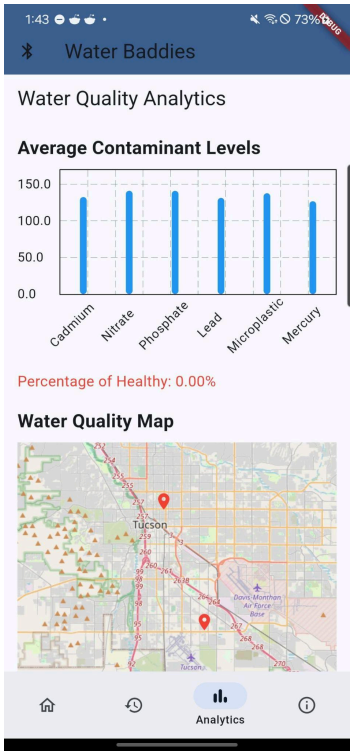


Figure 5.20: Analytics Page

About:

The About page is a simple text page containing information about the team, the project, instructions on how to use it, and an FAQ. Any logic in the About section is purely for UI or dropdowns and does not serve a greater complex purpose to the functionality of the app.

6.3. User Navigation CSC

With main operations in the *BaddiesHomePage*, the user navigation CSC builds the top and bottom bars of the screen. This contains the title of the app, the *BluetoothDrawer*, a button to open it, and buttons at the bottom of the screen to open each page. The class stores the currently selected index for the page selected on the bottom navigation bar and fills the body of the screen with that specific page. For instance, the history page is at index 1, so once the user clicks the history icon in the bottom navigation bar, the body of the scaffold is filled with the *History* class.

```dart
class _BaddiesHomePageState extends State<BaddiesHomePage> {
  int currentPageIndex = 0;

  @override
  Widget build(BuildContext context) {
    Widget page;
    switch (currentPageIndex) {
      case 0:
        page = WaterBaddiesInfo();
      case 1:
        page = History();
      case 2:
        page = AnalyticsPage();
      case 3:
        page = About();
      default:
        throw UnimplementedError('no widget for $currentPageIndex');
    }
```

Figure 5.19: Case statement to control pages

6.4. Local Persistent Storage CSC

As mentioned in section 5.2, every time the user fetches new data, it is automatically inserted into the local app storage of the Water Baddies App. The local storage is saved with the app data and is persistent across app sessions. The data can be viewed in the History section of the Water Baddies App. The CSC loops through all of the available histories, creating a collapsible tile for each, and lists the date of the data and any warnings about the data in the tile. Once the user clicks on the tile, it drops down, showing the levels of Lead, Cadmium, Phosphate, Nitrate, Nitrite, and Microplastics, as well as the coordinates of where the phone was when it fetched the data from the water baddies system.

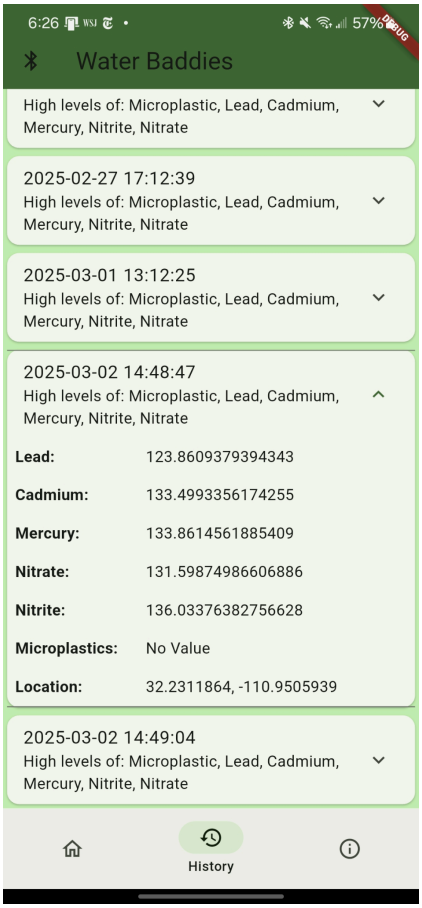


Figure 5.18: History Page

```
return ListView.builder(
  itemCount: data.length,
  itemBuilder: (BuildContext context, int index) {
    return Card(
      child: ExpansionTile(
        title: (data[index].containsKey(key: 'Date')
        ? Text(data: data[index]['Date'])
        : const Text(data: "No Date")),
        subtitle: Text(data: "High levels of: ${data[index]['Healthy'].join(', ')}"),
        children: <Widget>[
          _buildKeyValueRow(key: 'Lead', data: data[index]),
          _buildKeyValueRow(key: 'Cadmium', data: data[index]),
          _buildKeyValueRow(key: 'Mercury', data: data[index]),
          _buildKeyValueRow(key: 'Nitrate', data: data[index]),
          _buildKeyValueRow(key: 'Nitrite', data: data[index]),
          _buildKeyValueRow(key: 'Microplastic', data: data[index]),
          _buildKeyValueRow(key: 'Location', data: data[index]),
        ],
        ), // ExpansionTile
    ); // Card
  },
```

Figure 5.19: Builder for the history information

6.5. Colorimetry Capture CSC

The Water Baddies System Wrapper, discussed later, calls the main functions for the colorimetry capture CSC. When the paperfluidic is below the Pi camera, the wrapper calls captuerPiImage() to take an image of the paperfluidic device. This function uses the cv2 library to open a feed to the Pi camera and save an image to the Raspberry Pi, returning the image path so it can be used in the analysis script:

```
def capturePiImage(self):
    picam = Picamera2()
    picam.configure(picam.create_still_configuration()) # Add this line
    picam.start()
    picam.set_controls({"AfMode": controls.AfModeEnum.Continuous})
    path = f'paperFluidicImages/{datetime.now().strftime("%Y-%m-%d-%H-%M-%S.%f")[:-3]}.png'
    picam.capture_file(path)
    picam.close()
    return path
```

Figure 5.20: CpaturePiImage() function

This image path is then fed to the load_and_process_image() function to crop the image to the just the paperfluidic device:

```
try:
    imagePath = self.capturePiImage()
    print(f"First paperfluidics image: {imagePath}")
    print("Waiting for lead reaction")
    time.sleep(3)
    leadImagePath = self.capturePiImage() #Just capture lead image
    print(f"Lead paperfluidics image: {leadImagePath}")

    #Just for testing
    testImagePath = "./test_images/paperfluidic.jpg"
    testLeadImagePath = "./test_images/paperfluidic_test.jpg"
except Exception as e:
        print(f"Error during image capture: {e}")
        raise ImageCaptureError("Error during capturing image from the PiCamera for lead

try:
    #leadConcentration = paperfluidic_concentration(imagePath, leadImagePath)['lead']
    print("Starting Lead Concentration Analysis")
    vals = pfa.paperfluidic_concentration(testImagePath, testLeadImagePath)
    leadConcentration = vals['Lead']
```

Figure 5.21: Capture Image and Analysis Function Calls

```
def crop_image(image_path):
    image = io.imread(image_path)

    # Convert to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

    # Apply thresholding to detect the white region (paper)
    _, thresh = cv2.threshold(gray, 200, 255, cv2.THRESH_BINARY)

    # Find contours of the detected regions
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
                                   cv2.CHAIN_APPROX_SIMPLE)

    # Find the largest contour (assuming it is the paper)
    largest_contour = max(contours, key=cv2.contourArea)

    # Get the bounding box of the largest contour
    x, y, w, h = cv2.boundingRect(largest_contour)
    w = 2300

    cropped_image = image[y:y + h, x:x + w]
    return cropped_image
```

Figure 5.22: Colorimetric Preprocessing Functions

Finally, the paperfluidic_concentration() function creates a dictionary of heavy metals and inorganics and their corresponding concentrations, as well as establishes coordinates in the image for where each color change will be in the image. It then uses the average RGB value of each reservoir and finds a color value based on the NTSC luminance equation. The concentration is found by fitting the color value into the curves created for each metal and inorganic through testing.

```
def paperfluidic_concentration(input_image_path, region_radius=100):
    color_changes = {"Cadmium": 0, "Lead": 0, "Nitrate": 0,
                     "Nitrite": 0, "Phosphate": 0}
    key_list = list(color_changes.keys())
    reservoir_coords = [(1400, 1960), (1950, 1110),
                        (440, 660), (460, 1640), (1380, 340)]

    input_image = crop_image(input_image_path)

    for i, (x, y) in enumerate(reservoir_coords):
        # Extract small regions around the reservoir in both images
        region = input_image[y - region_radius:y + region_radius,
                             x - region_radius:x + region_radius]

        # Compute mean R, G, B values
        mean_r = np.mean(region[:, :, 0])
        mean_g = np.mean(region[:, :, 1])
        mean_b = np.mean(region[:, :, 2])

        color_value = 0.229*mean_r + 0.587*mean_g + 0.114*mean_b

        if i == 0:
            concentration = cadmium_concentration(color_value)
        elif i == 1:
            concentration = lead_concentration(color_value)
        elif i == 2:
            concentration = nitrate_concentration(color_value)
        elif i == 3:
            concentration = nitrite_concentration(color_value)
        else:
            concentration = phosphate_concentration(color_value)

        color_changes[key_list[i]] = concentration

    return color_changes
```

Figure 5.23: Calculation Function For Paperfluidics

6.6. Microplastic Illuminator and Capture CSC

The Water Baddies System Wrapper, discussed later, calls the main functions for the microplastic illuminator and capture CSC. When the optical slide is below the microscope, the wrapper calls the captureMicroscopeImage() function. This function uses the cv2 library to open a feed to the microscope and save an image to the Raspberry Pi, returning the image path so it can be used in the analysis script:

```
def captureMicroscopeImage(self):
    cap = cv2.VideoCapture(8)
    if not cap.isOpened():
        print("Error opening video stream or file")
        raise Exception("Couldnt open microscope stream")

    ret, frame = cap.read()

    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        raise Exception("Couldnt open microscope picture frame")

    path = f'plasticImages/{datetime.now().strftime("%Y-%m-%d-%H-%M-%S.%f")[:-3]}.png'
    cv2.imwrite(path, frame)

    return path
```

Figure 5.24: Microscope Capture Function

Using the SciKit library, a Python script does an image analysis of captured fluorescence images taken with the microscope to determine an estimated concentration in particles per milliliter. First, cropping is applied to the image to limit the area of analysis to the water on the reservoir of the microscope slide.

It then converts the image to grayscale and then applies a threshold filter to eliminate noise or glare and isolate the microplastic particles. It then counts the number of microplastic particles based on the thresholding and isolation and computes the approximate concentration using the 0.075 milliliter sample volume and a conversion factor determined experimentally to account for low concentrations and experimental noise. The following images show the code for the analysis as well as an example with the original fluorescence image along with the corresponding filtered image that is used for the calculation.

```python
def microplastic_concentration(image_path):
    image, radius = crop_image(image_path)
    image = color.rgb2gray(image)
    threshold = filters.threshold_yen(image)

    cleaned_image = morphology.closing(image >= threshold)

    # Label connected regions
    labeled_image = measure.label(cleaned_image, background=0)
    regions = measure.regionprops(labeled_image)

    # Count microplastic particles
    num_particles = len(regions)

    # Volume of each sample (in millimeters)
    volume = 0.075

    # Prevent falsely counting low-signal for particles
    if num_particles >= 10:
        num_particles = 0

    concentration = conversion_constant * num_particles / volume

    print(f'Particles counted: {num_particles} particles')
    print(f'Approximate concentration: {concentration:.2f} particles/mL')

    return concentration
```

Figure 5.25: Full microplastic image analysis script

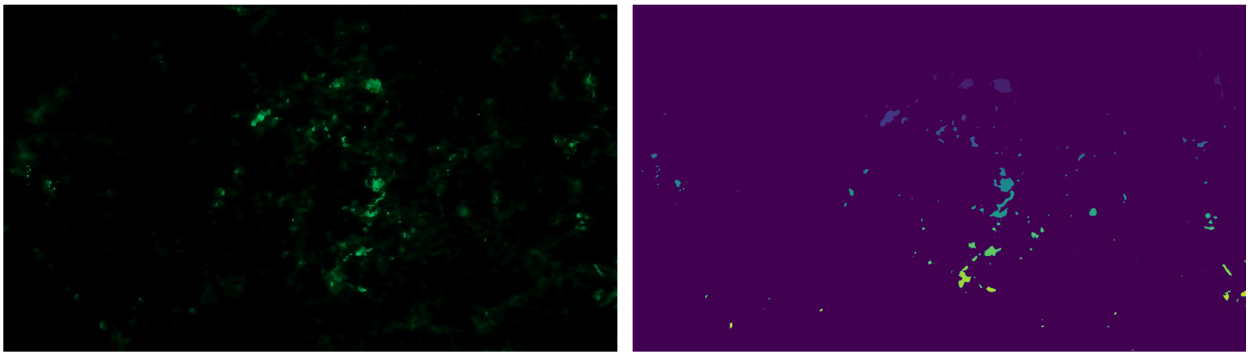


Figure 5.26: Comparison between a fluorescence image taken by a microscope (left) and the filtered image used for concentration calculations (right).

6.7. Dropper CSC

The dropper CSC works using the Adafruit motor kit library. The library allows us to create a kit and then move a specified stepper motor one step for any number of times it takes for the dropper motor to move the syringe enough. The dropper csc and the conveyor control csc use the same function, run_stepper(), to move their respective stepper motor. The dropper for microplastics is referenced as steppe12 and is run from the dispensePlasticWater() function. In this function we periodically check that there is enough water in the syringe to run a trial.

```
def dispensePlasticWater(self):
    self.display.updateQueue({"stage":"Dispensing water"})
    print("Dispensing water")
    canMove = self.plasticMotorIR.is_object_detected()
    for i in range(1):
        if (canMove):
            self.run_stepper(self.kit2.stepper1, 4, stepper.FORWARD)
            canMove = self.plasticMotorIR.is_object_detected()
        else:
            self.display.updateQueue({"warning":"Syringes were not full
            break
    return
```

Figure 5.27: Microplastic Dispense Water Function

For safety reasons, whenever we run a stepper motor we first check if the doors are closed and halt the movement of motors if the doors are not closed. We wait for the doors to be closed and then run the stepper motors.

```
def run_stepper(self, stepper_motor, steps, direction=stepper.FORWARD, style=stepper.DOUBLE):
    if (not self.areDoorsClosed()):
        closed = self.areDoorsClosed()
        self.display.updateQueue({"warning":"CLOSE DOORS"})
        while (not closed):
            closed = self.areDoorsClosed()

    for i in range(steps):
        stepper_motor.onestep(direction=direction, style=style)
```

Figure 5.28: Motorkit onestep function inside run_stepper()

6.8. Conveyor Control CSC

The conveyor control CSC is a simple runstepper function that moves for 1cm until the IR breakpoint is detected by the conveyor verification sensor CSC. The run_stepper function is highlighted in Figure 5.21. The Conveyor CSC is used to fetch a slide via the nubs affixed to the conveyor belt and move the slide underneath the dropper. The CSC then moves the

conveyor and slide underneath the camera or microscope depending on the subsystem. Finally the CSC moves the conveyor belt back to the starting position before the cartridge, dumping the slide off the end of the conveyor belt during.

```
def resetConveyorBelt(self, ir, message, motor):
    print(message)
    detected = ir.is_object_detected()
    while (not detected):
        print("Moving Conveyor Belt")
        self.run_stepper(motor, self.cm_step)
        detected = ir.is_object_detected()

def moveConveyorToSensor(self, targetIR, startIR, message, motor):
    print(message)
    detected = targetIR.is_object_detected()
    while ((not detected)):
        print("Moving Conveyor Belt")
        self.run_stepper(motor, self.cm_step)
        detected = targetIR.is_object_detected()
        startDetected = startIR.is_object_detected()
        if (startDetected):
            raise ConveyorGoAroundError("Conveyor belt at the start, although its not supposed to be.")
```

Figure 5.29: Move Conveyor and Reset Conveyor Functions

The moveConveyorToSensor() function takes in a target IR break sensor, the IR break sensor at the start of the system, a message to inform the user where the conveyor is being moved to, and the motor for the specific conveyor. The motor is moved until the breakpoint sensor is blocked by the nubs on the conveyor belt. If the IR break sensor at the start detects the nubs, then that means the target IR sensor missed the nub, and the conveyor moved all the way back to the beginning. This means that the slide has been thrown off the edge of the conveyor belt and the sequence should be aborted.

The resetConveyorBelt() function is used at the end of the sequence for microplastic or paperfluidic detection or in case of a fatal error to send the conveyor belt, the IR nubs, and the nub that grabs the slides back to the start of the system. Below is an example usage of these functions in the microplastic detection sequence:

```
def microplasticDetection(self, key):
            self.resetConveyorBelt(firstIR, "Resetting conveyor belt", self.kit.stepper1)

        concentration = sum

        mpChar = self.getCharacteristic("Microplastic")
        if (mpChar):···
        else:
            print("Microplastic characteristic not found")
    except Exception as e:
        print(f"Caught exception: {e}")
        try:
            self.resetConveyorBelt(firstIR, "Cancelling microplastic detection and discarding any active trays", self.kit.stepper1)
        except Exception as ee:
            print(f"Fatal error while canceling microplastic detection, after an error had already occured. FATAL ERROR: {ee}")
        return
```

Figure 5.30: Move Conveyor and Reset Conveyor Functions Usage Examples

6.9. Conveyor Verification Sensor CSC

The IR break sensors used in the system wrapper are a custom class that takes in a GPIO pin as input upon creation. The class contains a function called is_object_detected which returns True if the IR sensor value is HIGH and returns False if it is LOW. The return value of this function is used in the conveyor CSC as shown in Figure 5.22.

```
class IRSensor:
    def __init__(self, pin):
        self.ir_sensor = Button(pin)

    def is_object_detected(self):
        if self.ir_sensor.value:
            #print("Object detected!")
            return True
        else:
            #print("No object detected!")
            return False
```

Figure 5.31: IR Sensor Class

6.10. Water Baddies Control GUI CSC

Using the DisplayHatMini and ST7789 library, a basic GUI has been developed to be displayed on the Raspberry PI HAT. The GUI is controlled using the DisplayHat class, which is a custom class to ensure consistency in the display of data. The display has two main purposes: to display the currently running stage and any warnings, and to listen for user inputs on buttons to start detection sequences or reset anything. The DisplayHat is initialized in the SystemWrapper and is used in the SystemWrapper via the updateQueue function. This function serves to hold messages from the system and slowly display them on the display in order to not break the system:

```
class DisplayHat():
    def updateText(self):
        while True:
            try:
                texts = self.messageQueue.get()
                if ("stage" in texts):
                    self.stage = texts["stage"]
                if ("warning" in texts):
                    self.warning = texts["warning"]
                time.sleep(3)
                self.update_display()
            except:
                pass

    def updateQueue(self, text):
        self.messageQueue.put(text)
```

Figure 5.32: Messaging Queue for Display Text

Upon initialization, the DisplayHat assigned a function to each button. In order for the button press and hold to work, the press is fired when the button is released, and the hold is fired after holding for 3 seconds. In the button_a_pressed and button_a_hold we check if the button has been held when its released and fire the press command if it has not.

```
class DisplayHat():
    def __init__(self, startMicroplasticDetection, startInorganicsMetalDetection, startAll, restartBluetooth, demo):
        self.displayhatmini.button_a.when_released = self.on_button_a_pressed
        self.displayhatmini.button_b.when_released = self.on_button_b_pressed
        self.displayhatmini.button_x.when_released = self.on_button_x_pressed
        self.displayhatmini.button_y.when_released = self.on_button_y_pressed

        self.displayhatmini.button_a.when_held = self.on_button_a_held
        self.displayhatmini.button_b.when_held = self.on_button_b_held
        self.displayhatmini.button_x.when_held = self.on_button_x_held
        self.displayhatmini.button_y.when_held = self.on_button_y_held

        self.messageThread = threading.Thread(target=self.updateText, daemon=True)
        self.messageThread.start()
```

Figure 5.32: DisplayHat Button Functions

```
def on_button_a_pressed(self):
    if not self.button_a_held:
        if (not self.plasticActive):
            self.button_a_held = False
            print("Microplastics pressed")
            self.plasticActive = True
            self.microplasticFunction()
        else:
            self.updateQueue({'warning': 'Cannot Start Microplastic

    self.button_a_held = False
```

Figure 5.33: DisplayHat Button Press

```
def on_button_a_held(self):
    print("Button A hold")
    self.updateQueue({'warning':
    self.button_a_held = True
```

Figure 5.34: DisplayHat Button Hold

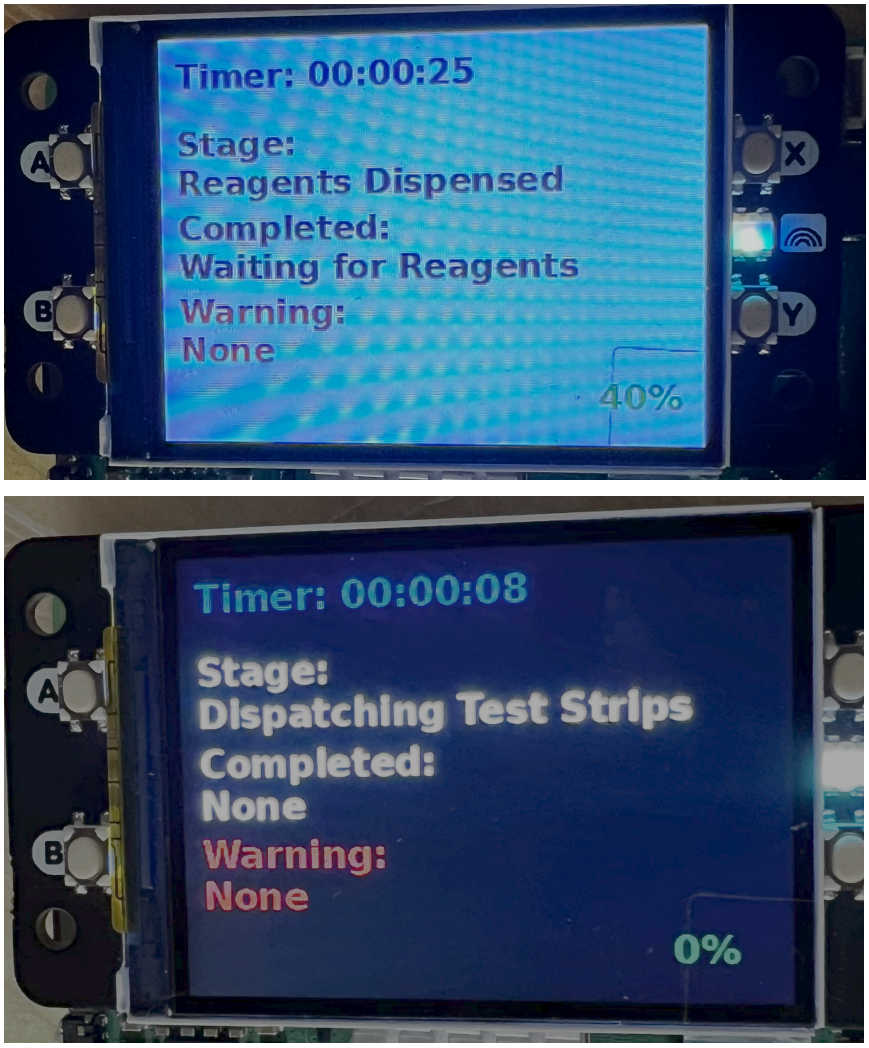


Figure 5.35: The GUI Display

6.11. Water Baddies System Wrapper

The Water Baddies System wrapper runs the entire Water Baddies Detection System. The System Wrapper orchestrates all of the different CSCs, running them when needed, performing timing management, and performing error handling. The system wrapper begins running the moment the Detection System, specifically the Raspberry Pi, is turned on. On initialization, the system will initialize any system variables, create the Bluetooth GATT service, and run the main event loop for the middleware for the Bluetooth:

```python
class System:

    def __init__(self):
        dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)

        self.bluetooth = None
        self.bus = dbus.SystemBus()
        self.app = None
        self.adv = None
        self.loop = None
        self.kit = MotorKit()
        self.kit2 = MotorKit(address=0x61)
        self.releaseMotors()
        self.cm_step = 31
        self.firstIR = IRSensor(14)
        self.dropperIR = IRSensor(18)
        self.microscopeIR = IRSensor(23)

        self.PlasticFirstIR = IRSensor(12)
        self.PlasticDropperIR = IRSensor(26)
        self.PlasticMicroscopeIR = IRSensor(20)

        self.plasticMotorIR = IRSensor(0)
        self.paperMotorIR = IRSensor(1)

        self.plasticLED = LED(19)

        self.motorSteps = 30

        self.paperButton = Button(4)
        self.plasticButton = Button(15)

        self.startBluetooth()
        self.display = DisplayHat(
            self.startMicroplasticDetection,
            self.startInorganicsMetalDetection,
            self.startDetection,
            self.restartBluetooth,
            self.startDemo)
```

Figure 5.36: systemWrapper.py initialization

During initialization, the wrapper also creates the display hat object which listens for input from the display buttons, releases the motors, and initializes all of the GPIO pins so they can be used in a multithreaded context.

MicroplasticDetection:
One of two main functions for the Water Baddies Detection System, this function links together multiple CSCs to analyze the concentration of microplastics in 5 water samples and send the average to the Water Baddies App via Bluetooth. The detection begins by initializing the IR breakpoint sensors with their associated GPIO pins. Next the function moves the conveyor to grab an optical slide and stop under the water dropper, which drops water on the slide.

```
def microplasticDetection(self, key):
    try:
        firstIR = IRSensor(26)
        dropperIR = IRSensor(20)
        microscopeIR = IRSensor(12)
        self.resetConveyorBelt(firstIR, "Resetting microplastic conveyor belt", self.kit.stepper1)
        time.sleep(2)
        sum = 0
        for i in range(5):
            print("Starting microplastic slide" + str(i+1))
            self.moveConveyorToSensor(dropperIR, firstIR, "Fetching microplastic slide and moving slide under dropper", self.kit.stepper1)
            time.sleep(2)
            self.dispensePlasticWater()
            time.sleep(2)
            self.moveConveyorToSensor(microscopeIR, firstIR, "Moving microplastic slide under microscope", self.kit.stepper1)
            time.sleep(2)
            try:
                imagePath = self.captureMicroscopeImage()
                print(f"Microplastic image path: {imagePath}")
            except Exception as e:
                print(f"Error during image capture: {e}")
                raise ImageCaptureError("Error during capturing image from the micropscope. Canceling microplastic job!")

            try:
                testImagePath = "./test_images/Snap_027.jpg"
                quantity = microplastic_concentration(testImagePath)
                #quantity = microplastic_concentration(imagePath)
            except Exception as e:
                print(f"Error during image analysis: {e}")
                raise ImageCaptureError("Error during capturing image from the micropscope. Canceling microplastic job!")

            sum += quantity
            self.resetConveyorBelt(firstIR, "Resetting conveyor belt", self.kit.stepper1)

        concentration = sum
```

Figure 5.35: Microplastic Detection Function

The conveyor then moves under the microscope, where an image is captured, and that image is analyzed using the Microplastic Illuminator and Capture CSC. Finally, the conveyor belt is reset, and the concentration value is written to Bluetooth.

```
mpChar = self.getCharacteristic("Microplastic")
if (mpChar):
    mpChar.WriteValue(str(concentration))
    print("Updated value:"+ str(concentration))

    self.updateKey(key)
else:
    print("Microplastic characteristic not found")
```

Figure 5.36: Concentration and Key Upload

For the app to know when the data has been updated, the Bluetooth service contains a key. The key is a timestamp of when the detection started and is compared to a stored key in the water baddies app.

InorganicsMetalDetection:
This function coordinates all of the functions and CSCs necessary for the Water Baddies Detection System to analyze the paper fluidic device in order to get the concentrations of inorganics and heavy metals in a water sample. The detection function first moves the motor and grabs a cartridge, stopping it under the water dropper.

```
def InorganicsMetalDetection(self, key):
    try:
        firstIR = IRSensor(26)
        dropperIR = IRSensor(20)
        microscopeIR = IRSensor(12)
        self.resetConveyorBelt(firstIR, "Resetting paperfluidic conveyor belt", self.kit.stepper1)
        time.sleep(2)

        self.moveConveyorToSensor(dropperIR, firstIR, "Fetching paperfluidics and moving the slide under the water dropper", self.kit.stepper1)
        time.sleep(2)
        self.dispenseFluidicWater()
        time.sleep(2)
        self.moveConveyorToSensor(microscopeIR, firstIR, "Moving paperluidics under the microscope", self.kit.stepper1)
        try:
            imagePath = self.capturePiImage()
            print(f"First paperfluidics image: {imagePath}")
            print("Waiting for lead reaction")
            time.sleep(3)
            leadImagePath = self.capturePiImage() #Just capture lead image
            print(f"Lead paperfluidics image: {leadImagePath}")

            #Just for testing
            testImagePath = "./test_images/paperfluidic.jpg"
            testLeadImagePath = "./test_images/paperfluidic_test.jpg"
        except Exception as e:
                print(f"Error during image capture: {e}")
                raise ImageCaptureError("Error during capturing image from the PiCamera for lead or base image. Canceling paperfluidics job!")
```

Figure 5.37: InorganicsMetalDetection

Once water is dispensed, the conveyor moves the cartridge under the piCamera where a photo is taken using the capturePiImage() function:

```
def capturePiImage(self):
    picam = Picamera2()
    picam.configure(picam.create_still_configuration()) # Add this line
    picam.start()
    picam.set_controls({"AfMode": controls.AfModeEnum.Continuous})
    path = f'paperFluidicImages/{datetime.now().strftime("%Y-%m-%d-%H-%M-%S.%f")[:-3]}.png'
    picam.capture_file(path)
    picam.close()
    return path
```

Figure 5.38: Capture Pi Image Function

The detection function then takes a picture after 30 seconds and processes the image, only extracting the value for lead. Due to the incubation time for the other baddies, the cartridge will sit for another 10 minutes, when another photo will be taken and compared to the original in the colorimetric analysis script.

```
try:
    #leadConcentration = paperfluidic_concentration(imagePath, leadImagePath)['lead']
    print("Starting Lead Concentration Analysis")
    vals = pfa.paperfluidic_concentration(testImagePath, testLeadImagePath)
    leadConcentration = vals['Lead']
except Exception as e:
        print(f"Error during image analysis: {e}")
        raise ImageCaptureError("Error analyzing lead image. Canceling paperfluidics job!")

print("Waiting for paperfluidic reactions")
time.sleep(5)

try:
    finalImagePath = self.capturePiImage()
    print(f"Final paperfluidics image: {finalImagePath}")

    testFinalImagePath = "./test_images/paperfluidic_test.jpg"
except Exception as e:
        print(f"Error during image capture: {e}")
        raise ImageCaptureError("Error during capturing final image from the PiCamera. Canceling paperfluidics job!")

try:
    #concentration = paperfluidic_concentration(imagePath, finalImagePath)
    concentration = pfa.paperfluidic_concentration(testImagePath, testFinalImagePath)
    concentration['Lead'] = leadConcentration
except Exception as e:
        print(f"Error during image capture: {e}")
        raise ImageCaptureError("Error during capturing image from the PiCamera. Canceling paperfluidics job!")
```

Figure 5.39: Colorimetric Capture CSC

The concentrations and the key are then uploaded via Bluetooth to the Water Baddies Mobile app. Finall,y the conveyor will be reset to the beginning of the system, dropping the cartridge off the end of the conveyor belt.

```
    nitriteChar = self.getCharacteristic("Nitrite")
    if (nitriteChar):
        nitriteChar.WriteValue(str(concentration["Nitrite"]))
        print("Updated value:"+ str(concentration["Nitrite"]))

        self.updateKey(key)
    else:
        print("Nitrite characteristic not found")

    self.resetConveyorBelt(firstIR, "Resetting the paperfludics conveyor belt", self.kit.stepper1)
except Exception as e:
    print(f"Caught exception: {e}")
    try:
        self.resetConveyorBelt(firstIR, "Canceling paperfluidics and resetting conveyor belt", self.kit.stepper1)
    except Exception as ee:
        print(f"Fatal error while canceling paperfluidic detection, after an error had already occured. FATAL ERROR: {ee}")
    return
```

Figure 5.40: End of InorganicsMetalDetection

7.0 Requirements Traceability

The requirements traceability matrix shows the flow down of software related system level requirements to the subsystem level with specific software requirements. To see what level each requirement will be verified on see the Verification Table. For requirements validation and procedures, see the Verification Plan.

System Requirement ID	System Requirements	Computer Requirement Flow-Down
SR001 (T)	The Water Baddies system shall detect the concentration of microplastics, metals, and inorganics in a water sample with 85-90% accuracy.	1. (T) Will analyze the images of 'water baddies' and calculate concentrations
SR003 (T)	The Water Baddies system shall use blue LEDs filtered with an excitation filter to 445nm to fluorescent microplastics.	(D/I) The Blue LEDs will turn on when the microplastic sample is ready to be analyzed
SR007 (T)	The Water Baddies App shall have a data loss rate of less than 1%.	1. (I/T) The software shall verify the validity of the data it receives 2. (D) The App will use Bluetooth Low Energy
SR015 (I)	The System shall have 4 GB of ram to be able to process and analyze data.	(D) The software must run using a maximum of 4GB of ram

SR016 (T)	The Water Baddies' water dispensing system shall dispense water in increments of 120 μL with a standard deviation of ± 20μL.	(T) The software shall spin the dropper for enough time for120 μL of water to be dispensed
SR017 (I)	The Water Baddies system shall analyze one paper fluidic and one microplastic sample per cycle.	(I/T) Every cycle, the software will analyze two images, one of the paperfluidics and one photo of microplastics
SR020 (D)	In an emergency shutoff, the system shall abort the previous test after being rebooted and give a visible alert from the mobile application.	1. (T/D) The software shall cycle power to all components and move them to their starting positions after a shutoff 2. (T) The mobile application shall display when the system shuts off
SR009 (D)	The Water Baddies GUI shall have an instruction set for using the system.	1. (D) The software shall load a list of instructions onto the touchscreen 2. (D) The mobile application shall have instructions for using the app
SR011 (D)	The Water Baddies system will generate a report outlining the concentration of microplastics within the sample, the concentrations of lead, cadmium, as well as the concentration of nitrates, phosphates, and nitrites in parts per ml.	1. (T/A) The Software will analyze data from the camera and microscope in order to generate the report. 2. (D) Shall be able to download a report of the system outputs. Display the graphs of each concentration
SR021 (D)	The Water Baddies App will give an audible and haptic alert if detection levels of each detectable are above the thresholds established by the EPA in their report on National Primary Drinking Water Regulations.	(D) Shall be built using haptic feedback and accessibility features

SR026 (D)	The Water Baddies system shall operate fully autonomously once accepting user input of water reservoir, test material cartridges, and test selection via touchscreen.	1. (D) The software shall run all devices in sequence, and process all data one detecting input from the user 2. (D) The mobile application shall collect result data once the cycle is finished
SR025 (D)	The Water Baddies App shall have a single connection page to aid the user in connecting to the device.	(I) Shall have a Bluetooth button that opens a page and displays all available devices
SR029 (D)	The Water Baddies App shall download the report as a PDF.	(D) Shall have a button to download the PDF
SR013 (D)	The Water Baddies system report shall display one particle concentration per carousel slide	(I) Shall show one group concentration on each page/slide
SR008 (D)	The Water Baddies System shall take a new user 10 minutes to learn.	1. (D) The GUI for both the App and Raspberry will have detailed instructions, showing step-by-step how to use it 2. (I/D) Shall have an intuitive interface design and instructions of how to use the app

8.0 Notes

Github Repository for App: https://github.com/austinmedina/WaterBaddiesApp
Github Repository for Raspberry Pi: https://github.com/austinmedina/WaterBaddiesPi

Verification Plans

Version 1.1

Document #100710

January 28th, 2025

WATER BADDIES - Microplastic, Heavy Metal and Inorganics Water Detection System for Environmental and Human Health

Team 25040

Brendan Bamberg

Jameson Brehmer

Daniel Knaus

Austin Medina

Alia Nichols

Aidan Talucci

Tyler Thursby

Revision History

1/27/2025 Rev(–) Created the initial draft of the Verification Procedures
1/28/2025 Rev(1.1) SR 3.2 switched to Blue and White LEDs and LEDs will be powered by Pi. Cleaned up the Verification plans to only include test and analysis procedures.
04/08/2025 Rev(1.2) Updated SR024, SR027, & SR028

Table of Contents

1.0 SR001

1.1. Paper Fluidic Verification

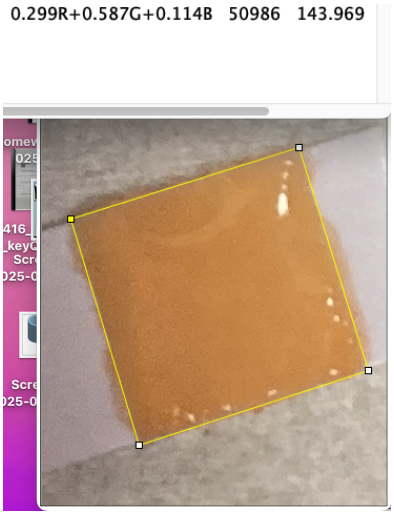
Requirement: Colorimetric change calculation will generate a concentration that is 85% accurate of a controlled concentration of heavy metals. Colorimetric change calculation will generate a concentration that is 85% accurate to a controlled concentration of inorganics.

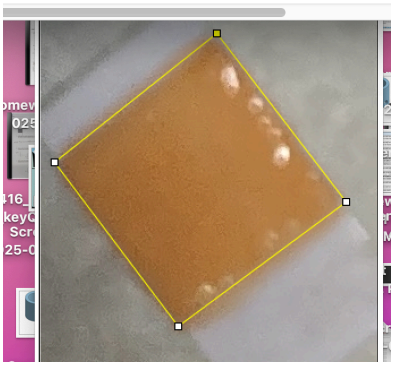
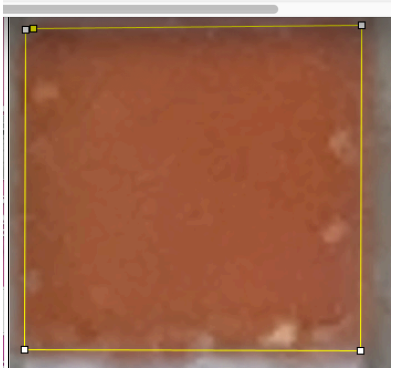
Verification Type: Test

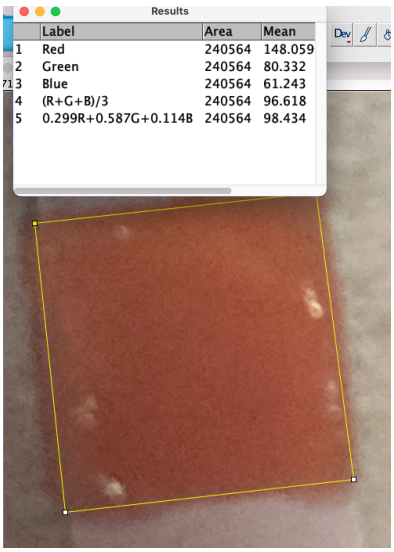
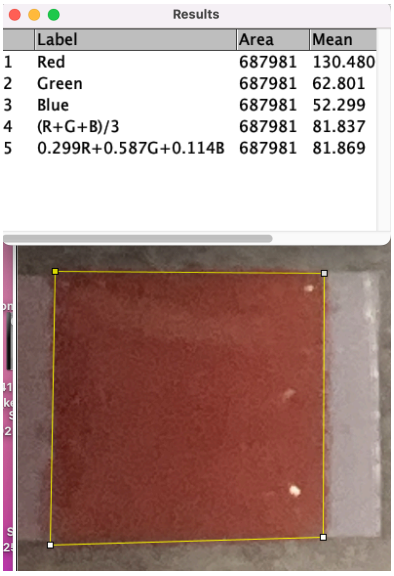
Expected date of completion: 3/6/25

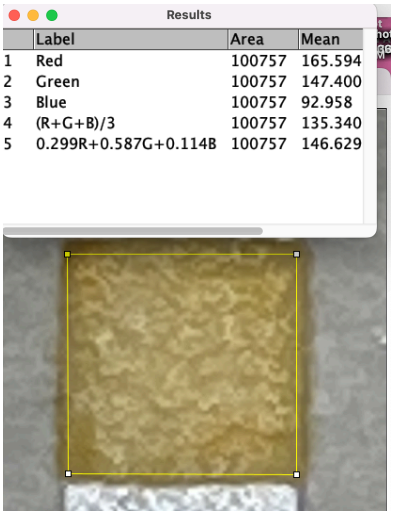
Procedure: First remove our selected test pad from the premade test strip using scissors. We then drop a 10 µl drop of our water with our selected concentration of the appropriate contaminants. After at least 30 seconds an image is taken of the test pad and ran through ImageJ to receive the luminance value. After each drop, luminance is determined three times and the middle value is taken concentration. This is repeated through every predetermined concentration. Eventually, each heavy metal concentration gradient will be analyzed to form a curve which relates RGB to the concentration of that heavy metal using luminance. The same procedure will be performed with different combinations of the heavy metals with different concentrations. Following the establishment of this curve, an unknown concentration of a mixture of heavy metals will be created and analyzed using our curve.

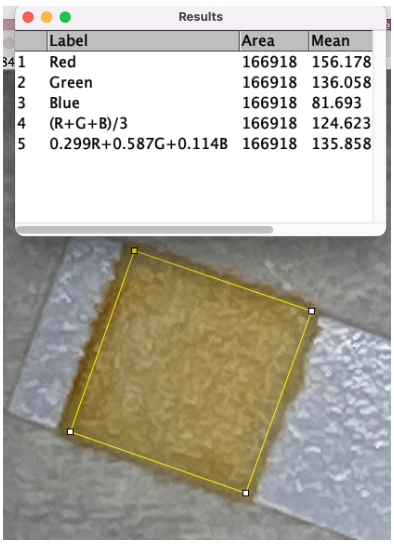
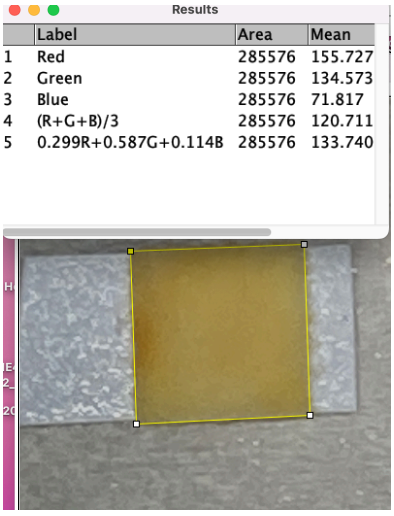
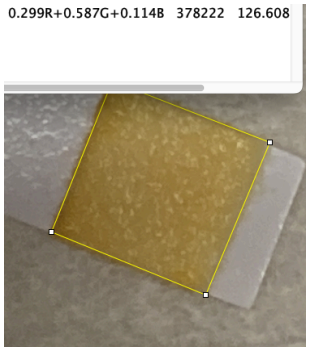
Data:

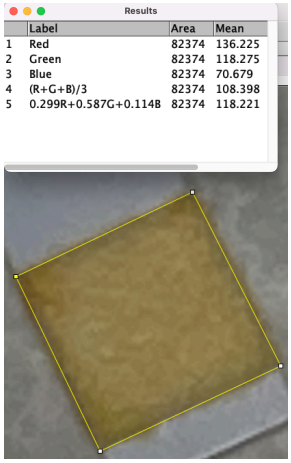
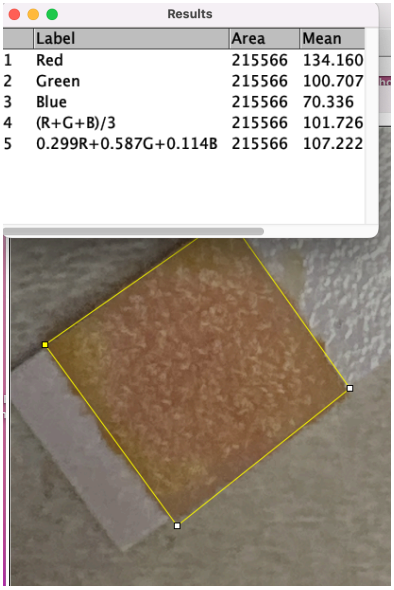
Cadmium: 44.5 nm Max Concentration		
Cadmium Concentration	ImageJ Value	Photo
0 (DD Water)	143.9	

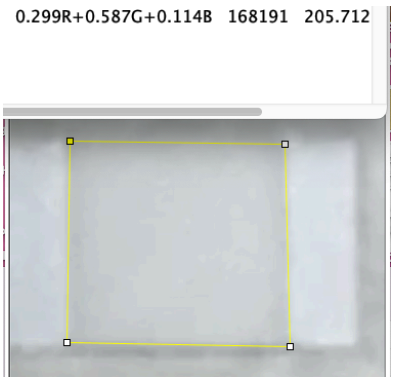
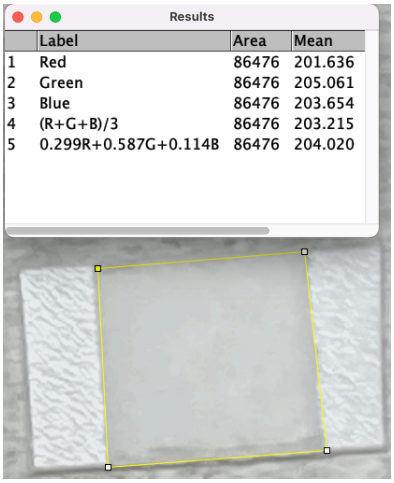
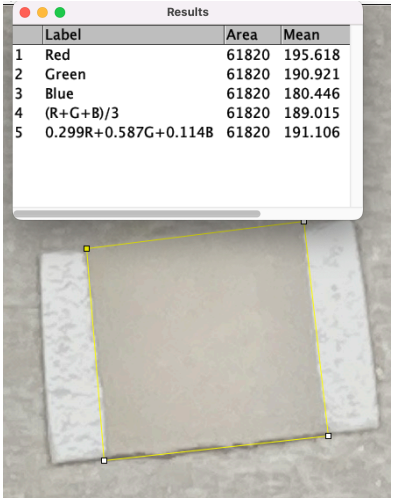
10 nm	130.45	0.299R+0.587G+0.114B 33876 130.450 
100 nm	119.7	0.299R+0.587G+0.114B 77760 119.666 
500 nm	106.2	0.299R+0.587G+0.114B 88750 106.154 

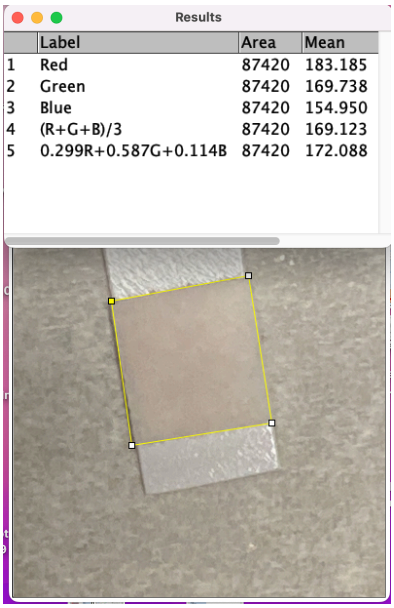
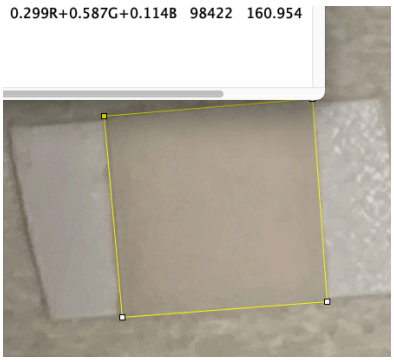
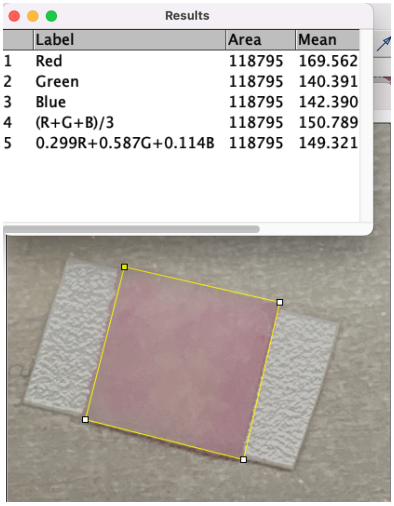
1000 nm	98.4	
5000 nm	81.9	

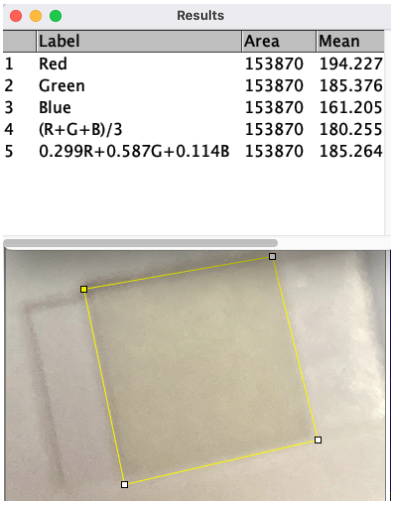
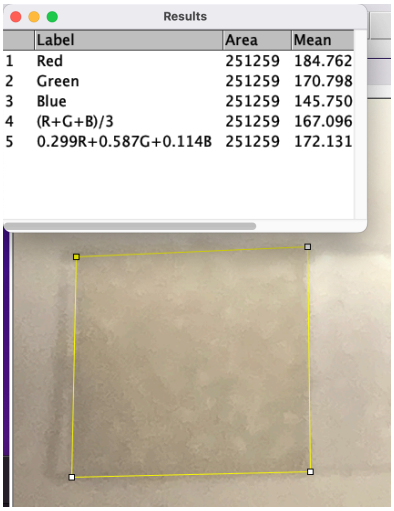
Lead: 48.3 nm max		
0 (DD Water)	146.6	

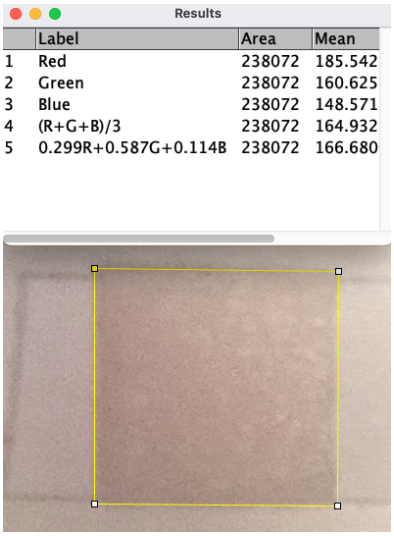
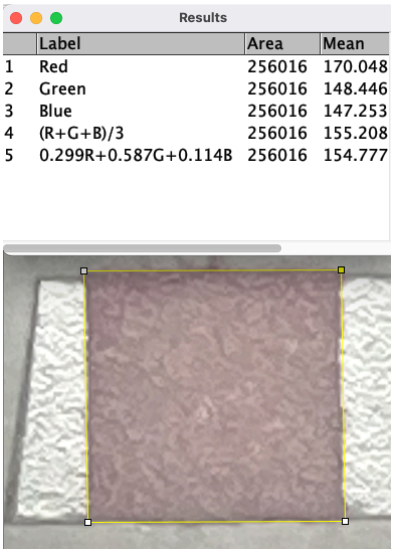
10 nm	135.9	
100 nm	133.7	
500 nm	126.8	

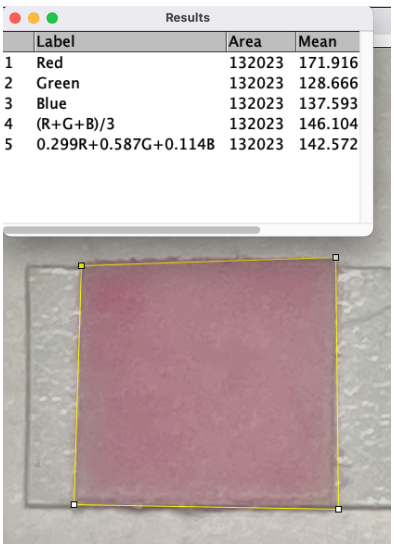
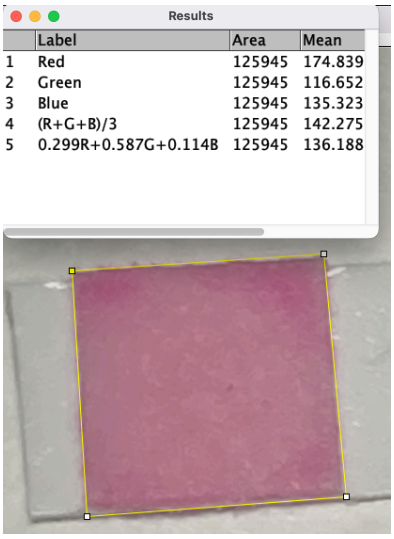
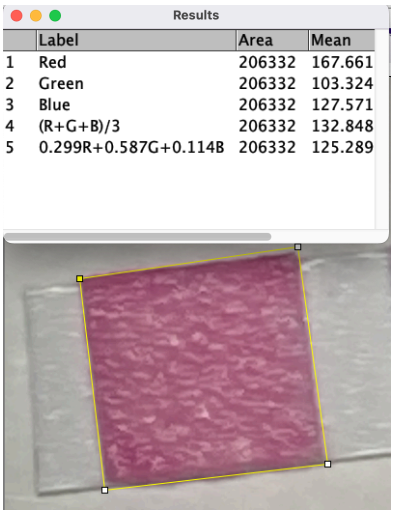
1000 nm	117.6	
.015 mm	107.2	

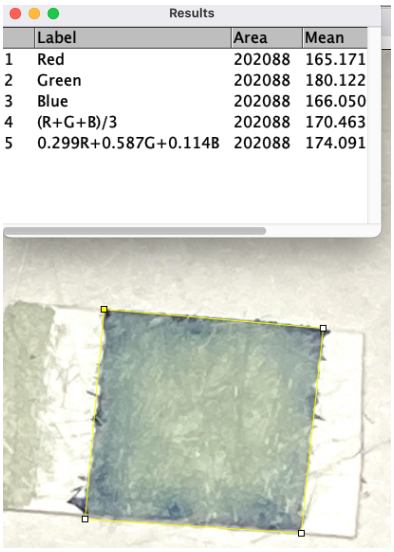
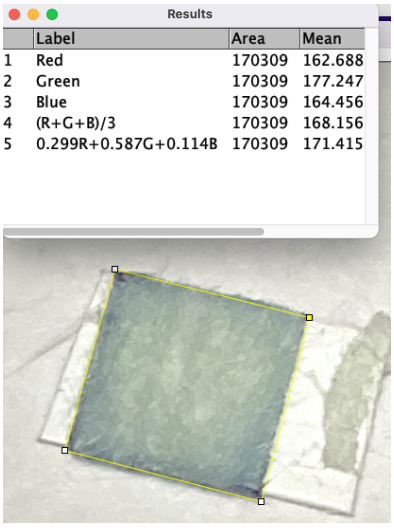
Nitrate: 10 ppm		
Nitrate Concentration	ImageJ Value	Photo
0 ppm	205.7	
25 ppm	204	
50 ppm	191.1	

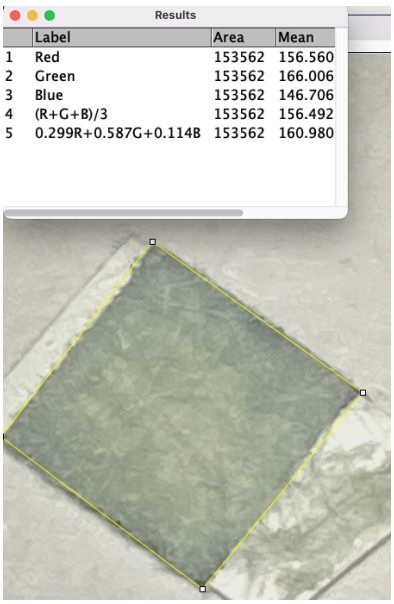
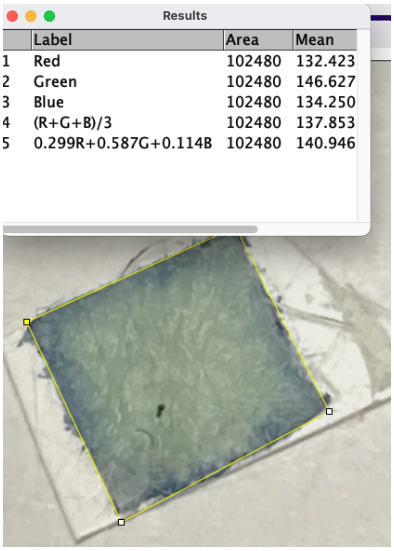
100 ppm	172.1	
500 ppm	161.0	
1000 pm	149.3	

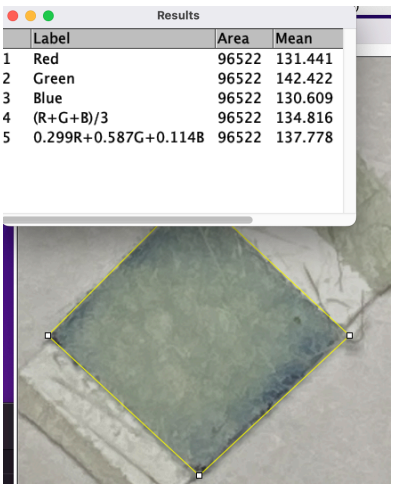
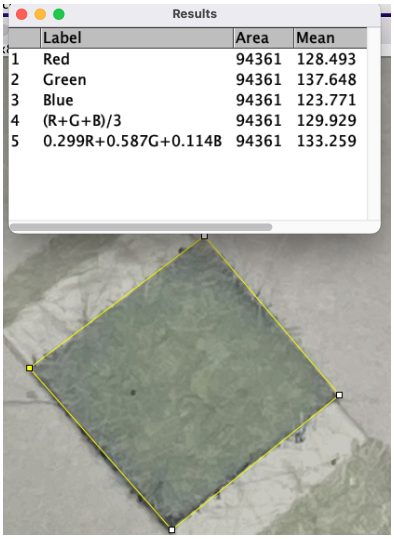
Nitrite: 1 ppm		
Nitrite Concentration	ImageJ Value	Photo
0 ppm	182.26	
1 ppm	172.13	

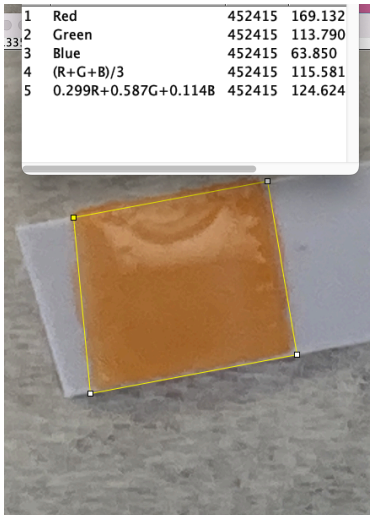
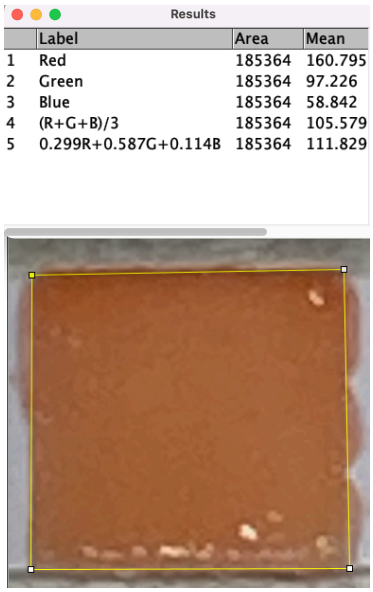
5 ppm	166.68	
10 ppm	154.78	

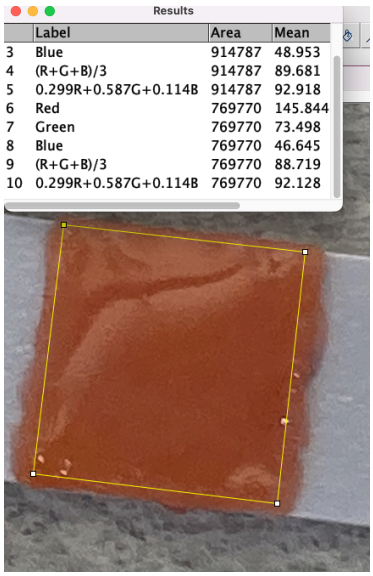
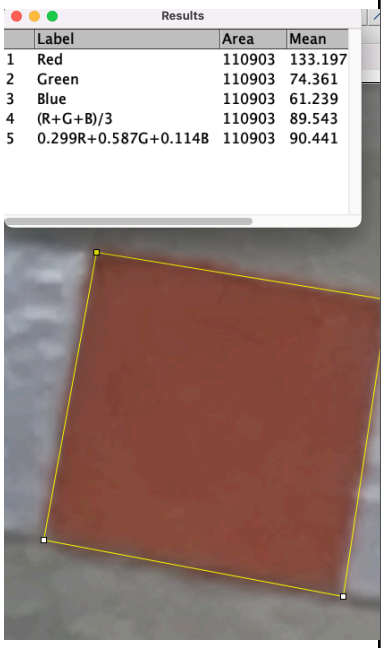
20 ppm	142.57	
40 ppm	136.19	
80 ppm	125.29	

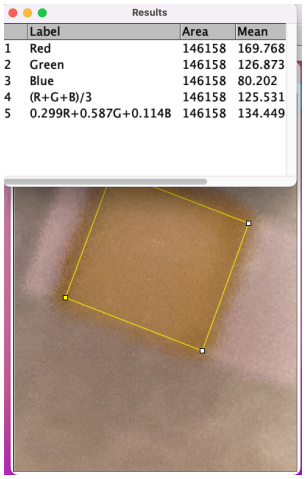
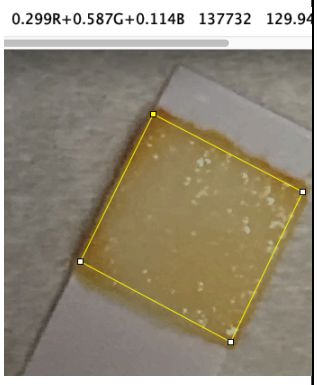
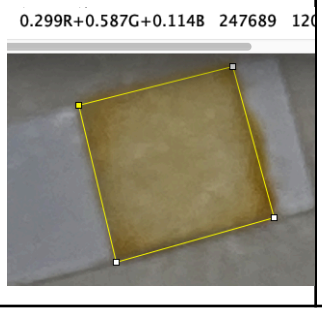
Phosphate: 100 ppb		
Phosphate Concentration	ImageJ Value	Photo
0 ppb	174.09	
100 ppb	171.42	

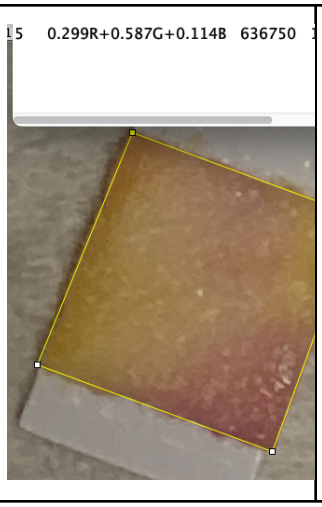
300 ppb	160.98	
500 ppb	140.95	

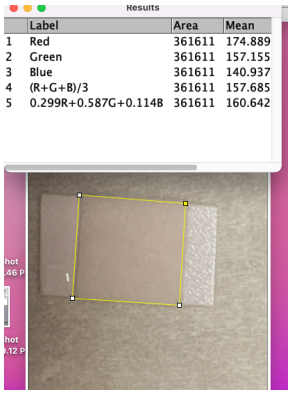
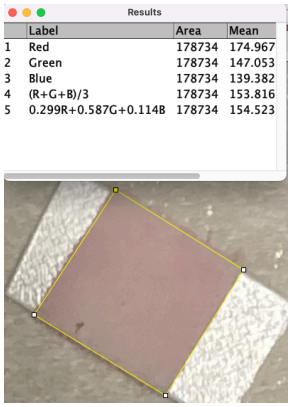
1000 ppb	137.78	
2500 ppb	133.259	

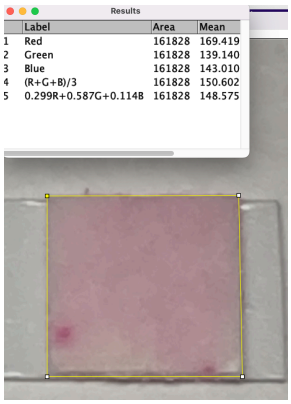
Cadmium			
Cadmium Concentration	ImageJ Value	Photo	Percent Error
55 nm	124.62		Projected Concentration: 62.02 12.76%
300	111.83		Projected Concentration: 330.47 10.16%

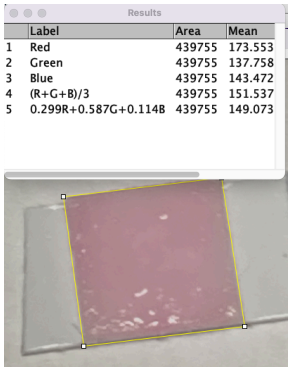
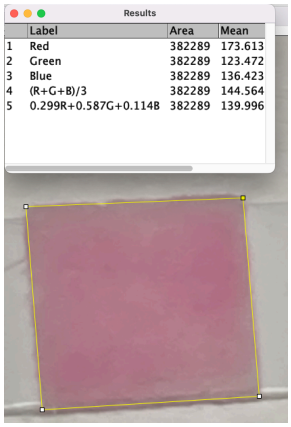
2750	92.128		Projected Concentration: 2632.45 4.27%
3000	90.441		Projected Concentration: 3040.92 1.36%

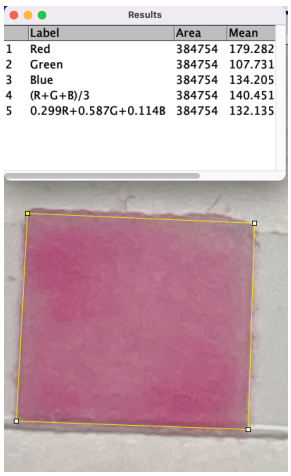
Lead			
Lead Concentration	ImageJ Value	Photo	Percent Error
55 nm	134.4		Projected Concentration: 62.52 13.67%
300	129.9		Projected Concentration: 294.8 1.9%
750	120.8		Projected Concentration: 826.09 10.54%

750	120.9		Projected Concentration: 820.65 9.42%

Nitrate			
Concentration	ImageJ Value	Photo	Percent Error
375 ppm			
525 ppm	160.64		Projected Concentration: 528.12 .59%
750 ppm	154.5		Projected Concentration: 790.6 5.41%

1000	148.58		Projected Concentration: 1000.85 .08%

Nitrite			
Concentration	ImageJ Value	Photo	Percent Error
15 ppm	149.07		Projected Concentration: 14.7 2%
30 ppm	140		Projected Concentration: 28.2 6%

60 ppm	132.14		Projected Concentration: 54.43 9.28%

Data sheet:

SR001			
Referenced VTP Paragraph Number:			
Analysis Referenced (for verification by T/A): Test			
Name of Test: Colorimetric Concentration Accuracy			
Unit Under Test: Paper Fluidic			
Name: Paper Fluidic Part Number:100100 Serial Number:			
Results (Pass / Fail): Pass		Date of Test: 4/2	
Recording of Test Measurement: Cadmium: Avg. Error: 7.13% Worst: 12.76% Lead: Avg. Error: 8.88% Worst: 13.67% Nitrate: Avg. Error: 2.03% Worst: 5.41% Nitrite: Avg. Error: 5.76% Worst: 9.28% Phosphate: Avg. Error: 5.5% Worst: 9%	Requirement (SR, with tolerances): Concentration accuracy must be within a 15% error.	Test Equipment Error:	Adjusted Test Limit:
Computations, (Include Analyses Results, if any): (55-62.52)/62.52 = -.1367			
Signatures: Tester____ Jameson Brehmer_____ Customer_____			

1.2. Optical Assembly Verification

Requirement: Shall be able to detect fluorescent microplastics ≥ 5 μm in size using a microscope, a laser, and a bandpass filter. Using a controlled quantity analysis test, accuracy can be calculated.

Verification Type: Analysis

Expected date of completion: 2/25/25

Procedure: To analyze the accuracy of the Water Baddies system, there will be procedures pertaining to accurately quantifying microplastic concentrations and accurately detecting color change in paper fluidics used to indicate the presence of inorganics and heavy metals. The algorithm will quantify the microplastics and give a number which will be compared to the value determined by the team by hand. The tester will place varying known quantities of microplastics ranging in size from 5μm to 5mm, then run the algorithm and verify the script counts the microplastics accurately. We have provided abridged data reflecting successful microplastic concentration testing. For future iterations, higher volume of trials is required for better confidence intervals. The System is required to undergo testing as a fully integrated design because the above data does not reflect the conditions in which the microplastics will be detected while in the enclosed system. While we tested to the best of our abilities in an environment that would best simulate the conditions inside of the machine, there are always limitations as student researchers.

Data:

Test	7.2 ppmL	
	Particles	Concentration
1	3	35.08
2	8	93.55
3	3	35.08
4	6	70.16
5	3	35.08
6	2	23.39
7	3	35.08
8	2	23.39
9	5	58.47
10	3	35.08
Average Concentration	50.67	44.44
Conversion factor	7.04	6.17
Avg CF Concentration	7.07	6.67
Percent Error (%)	1.77	7.38

Test	6.6 ppmL	
	Particles	Concentration
1	4	46.77
2	5	58.47
3	2	23.39
4	2	23.39
5	3	35.08
6	3	35.08
7	3	35.08
8	8	93.55
9	3	35.08
10	3	35.08
Average Concentration	48	42.097
Conversion factor	7.272727273	6.378333333
Avg CF Concentration	6.70	6.32
Percent error (%)	1.52	4.28

Test	2.52 ppml	
	Particles	Concentration
1	0	0
2	0	0
3	4	46.77
4	4	46.77
5	0	0
6	0	0
7	0	0
8	2	26.667
9	4	46.77
10	0	0
Average Concentration	18.66666667	16.6977
Conversion factor	7.407407407	6.626071429
Avg CF Concentration	2.605575118	2.505793997
Percent Error (%)	3.395838014	0.563730267

Test	8.5 ppmL	
	Particles	Concentration
1	4	46.77
2	4	46.77
3	4	46.77
4	3	35.08
5	6	70.16
6	2	23.39
7	7	81.85
8	7	81.85
9	4	46.77
10	1	11.69
Average Concentration	56	49.11
Conversion factor	6.588235294	7.440909091
Avg CF Concentration	7.82	7.37
Percent Error (%)	8.04	13.30

1.3. Electrical Assembly Verification

Requirement: The subsystem will power the LEDs and Camera for the correct amount of time

Verification Type: Test

Expected date of completion: 2/25/25

Procedure:

1. Power on and then run the system.
2. The automated system will use IR Break Sensor to recognize when the objects are in the correct positions.
3. The LED will stay on for the duration of the time for the Microscope and Camera to take in data.
4. Once the data has been collected, the system will discard the test samples.

Data sheet:

SR001.3			
Referenced VTP Paragraph Number:			
Analysis Referenced (for verification by T/A): T			
Name of Test: Software LED And Camera Test			
Unit Under Test: **Raspberry Pi/Microplastic Capture CSC/Colorimetric**			
Name: Part Number: Serial Number:			
Results (Pass / Fail): **PASS**		Date of Test: 4/9/2025	
Recording of Test Measurement: file path exists and is lit	Requirement (SR, with tolerances): The subsystem will power the LEDs and Camera for the correct amount of time	Test Equirment Error: N/A	Adjusted Test Limit: N/A
Computations, (Include Analyses Results, if any): N/A			
Signatures: Tester: **Austin Medina** Customer_____			

1.4. Software Assembly Verification

Requirement: Will analyze the images of 'water baddies' and calculate concentrations.

Verification Type: Test

Expected Date of Completion: 2/25/25

Procedure:

1. Take images of known concentrations of ‘water baddies’ and pass them to the corresponding analysis software
2. Run the analysis software and verify that the output is a concentration that matches the known concentration

Data sheet:

SR001.4			
Referenced VTP Paragraph Number:			
Analysis Referenced (for verification by T/A): T			
Name of Test: Software Detection Test			
Unit Under Test: **Raspberry Pi/Microplastic Capture CSC/Colorimetric**			
Name: Part Number: Serial Number:			
Results (Pass / Fail): **PASS**		Date of Test: 2/23/25	
Recording of Test Measurement: Microplastics: 300.508572, Lead: 34.4, Mecury: 15.6, Nitrate: 56.33, Nitrite: 76.52, Cadmium: 56.98	Requirement (SR, with tolerances): The Water Baddies App shall have a data loss rate of less than 1%	Test Equirment Error: N/A	Adjusted Test Limit: N/A
Computations, (Include Analyses Results, if any): N/A			
Signatures: Tester: **Austin Medina** Customer_____			

2.0 SR002

2.1. Optical Assembly Verification

Requirement: Magnification verification will be done by inspection using the microscope specifications sheet

Verification Type: Inspection

Expected Date of Completion: 1/28/25

Procedure:

1. Check IDL Part 100730 to verify the magnification.

Datasheet:
https://m.media-amazon.com/images/I/C1D0PLjqwwL.pdf

3.0 SR003

3.1. Optical Assembly Verification

Requirement: Verify the Blue LEDs emit wavelength of 445nm for excitation

Verification Type: Test

Expected Date of Completion: 2/4/25

Procedure:

1. Use a spectrometer to measure the emitted spectrum of the LED and verify the desired wavelengths are contained in the emission spectrum, from around 440-450nm.

Data sheet:

4.0 SR004

4.1. Optical Assembly Verification

Requirement: The emission filter will only allow wavelengths between 500 and 550 nm.

Verification Type: Test

Expected Date of Completion: 2/4/25

Procedure:

1. Using a white LED as a broad spectrum source, the light will be placed in front of the spectrometer fiber and its spectrum will initially be verified to range from 400 to 700 nm.
2. The white LED will then be placed in front of the filter and the filter will be placed in front of the spectrometer.
3. The spectrometer will be plugged into the tester's computer who will visualize the spectrum produced.
4. The spectrum collected through the filter will be analyzed and verified that the collected light will be within 500 to 550 nm.

Data sheet:

5.0 SR005

5.1. Requirement: The Water Baddies system shall use a wired camera to perform colorimetry with at least a resolution of 1280 x 720.

Verification Type: Inspection

Expected Date of Completion: 1/28/25

Procedure:

1. Check IDL Part 100740 to verify the magnification.

Data sheet:

Sensor:	Sony IMX708
Resolution:	11.9 megapixels
Sensor size:	7.4mm sensor diagonal
Pixel size:	1.4µm × 1.4µm
Horizontal/vertical:	4608 × 2592 pixels
Common video modes:	1080p50, 720p100, 480p120

https://datasheets.raspberrypi.com/camera/camera-module-3-product-brief.pdf

6.0 SR006

6.1. Requirement: The system will weigh less than 25 lbs.

Verification Type: Test

Expected Date of Completion: 4/20/25

Procedure:

1. The system will be placed on a scale.
2. The number will be verified to be less than 25 lbs.

Data sheet:

7.0 SR007

7.1. Requirement: The Water Baddies App shall have a data loss rate of less than 1%.

Verification Type: Test

Expected Date of Completion: 2/23/25

Procedure:

1. In WaterBaddiesPi/bluetoothCreation/baddiesDetection.py on the Raspberry Pi program the getConcentration() function to return a constant value of 5
2. Go to the terminal and activate the bluetoothScript using the followig commands
   ~ cd
   ~ cd Desktop/WaterBaddies/WaterBaddiesPi/bluetoothCreation
   ~ source wbe/bin/activate
   (wbe)~ cd ./bluetoothCreation
   (wbe)~ python3 bluetoothDetection.py
3. Now that the bluetooth is started, open the water baddies app on your device
4. In the top left hand corner click the bluetooth button
5. This will open a bluetooth drawer. Once you see the WaterBaddiesDetectionSystem, click on the tile to connect
6. Once you see the connection message tap out of the side bar by clicking on the main page
7. Wait until you see the “Fetch New Data” button appear
8. When the button appears, click on it
9. Next for the 3 card titled “Inorganics”, “Metals”, and “Microplastics”, select “Show Chart”
10. For each chart, verify the concentrations of each bar on the chart read a quantity of 5
11. Repeat steps 3-10 3 times to verify apps consistency
12. Pass if all read 5, fail if any do not read 5

Data sheet:

SR007			
Referenced VTP Paragraph Number:			
Analysis Referenced (for verification by T/A): T			
Name of Test: Bluetooth Data Loss Test			
Unit Under Test: **Mobile App / Raspberry Pi**			
Name: Part Number: Serial Number:			
Results (Pass / Fail): **PASS**		Date of Test: 2/23/25	
Recording of Test Measurement: 0%	Requirement (SR, with tolerances): The Water Baddies App shall have a data loss rate of less than 1%	Test Equipment Error: N/A	Adjusted Test Limit: N/A
Computations, (Include Analyses Results, if any): N/A			
Signatures: Tester: **Austin Medina** Customer_____			

8.0 SR010

8.1. Electrical Assembly Verification

Requirement: The components shall be rated to 120V 60Hz, and the battery will be compatible with Type A and B outlets

Verification Type: Inspection

Expected Date of Completion: 2/28/25

Procedure:

1. Plug the battery to charge into a wall outlet.

Data sheet:

9.0 SR012

9.1. Requirement: The Water Baddies App shall connect using Bluetooth 5/Bluetooth Low Energy

Verification Type: Demonstration/Inspection

Expected Date of Completion: 2/28/25

Procedure:

1. Connect the Water Baddies App to the Raspberry Pi via Bluetooth and show that there is a transfer of data via Bluetooth
2. Check IDL Part 100420 and Part 100391 to verify the use of Bluetooth Low Energy

Data sheet:
https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf

10.0 SR014

10.1. Requirement: The Water Baddies system shall utilize a 5x5 cm double-layer microfluidic paper-based device.

Verification Type: Inspection

Expected Date of Completion: 2/28/25

Procedure:

1. The paper fluidic device shall be measured using a ruler to confirm 5x5 cm size.

Data sheet:

SR016	
Referenced VTP Paragraph Number:	
Analysis Referenced (for verification by T/A): Test	
Name of Test: Paper Fluidic Water Retention	
Unit Under Test: Paper Fluidic	
Name: Paper Fluidic Part Number:100100	
Results (Pass / Fail): Pass	Date of Test: 2/23/25

Recording of Test Measurement: .75 mL drop dispensed from dropper	Requirement (SR, with tolerances): Paper fluidic shall fully absorb the drop and diffuse the liquid	Test Equipment: N/A: Syringe	Adjusted Test Limit: N/A
Computations, (Include Analyses Results, if any): N/A			
Signatures: Tester: Jameson Brehmer Sponsor: _____			

11.0 SR015

11.1. Requirement: The System shall have 4gb of ram to be able to process and analyze data.

Verification Type: Inspection

Expected Date of Completion: 1/28/25

Procedure:

1. Check IDL Part 100420 to verify ram.

Data sheet:
https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf

12.0 SR016

12.1. Paper Fluidics Verification

Requirement: Channel length will be able to maintain full drop without additional flowing solution while getting equal amount to each reagent well when using a 70 µL drop from a micropipette.

Verification Type: Test

Expected Date of Completion: 2/25/25

Procedure:

1. A 70 µL drop will be dispensed on the loading area of the paper fluidic device
2. The capillary action will be allowed to complete in full until there is no standing liquid in the loading area
3. The reaction areas will then be analyzed using imageJ

Data sheet:

12.2. Electrical Assembly Verification

Requirement: The subsystem shall supply enough power to the motors to dispense water

Verification Type: Test

Expected Date of Completion: 2/25/25

Procedure:

1. Create a python script that runs 2 stepper motors simultaneously, each independently both forwards and reverse.
2. Use the DMM to test the current and voltage. 12 volts and roughly 400-600mA is needed for each motor. Hook the DMM in series with the motors to measure the current. Hook the DMM in parallel with the motors to measure the voltage levels.

Data sheet:

SR0012.2-SR0012.4			
Referenced VTP Paragraph Number:			
Analysis Referenced (for verification by T/A): T			
Name of Test: Water Dropper Test			
Unit Under Test: Water Dropper			
Name: Part Number: Serial Number:			
Results (Pass / Fail): **PASS**		Date of Test: 4/9/2025	
Recording of Test Measurement: 75mL dispense, motor moved, and syringe depressed fully	Requirement (SR, with tolerances): The subsystem shall supply enough power to the motors to dispense water	Test Equirment Error: N/A	Adjusted Test Limit: N/A
Computations, (Include Analyses Results, if any): N/A			
Signatures: Tester: **Austin Medina** Customer_____			

12.3. Dropper Assembly Verification

Requirement: The dropper system will utilize micro syringes to accurately dispense small amounts of water within range

Verification Type: Test

Expected Date of Completion: 2/25/25

Procedure:

1. Micro syringe will be retracted until the base is at 75 uL
2. The micro syringe will be placed into the linear actuator mount
3. Run the test motor script with a distance of 31 * 6
4. The syringe should fully depress, push in the syringe manually to see if it can be lowered more
5. If the syringe can’t be depressed more then it’s a pass
6. Test will be repeated multiple times to ensure accuracy.

Data sheet: See 12.3

12.4. Software Assembly Verification

Requirement: The software shall spin the dropper for enough time for 70 μL of water to be dispensed

Verification Type: Test

Expected Date of Completion: 2/25/25

Procedure:

1. Initiate dropper release sequence and drop water into a graduated microcentrifuge tube or other container with high volumetric resolution and verify it is the correct amount.

Data sheet: See 12.3

13.0 SR018

13.1. Requirement: The Water Baddies colorimetry subsystem shall have a white spectrum LED.

Verification Type: Test

Expected Date of Completion: 2/4/25

Procedure:

1. The white LED spectrum will be produced using a spectrometer.

Data sheet:

14.0 SR021

14.1. Mobile App Verification

Requirement: Shall be built using haptic feedback and accessibility features

Verification Type: Demonstration

Expected Date of Completion: 2/28/25

Procedure:

1. Modify the ReadValue function in the genericCharacteristic class on the Raspberry Pi to return a constant value of 5
2. Start the bluetooth service on the Raspberry Pi

3. On the app, connect to the Water Baddies Detection System
4. Click “Fetch New Data” when prompted
5. Write Pass/Fail in the data sheet, pass if the following and fail if any of the 3 did not occur
    a. Warning message appeared
    b. Warning announced outloud
    c. Phone buzzed

Status: Pass

15.0 SR022

15.1. Requirement: The Water Baddies system shall make use of 2 ml reloadable syringes to accept test species into the system.

Verification Type: Inspection

Expected Date of Completion: 2/28/25

Procedure:

1. Open the door on the device and remove the syringe from its holder.
2. Fill the syringe with water and prime the syringe by depressing the plunger to remove excess air.
3. Place the syringe into its holder within the device, ensuring that it is properly secured and aligned with the dropper motors.

Data sheet: Pass

16.0 SR024

16.1. Electrical Assembly Verification

Requirement: The battery will have enough power for it to last 1 hour

Verification Type: Analysis/Test

Expected Date of Completion: 4/20/25

Procedure:

1. Power on the system and set a timer. Keep running the system and executing tests.
2. Time how long it takes for the device to power off and verify the battery lasted for at least 1 hour.

Data sheet:

Water Baddies Data Sheets
SR0024

<table>
<tr><td colspan="4">Referenced VTP Paragraph Number:</td></tr>
<tr><td colspan="4">Analysis Referenced (for verification by T/A): Analysis & Test</td></tr>
<tr><td colspan="4">Name of Analysis: The battery life of the system shall last at least 1 hour.</td></tr>
<tr><td colspan="4">Unit Under Analysis: Electrical Subassembly</td></tr>
<tr><td colspan="4">Name: Raspberry Pi 5, Mini Display, Active Cooler, IR Break Sensor, Adafruit Motor HAT, Adafruit Motor, Servo Motor, Servo Motor Hat, Battery, Door Button
Part Number: 100420, 100430, 100440, 100450, 100460, 100470, 100494, 100496, 100410, 100494</td></tr>
<tr><td colspan="2">Results (Pass / Fail):
Pass</td><td colspan="2">Date of Analysis: 11/20/2024
Date of Test: 04/20/2025</td></tr>
<tr><td>Analysis:
$\frac{8300mAh}{6000mAh}$ * 60min = 83 mins or 1hr and 23 minutes as a conservative estimate of run time.
$t = \frac{8300mAh}{3400mAh} \cdot 60\,\text{min}$ Using our tested current. We get 146 minutes or 1hr and 26 minutes</td><td>Requirement (SR, with tolerances): Minimum Battery run time of 1hr</td><td>Test Equipment Error: Timer</td><td>Adjusted Test Limit N/A</td></tr>
<tr><td colspan="4">Recording of Test Measurement: The batteries lasted 2 hrs and 23 minutes before shutting down.</td></tr>
<tr><td colspan="4">Signatures:

Tester: Tyler Thursby

Sponsor: _____</td></tr>
</table>

Note: analysis is complete above

17.0 SR025

17.1. Mobile App Verification

Requirement: Shall have a bluetooth button that opens a page and displays all available devices

Verification Type: Demonstration

Expected Date of Completion: 2/28/25

Procedure:

1. Open the Water Baddies App and look for the Bluetooth icon in the top left corner

2. If found click on the button and verify the side panel opens up and displays available bluetooth device
3. Fail if either or all above did not occur, pass if both 1 and 2 occurred

Status: Pass

18.0 SR027

18.1. Requirement: The Water Baddies System shall have an operational temperature between 15-35ºC.

Verification Type: Analysis/Test

Expected Date of Completion: 4/20/25

Procedure:

1. Place the device in a place where the temperature is at or below 15c (59f), measure the ambient temperature with a thermometer.
2. Allow the system to be in the environment for at least 10 minutes
3. Run the system 5 times consecutively.
4. Verify that all electrical components are working properly.
5. Place the device in a place where the temperature is at or above 35c (95f), measure the ambient temperature with a thermometer
6. Allow the system to be in the environment for at least 10 minutes
7. Run the system 5 times consecutively.
8. Verify that all electrical components are working properly.

Data sheet:

Water Baddies Data Sheets
SR0027
Analysis Referenced (for verification by T/A): Analysis & Test
Name of Analysis: The Water Baddies System shall have an operational temperature between 15-35C (59-95F).
Unit Under Analysis: Electrical Subassembly
Name: Raspberry Pi 5, Mini Display, Active Cooler, IR Break Sensor, Adafruit Motor HAT, Adafruit Motor, Servo Motor, Servo Motor Hat, Battery, Door Button Part Number: 100420, 100430, 100440, 100450, 100460, 100470, 100494, 100496, 100410, 100494

<table>
<tr><td colspan="2">Results (Pass / Fail):
Pass</td><td colspan="2">Date of Analysis: 11/20/2024
Date of Test: 04/20/2025</td></tr>
<tr><td>Analysis:
<table><tr><td></td><td colspan="2">Operational Temp (Data sheet)</td></tr><tr><td></td><td>Min Temp (°C)</td><td>Max Temp (°C)</td></tr><tr><td>Raspberry Pi 5</td><td>0</td><td>70</td></tr><tr><td>Stepper Motor</td><td>-20</td><td>50</td></tr><tr><td>Adafruit Hat</td><td>N/A</td><td>N/A</td></tr><tr><td>Display HAT</td><td>N/A</td><td>N/A</td></tr><tr><td>IR Break Beam</td><td>N/A</td><td>N/A</td></tr><tr><td>Camera</td><td>0</td><td>50</td></tr><tr><td>Microscope</td><td>N/A</td><td>N/A</td></tr><tr><td>LED</td><td>N/A</td><td>N/A</td></tr><tr><td>Battery</td><td>-20</td><td>60</td></tr><tr><td>DC-DC 12V to 5V</td><td>N/A</td><td>N/A</td></tr><tr><td>Door Button</td><td>-40</td><td>40</td></tr></table></td><td>Requirement (SR, with tolerances):
The Minimum of 15C and a Maximum of 35C</td><td>Test Equipment Error:
Thermometer</td><td>Adjusted Test Limit N/A</td></tr>
<tr><td colspan="4">Recording of Test Measurement: Passed the system was able to run at the temperature extremes for 5 times consecutively.</td></tr>
<tr><td colspan="4">Signatures:

Tester: Tyler Thursby

Sponsor: _____</td></tr>
</table>

19.0 SR028

19.1. Requirement: The Water Baddies System’s components shall not exceed 12 volts and 6 Amps.

Verification Type: Analysis/Test

Expected Date of Completion: 4/03/25

Procedure:

1. Use a DMM and measure the maximum current and voltage running through the PI and its components while it's running. Use Another DMM, and power on the motors. Measure the current and voltage going to the motors. Use the maximum voltage and add the two currents to achieve the total current and voltage.

Data sheet:

Water Baddies Data Sheets
SR0028
Analysis Referenced (for verification by T/A): Analysis
Name of Analysis: The Water Baddies System’s components shall not exceed 12 volts and 6

Amps.

Unit Under Analysis: Electrical Subassembly

Name: Raspberry Pi 5, Mini Display, Active Cooler, IR Break Sensor, Adafruit Motor HAT, Adafruit Motor, Servo Motor, Threaded Stepper Motor, Servo Motor Hat
Part Number: 100420, 100430, 100440, 100450, 100460, 100470, 100494, 100495, 100496

Results (Pass / Fail):
Pass

Date of Analysis: 11/20/2024
Date of Test: 04/03/2025

Analysis:

Components	Quantity	Voltage(V)	Current(mA)
Raspberry Pi 5	1	5	3000
Display	1	5	150
Camera	1	3.3	300
Active Cooler	1	5	300
White LED	1	3.3	16
Blue LED	1	5	16
Microscope	1	5	900
Adafruit Motor Hat	4	5	80
Motors	9	12	400
IR Beam Sensor	6	3.3	120
Total:		12	5802

Requirement (SR, with tolerances): The maximum 12V and 6A

Test Equipment Error: Digital Multimeter

Adjusted Test Limit N/A

Recording of Test Measurement:

Electrical Components	Current	Voltage	Power
Raspberry Pi 5 (+ Components)	1.34	5	6.7
Adafruit HAT Motor 1 + 2	1.035	12	12.42
Adafruit HAT Motor 3+ 4	1.035	12	12.42
Total	3.41	12	31.54

Signatures:

Tester: **Tyler Thursby**

Sponsor: _____

Part Drawings

Version 1.0

Document #100750

March 4th, 2025

WATER BADDIES - Microplastic, Heavy Metal and Inorganics Water Detection System for Environmental and Human Health

Team 25040

Brendan Bamberg

Jameson Brehmer

Daniel Knaus

Austin Medina

Alia Nichols

Aidan Talucci

Tyler Thursby

Included are parts of assemblies manufactured by the Water Baddies team. Purchased parts will be defined elsewhere via cut sheets if available and may have similar placeholders inserted in assemblies for visual aid and dimensioning.

Link to Folder with All Part Drawings: CAD Drawings