

```
|||||| HEAD ===== ||||| HEAD  
||||| HEAD ===== |||||| Stashed changes |||||| 9ae2eabe5e6d430ba55097c3c2d1ff9519d985e5  
||||| refs/remotes/origin/main
```

Theta Beta Engineer Project Report

Foundation Color Identifier and Dispenser
University of California, Irvine

Akhil Nandhakumar, Allyson Lay, Dalen Avrin Smith,
Elizabeth Yancey, Emma Shin, Harmeet Singh, Ival Momoh,
Jay Kim, Richard Tokiyeda, Victoria Sun

November 17, 2025

¹ **Contents**

² **List of Figures**

³ **List of Tables**

⁴ Chapter 1

⁵ Abstract

⁶ The Foundation Identifier and Dispenser aims to develop a machine that is able to extract
⁷ samples from a picture of a human to determine the shade of their skin, allowing the machine
⁸ to dispense a corresponding foundation shade. The results produced should be both accurate
⁹ and reproducible. Equipped with computer vision libraries and color correction algorithms,
¹⁰ the system allows the user to take an picture alongside a reference color sheet, which has
¹¹ colors of known values. These captured values are processed to correct both camera bias,
¹² and lighting correction so that results may remain consistent regardless of lighting conditions
¹³ during image capture. The system converts RGB values to LAB values, which are higher in
¹⁴ accuracy in physical color mixing, as opposed to RGB, which is used to describe pixel colors.
¹⁵ The program calculates how much of each color is needed to recreate the user's skin pigment.
¹⁶ These pigments are then dispensed via a mechanical system comprised of a Raspberry Pi,
¹⁷ servo motors, and syringes. This project demonstrates an application of computer vision in
¹⁸ the cosmetic market to alleviate the burden of overconsumption and promote inclusivity.

¹⁹ Chapter 2

²⁰ Introduction

²¹ 2.1 Research

²² 2.1.1 Background of Problem

²³ Testing foundation colors can be a frustrating experience for many, who are unable to
²⁴ find the perfect balance. At the end of the day, no line of foundation can realistically provide
²⁵ colors that cater to every possible skin tone. The seemingly unresolvable desire for a perfect
²⁶ shade leads many makeup users to spend hundreds on shades that are "close enough." This
²⁷ leads to lots of waste, not just in money, but in bottles thrown out after purchase because
²⁸ the match was ultimately unsatisfying.

²⁹ The beauty industry produces 120 billion packaging units per year and 95% of these units are discarded, as opposed to recycled [?]. In addition, traditionally a custom skin matched foundation can cost anywhere from \$60-100 per bottle, which leaves the consumers with the dilemma of whether they should take the gamble on the bottle that's just almost right versus breaking their wallet on a hand matched bottle. Our custom mixer machine offers the same accuracy in skin tone while also promoting cheaper makeup, sustainability, and waste-reduction.

³⁶ Our foundation color picker abandons the concept of creating a set of discrete skin tones to choose from, instead, opting for custom mixed shades depending on the skin color detected by a picture. It also offers the unique ability to sample a shade without having to commit to a full sized bottle.

⁴⁰ 2.1.2 Existing Solutions

⁴¹ BoldHue provides an option for an AI powered foundation mixer. It matches skin tone through a smart wand that detects color. What they promised was millions of shades and the ability to store user's shades in profiles. However, their color detection methods provide room for lots of error because color isn't perceived the same depending on the lighting of the room, as well as the high cost of their machine.

⁴⁶ Our product will have built in lighting correction that removes biases from the camera and uses the Macbeth Color Reference sheet as a set of control colors. The reference sheet

48 is what will allow the machine to recalibrate it's understanding of what colors are being
49 percieved.

50 **2.2 Design Choices**

51 **2.2.1 Past Considerations and Scrapped Plans**

52 **Off-the-Shelf Syringe Pump System**

53 An early design decision we explored was to use commercially available syringe pump
54 system available for purchase from a third party. This idea was ultimately abandoned due
55 to bulkiness of commercial pumps, as well as high cost. Although designing our own pump
56 system required more work from our mechanical team to design the system from scratch,
57 relying on an existing system would have limited our freedom, increased our costs, and taken
58 away from the educational value of the project. Designing a custom syringe pump offered
59 hands-on learning experiences in CAD design.

60 **Quick-Swap Syringe Mounting Attachment**

61 We also considered designing the Syringe housing to allow users to detatch and fill syringes
62 in between uses. This feature was ultimately unnecessary as the syringes don't need to be
63 manually pumped. As long as the moters can rotate the opposite direction, the syringes dont
64 have to be removed to retract. The final design eliminated the syringe-swapping mechanism
65 in favor of refilling via holding a paint vial under the syringe and letting the machine retract
66 the pump to draw in fluid.

67 **Single-Piece Printed Housing**

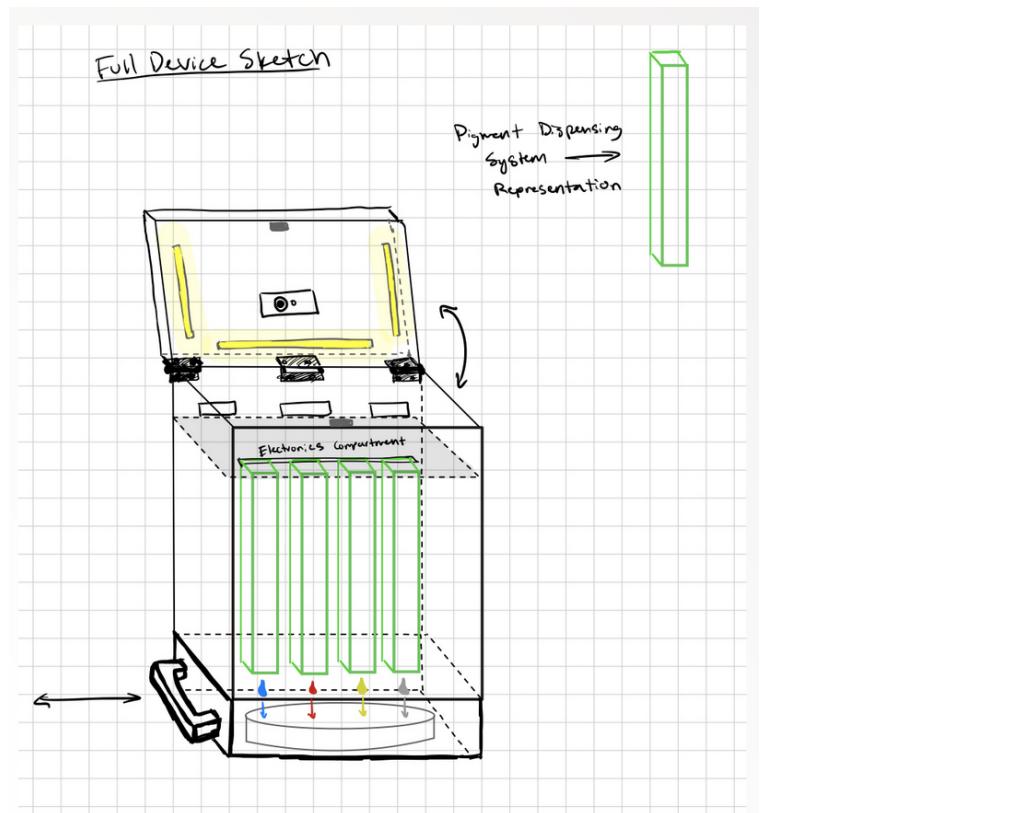
68 Our initial housing was a single piece. However, most 3D printers that were available
69 have a maximum build volume of 256 mm x 256 mm x 256 mm, and the housing dimensions
70 we had for the initial design was too close to this limit. Printing the structure in a single
71 piece also risks expensive failed prints, warping, and geometric constraints. We shifted to
72 a housing unit comprised of three interlocking sections, with custom brackets to join the
73 segments securely.

74 Chapter 3

75 Hardware Design and Specifications

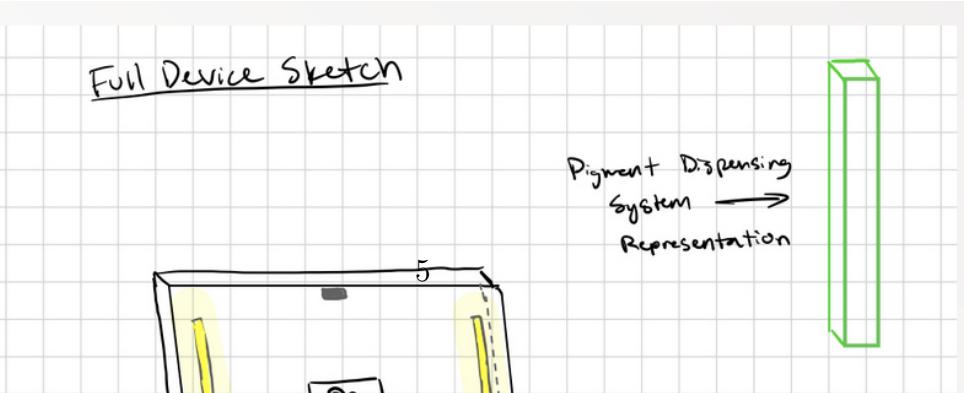
76 3.1 First Iteration

77 3.1.1 Initial CAD Model and Hand Sketches



<<<< HEAD

===== <<<



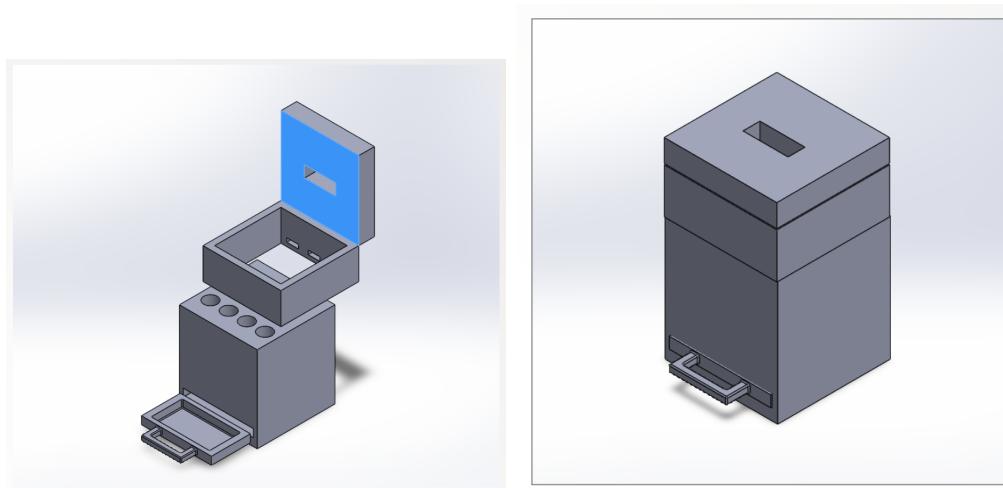


Figure 3.2: First CAD Models: Housing

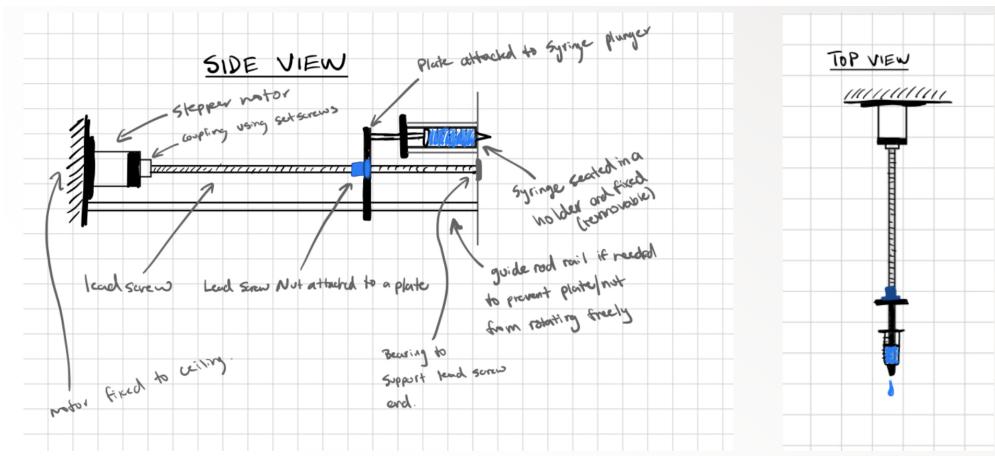


Figure 3.3: First Sketch: Dispenser Systems



Figure 3.4: First CAD Models: Dispenser Systems

78 Our first CAD Models were very simple and provide the most general idea we had at
79 our projects conception. We made large changes to the design of the housing because the
80 dimensions would have been much wider if we lined the syringes up, as shown in Figures 3.1
81 and 3.2.

82 The preliminary sketch we have of the dispenser system has stayed consistent throughout
83 development.

3.2 Second Iteration**3.2.1 Updated CAD Models**

Assembly

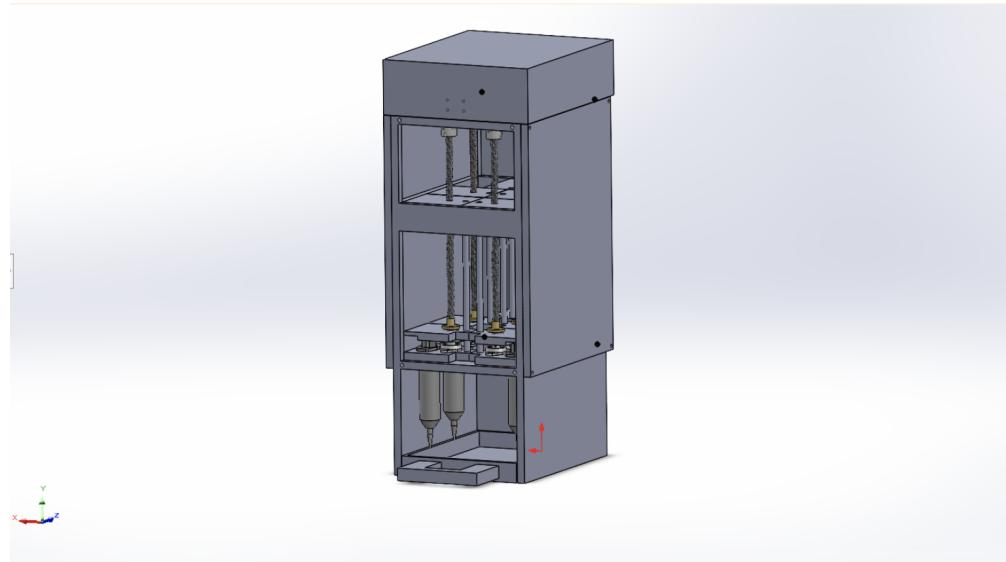


Figure 3.5: Second Iteration: Full Assembly

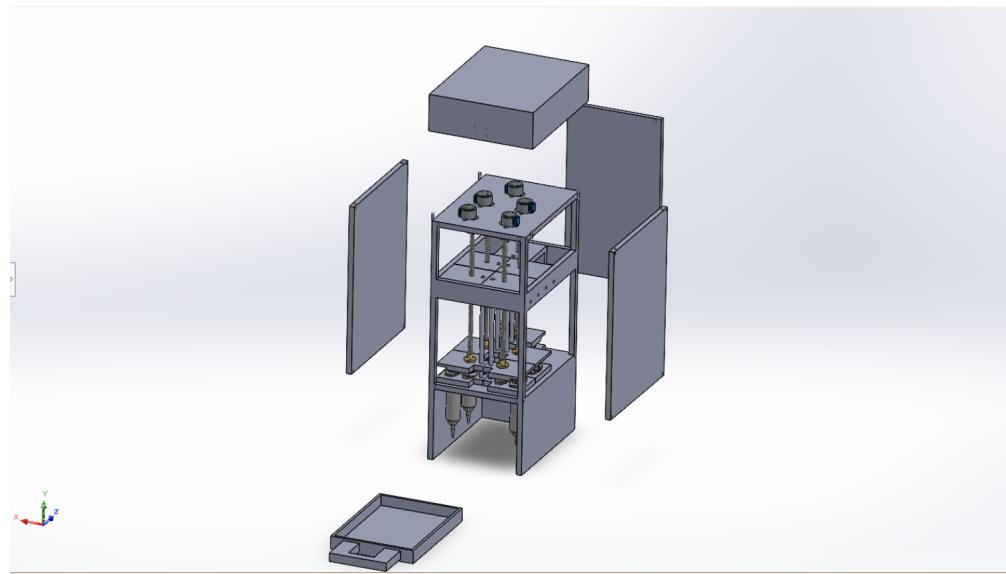


Figure 3.6: Second Iteration: Exploded View

87

Housing

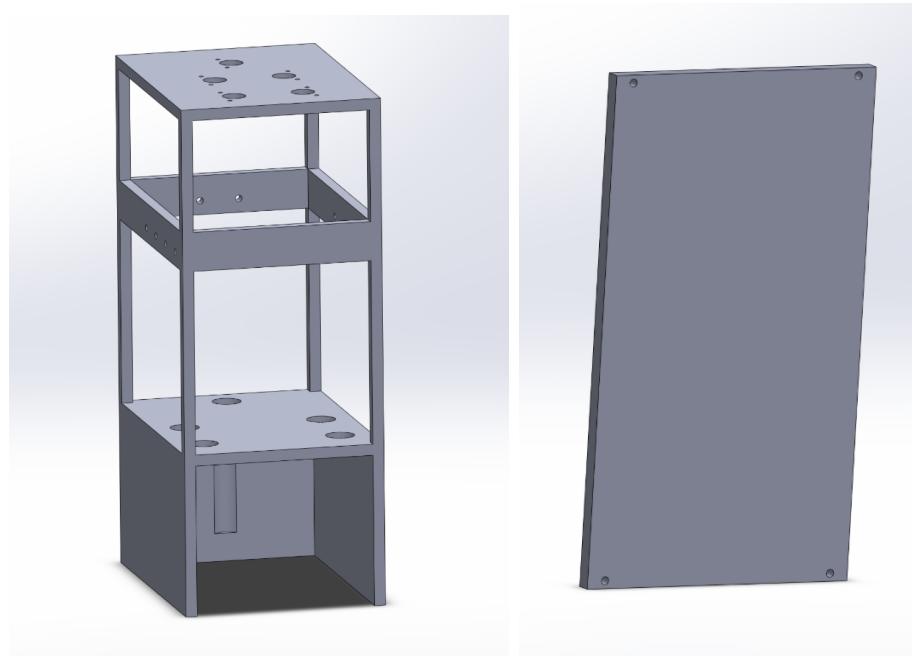


Figure 3.7: Second Iteration: Housing Skeleton and Walls

88 The housing skeleton is what holds the syringes in place while the walls create an enclosure
89 so that the internal structure may be protected and hidden from users. The main concern
90 with this iterations design was with the housing skeleton's stability. Since we were 3D
91 printing all parts for the demo, the empys spaces could be flimsy, so the final iteration
92 includes a redesign.

93 **Dispenser**

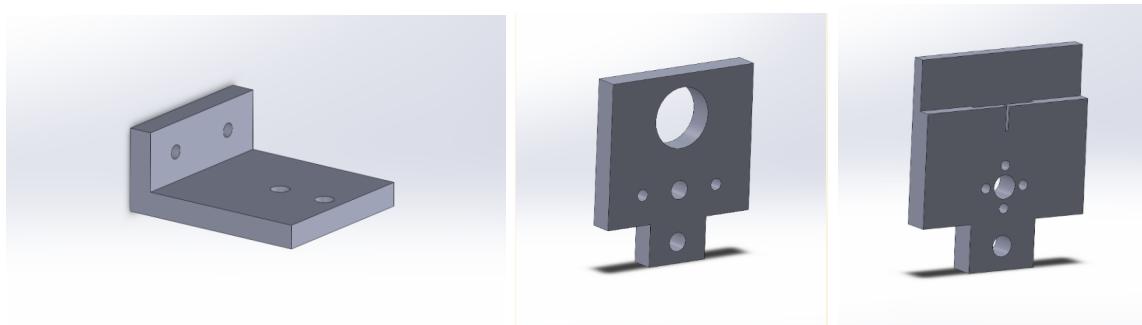


Figure 3.8: Second Iteration: Brackets for Dispenser System

94 These brackets hold and control the dispenser system. The first L-shaped Bracket holds
95 the guide shaft in place. The other two are for stabilizing and pushing the syringe plunger
96 down. This second iteration is missing one of the parts but it will be shown in the next
97 section, along with final drawings of all parts.

98 3.3 Third Iteration

99 <<<< HEAD ===== <<<< HEAD

100 3.3.1 Final CAD Models

101 *Assembly*

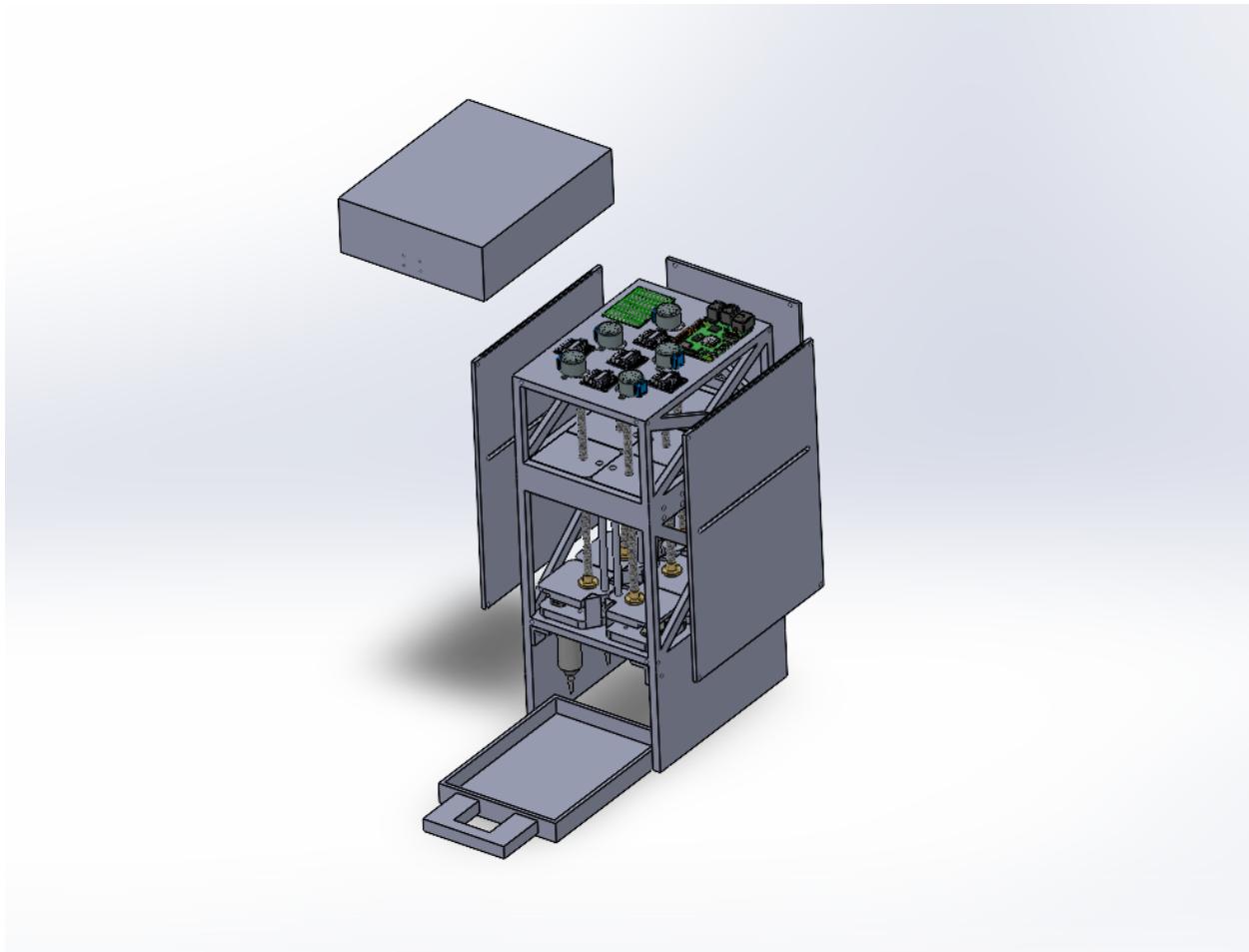


Figure 3.9: Final: Full Assembly *Exploded*

102

Housing

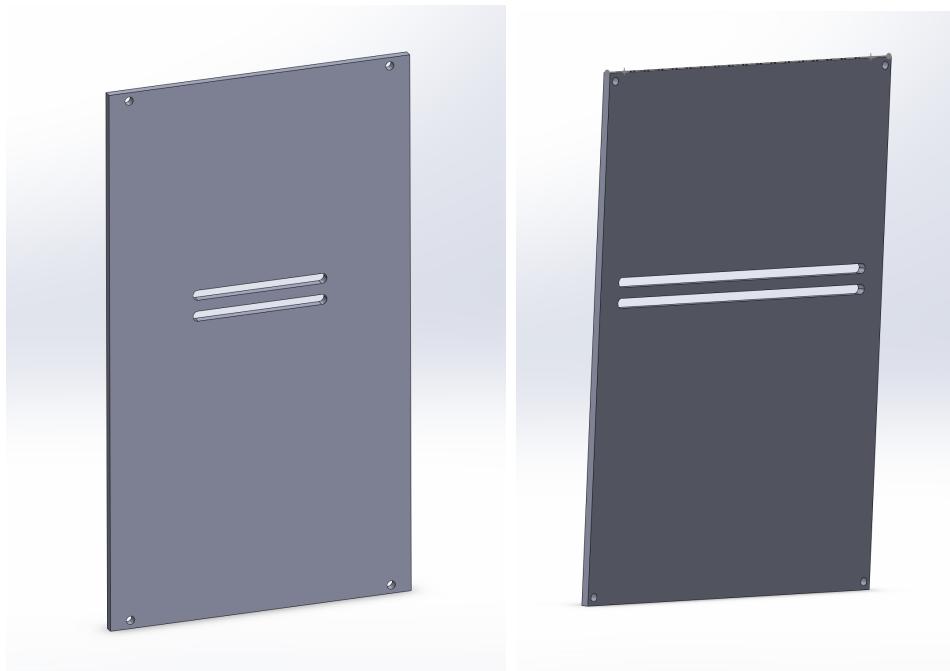


Figure 3.10: Housing: Walls

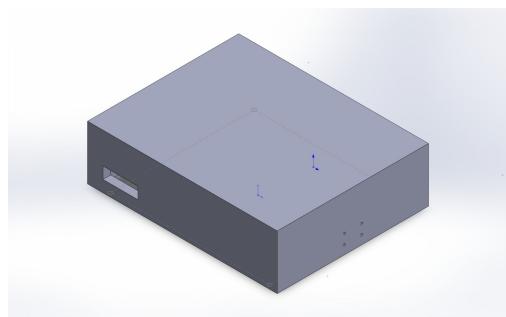


Figure 3.11: Housing: Lid

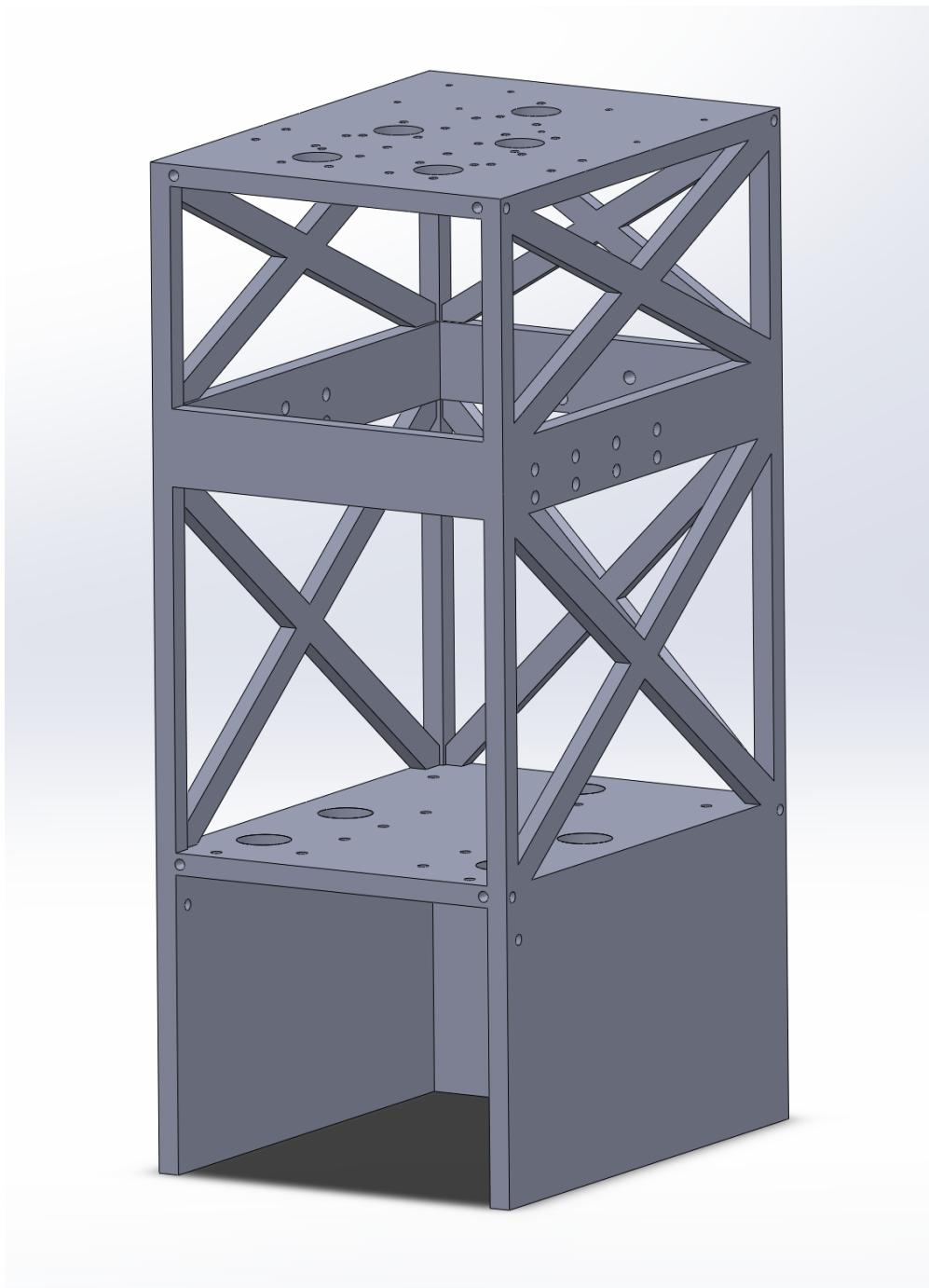


Figure 3.12: Housing: Skeleton

¹⁰³ The Housing walls were altered to include a slit because there are some screws in the
¹⁰⁴ skeleton of the product that would press up against the walls if we kept the second iteration's
¹⁰⁵ design. The slit was created to give the screws space and ensure that the components inside
¹⁰⁶ are not putting pressure on each other.

107

Dispenser

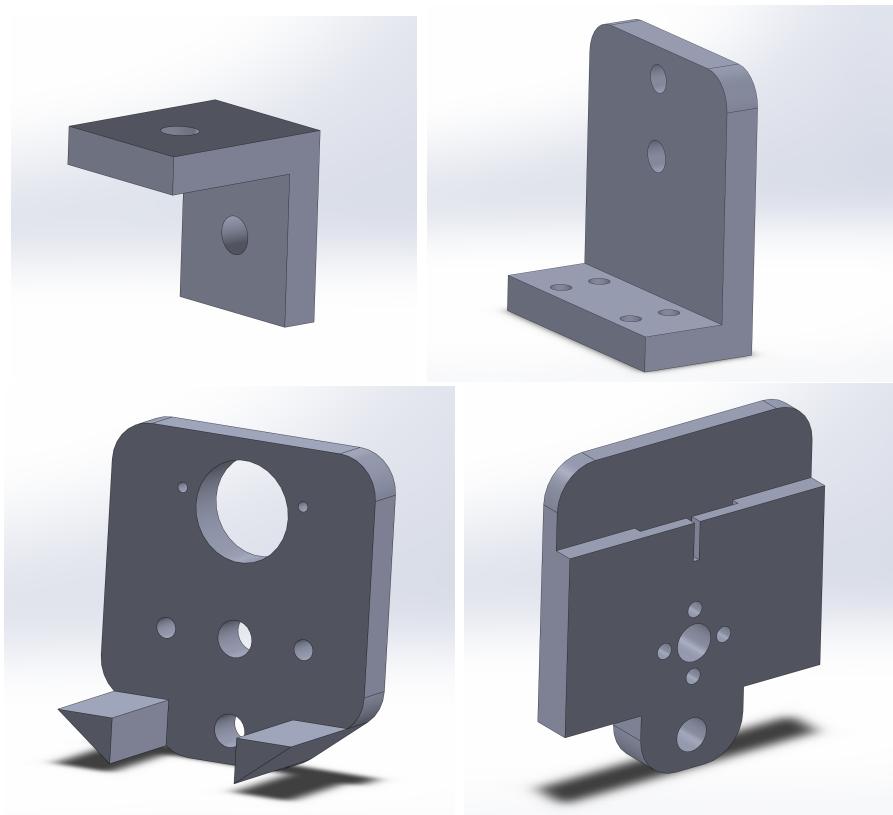


Figure 3.13: Dispenser System : Brackets - for connecting to housing, guide shaft, stabilizing syringe, and pushing down syringe plunger

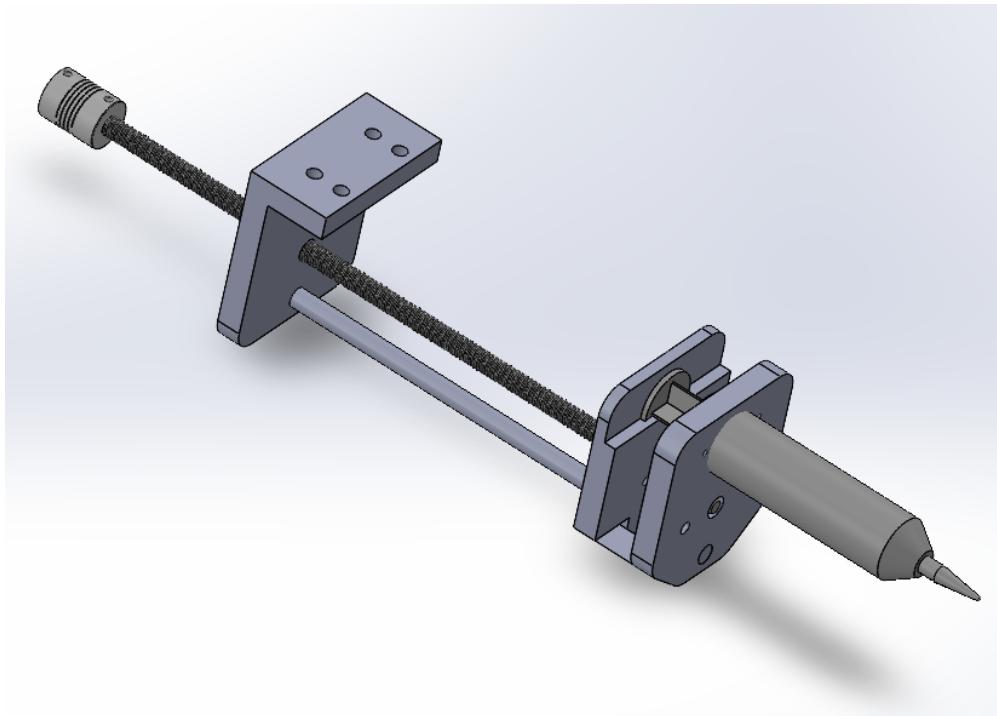


Figure 3.14: Dispenser System : Full assembly

108 The final iteration finalizes all brackets needed for the dispenser system, and includes a
109 complete view of all the parts put together. We have included a L-bracket that connects the
110 dispenser to the casing, an L bracket that holds the guide rod, a syringe bracket to stabilize
111 and hold the syringe in place, and finally, the plate that holds the plunger, and moves with
112 the screw to push the plunger down.

113 **3.3.2 Final Manufacturing Drawings for Custom Parts**

114 ====== <<<< Updated upstream

115 **3.3.3 Final CAD Model**

116 **3.3.4 Final Manufacturing Drawings for Custom Parts**

117 **3.3.5 Commentary**

118 ====== >>>> refs/remotes/origin/main

119 **3.3.6 Final CAD Models**

120 *Assembly*

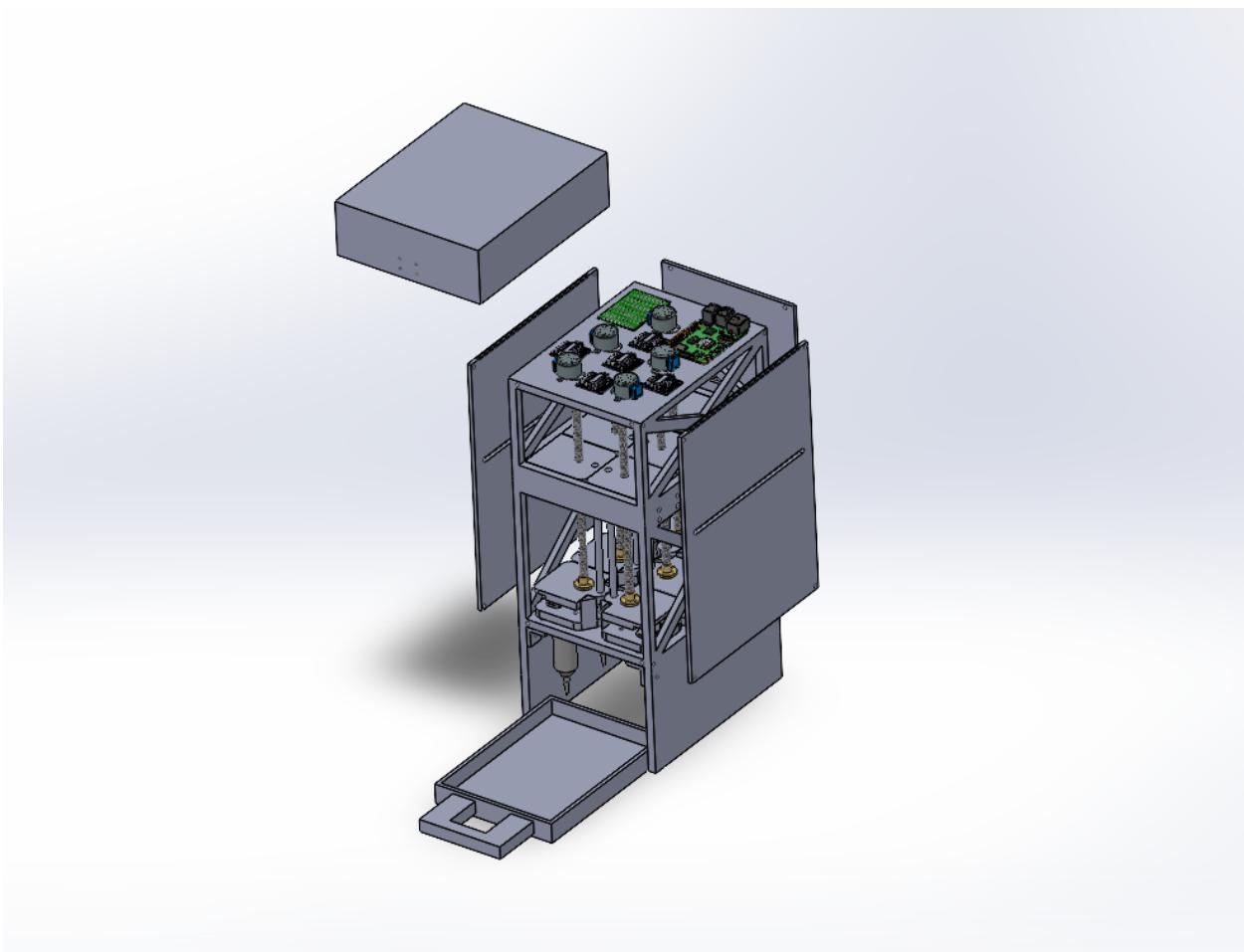


Figure 3.15: Final: Full Assembly *Exploded*

₁₂₁ »»»> 9ae2eabe5e6d430ba55097c3c2d1ff9519d985e5

₁₂₂ **Housing**

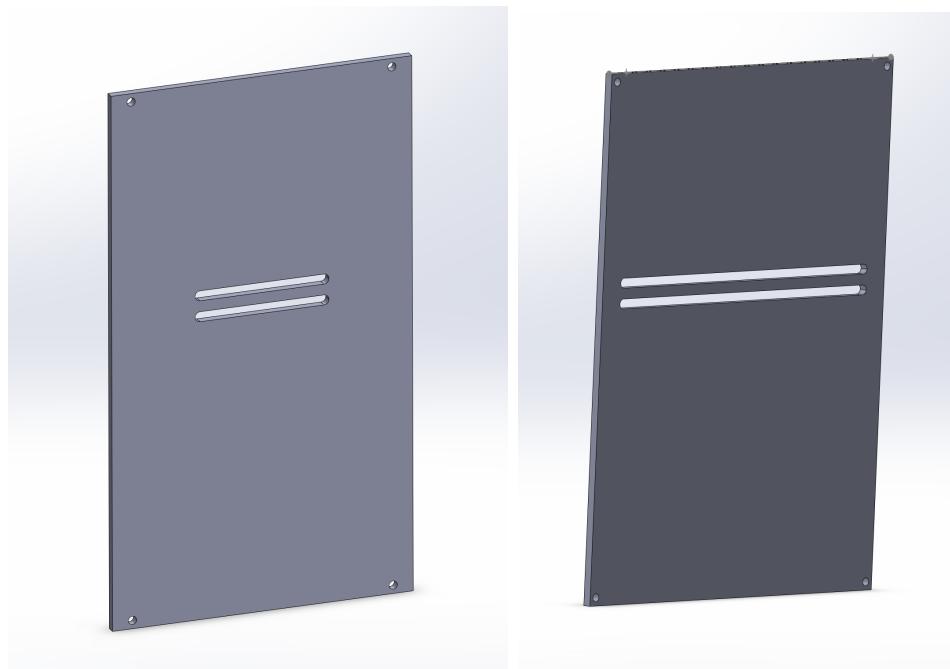


Figure 3.16: Housing: Walls

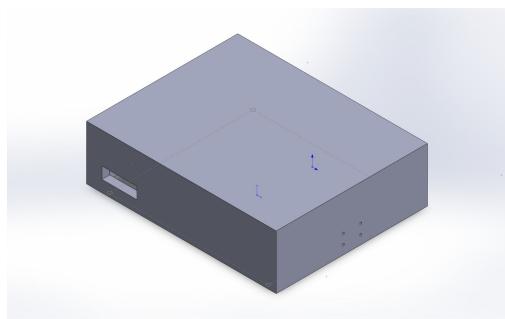


Figure 3.17: Housing: Lid

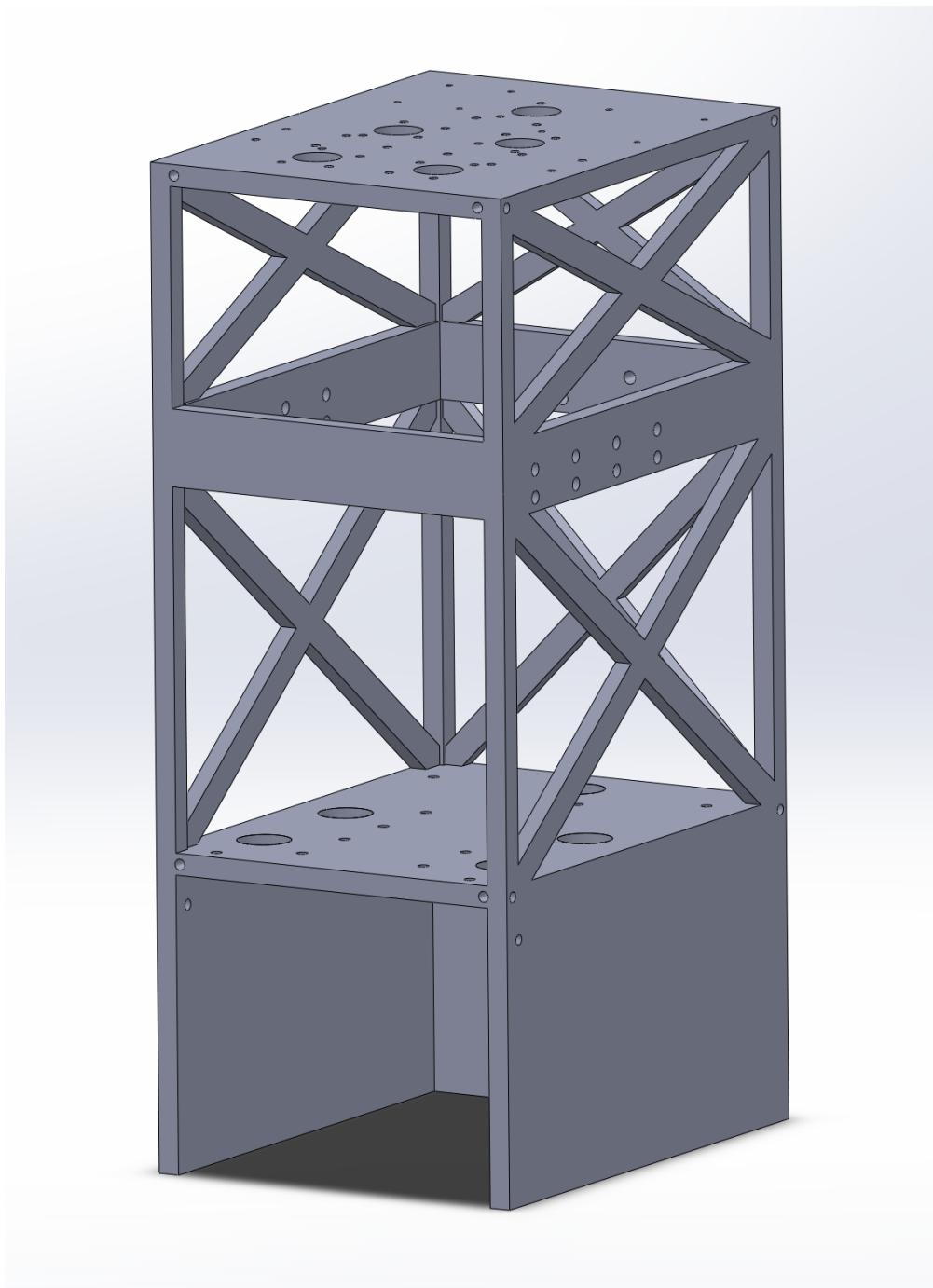


Figure 3.18: Housing: Skeleton

¹²³ The Housing walls were altered to include a slit because there are some screws in the
¹²⁴ skeleton of the product that would press up against the walls if we kept the second iteration's
¹²⁵ design. The slit was created to give the screws space and ensure that the components inside
¹²⁶ are not putting pressure on each other.

127

Dispenser

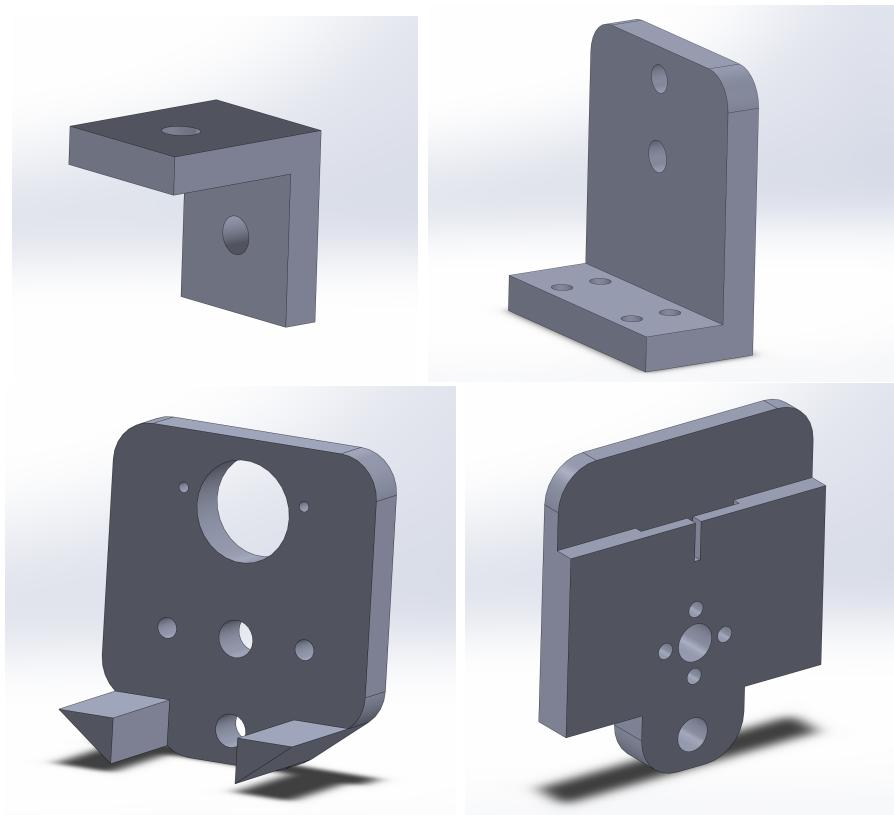


Figure 3.19: Dispenser System : Brackets - for connecting to housing, guide shaft, stabilizing syringe, and pushing down syringe plunger

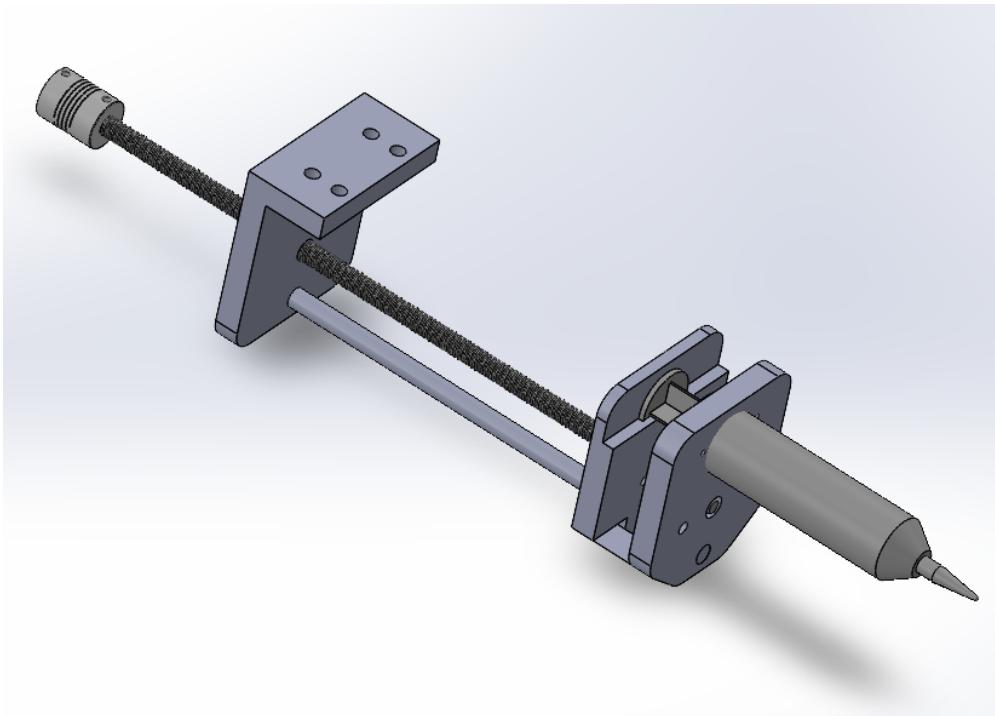


Figure 3.20: Dispenser System : Full assembly

128 The final design consists of four primary components: an L-bracket connecting the dis-
129 penser to the casing, a secondary L-bracket securing the guide rod, a syringe bracket that
130 stabilizes and positions the syringe, and a plunger plate driven by the lead screw to depress
131 the syringe plunger.

132 The final iteration incorporates several refinements across all components. Mounting
133 brackets and L-brackets were redesigned with rounded edges to increase clearance between
134 moving elements. The lower syringe bracket was modified to include two vertical extrusions
135 that act as contact surfaces for limit switches. Additionally, mounting holes for fasteners
136 were added throughout the assembly to improve structural stability and ease of integration.

137

¹³⁸ 3.3.7 Final Manufacturing Drawings for Custom Parts

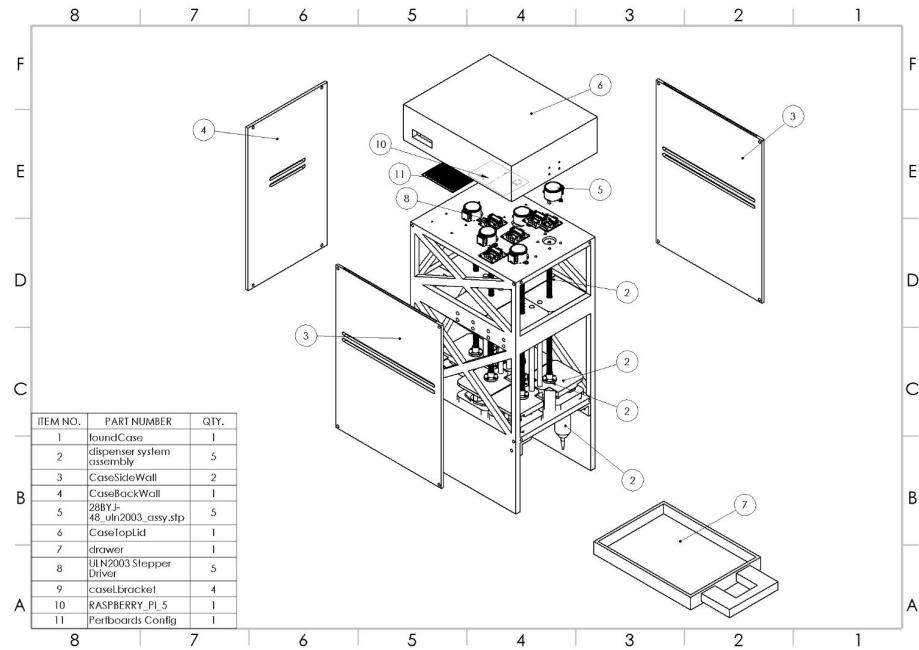


Figure 3.21: Full Assembly Exploded View

139

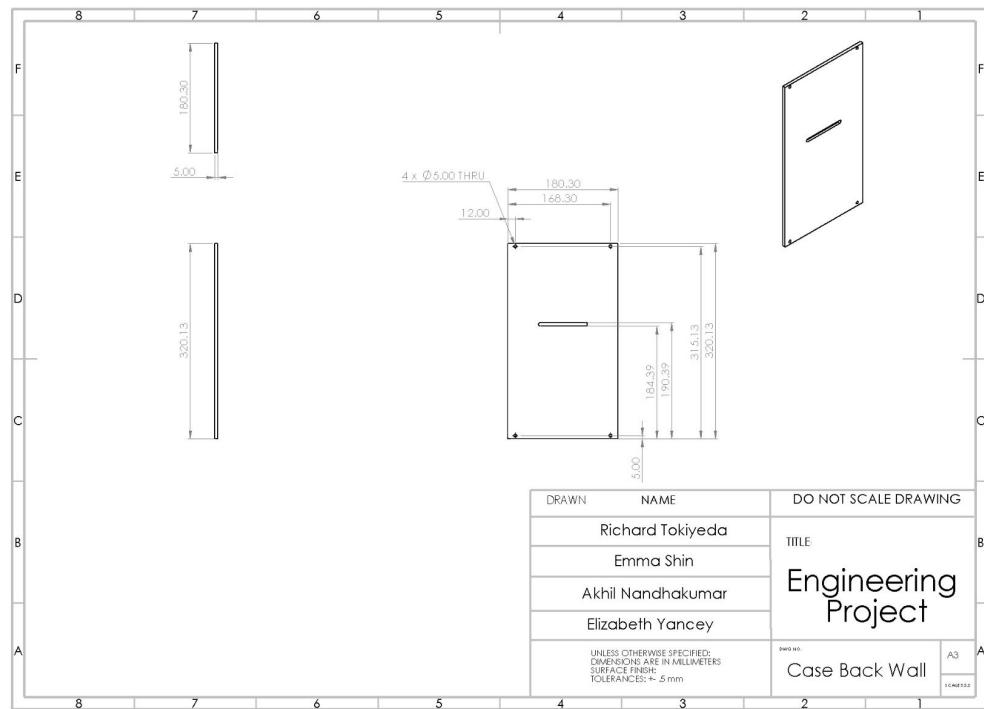
Housing

Figure 3.22: Housing: Walls - Back

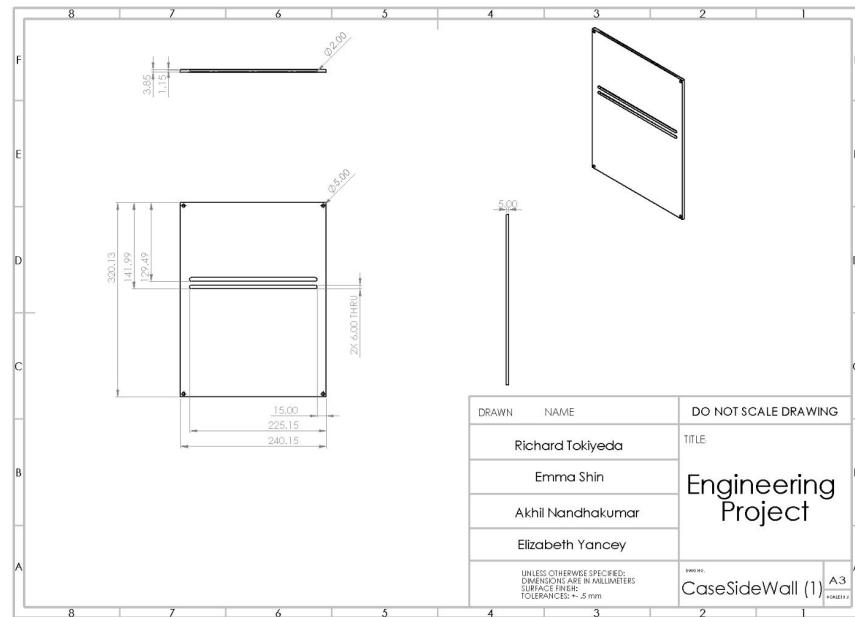


Figure 3.23: Housing: Walls - Side

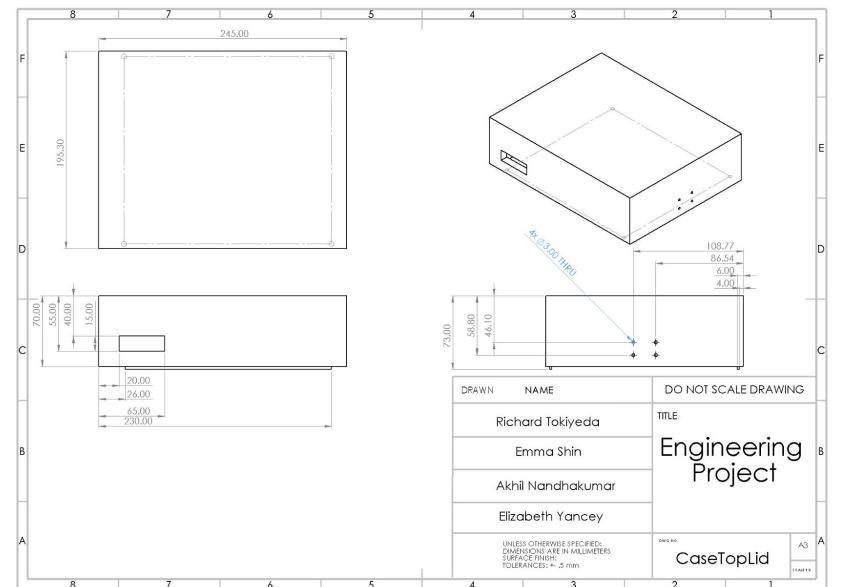


Figure 3.24: Housing: Lid

140

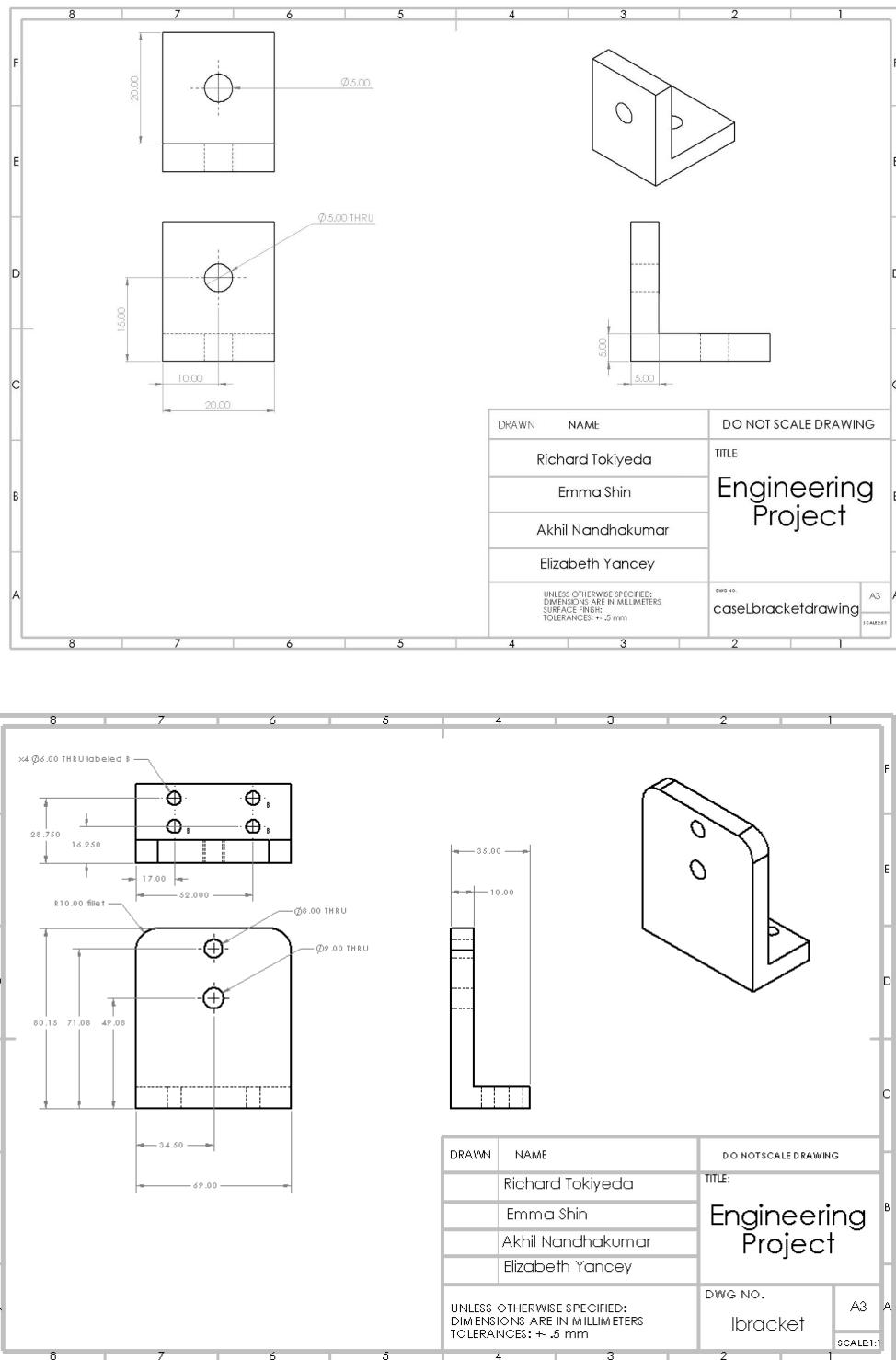
Dispenser

Figure 3.25: Dispenser System : L-Brackets

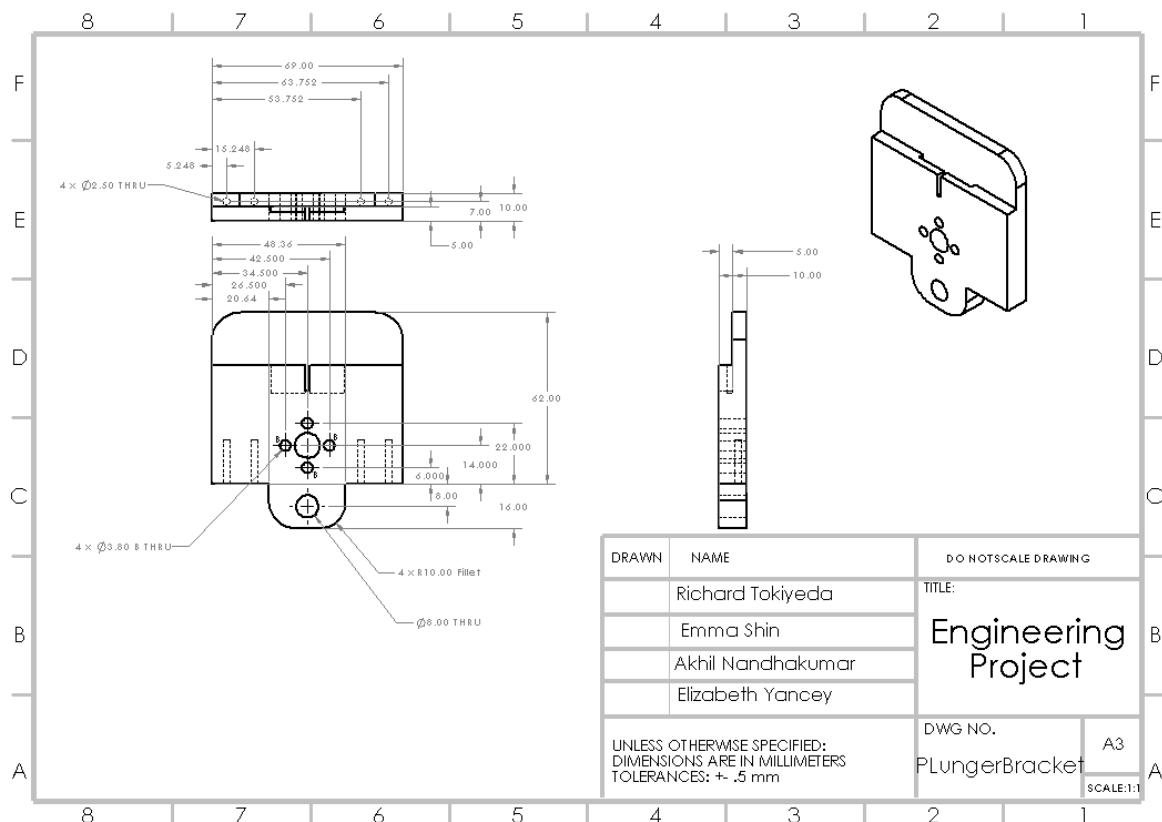
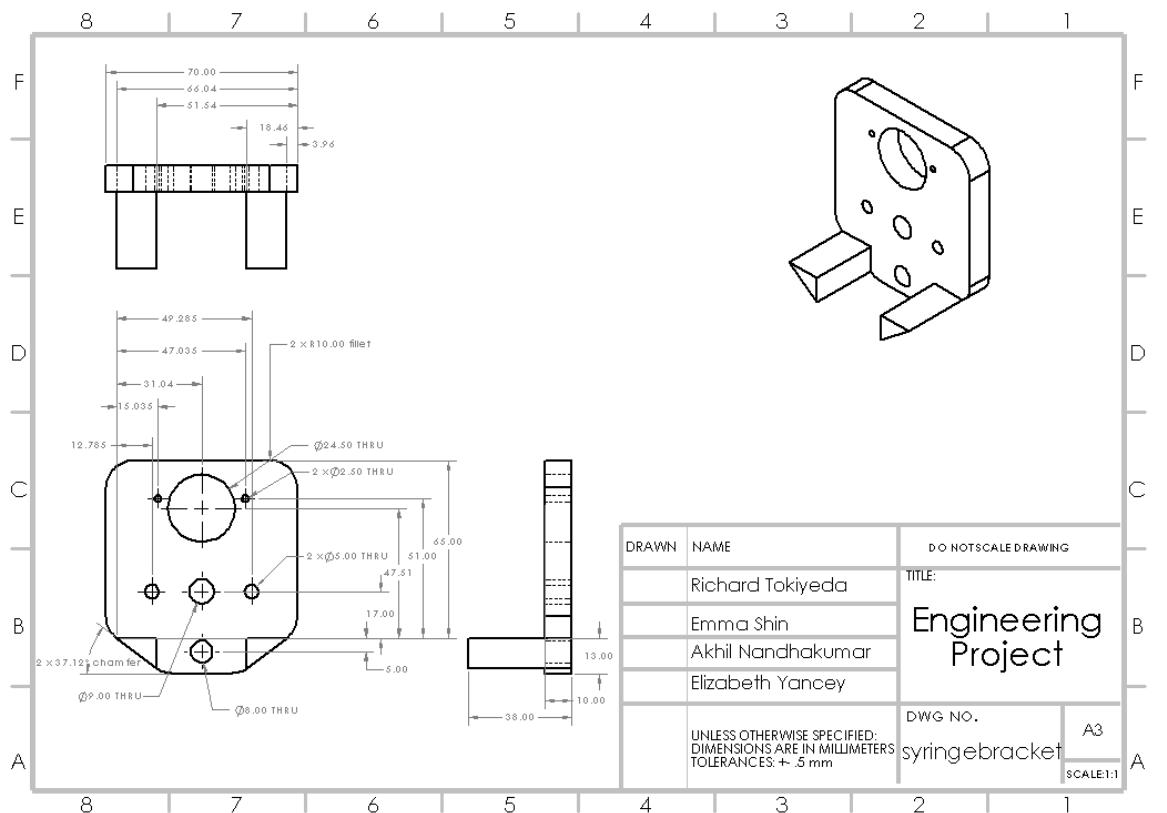


Figure 3.26: Dispenser System : Syringe Brackets
24

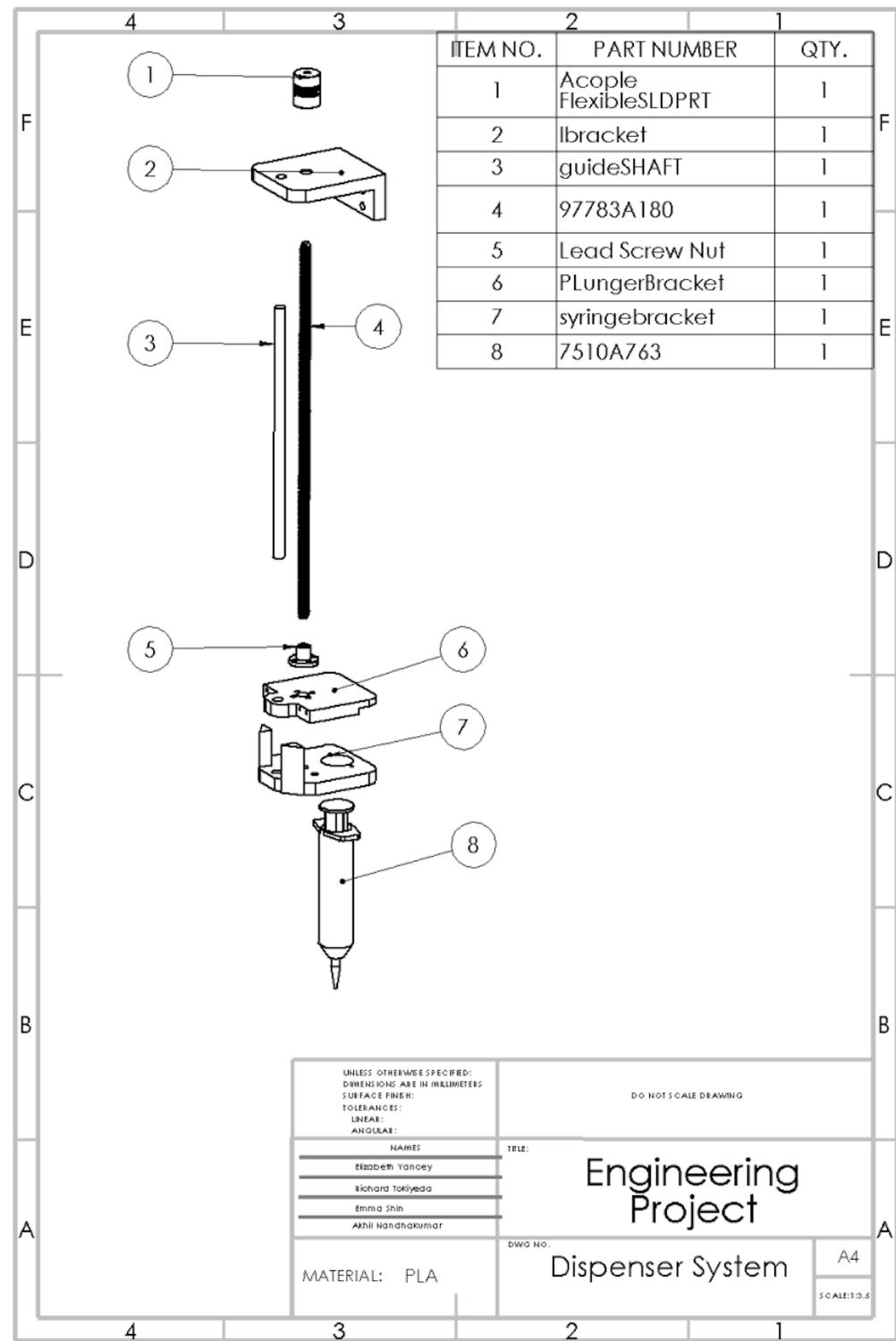


Figure 3.27: Dispenser System : Full assembly Exploded View

¹⁴¹ Chapter 4

¹⁴² Software Design

¹⁴³ 4.1 First Iteration

¹⁴⁴ 4.1.1 Preliminary Diagrams and Psuedocode

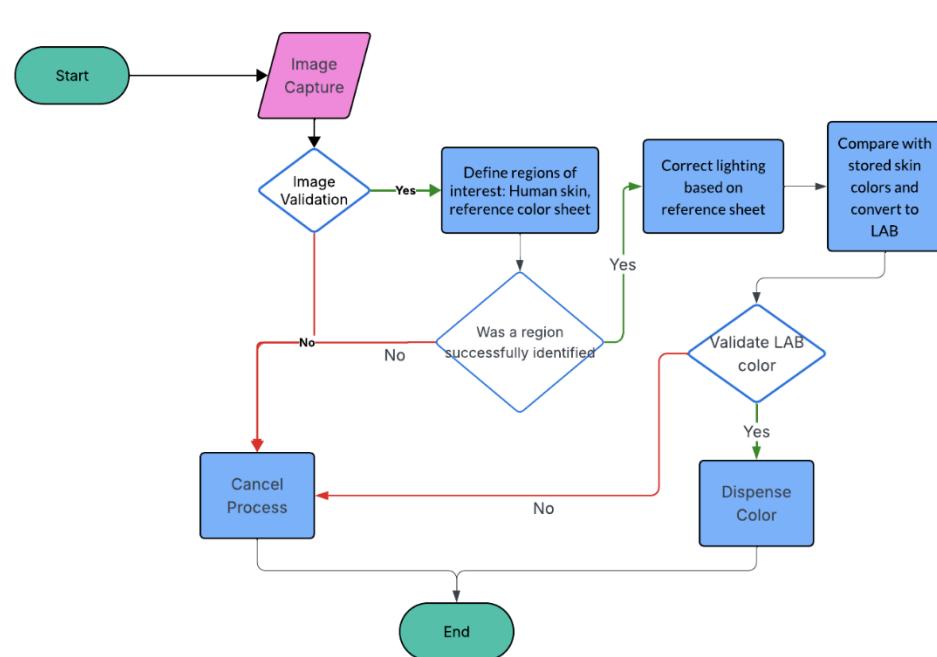


Figure 4.1: Initial Flowchart for Control Logic

¹⁴⁵ <<<< HEAD ===== <<<< HEAD

¹⁴⁶ **4.1.2 User Flow Diagram**

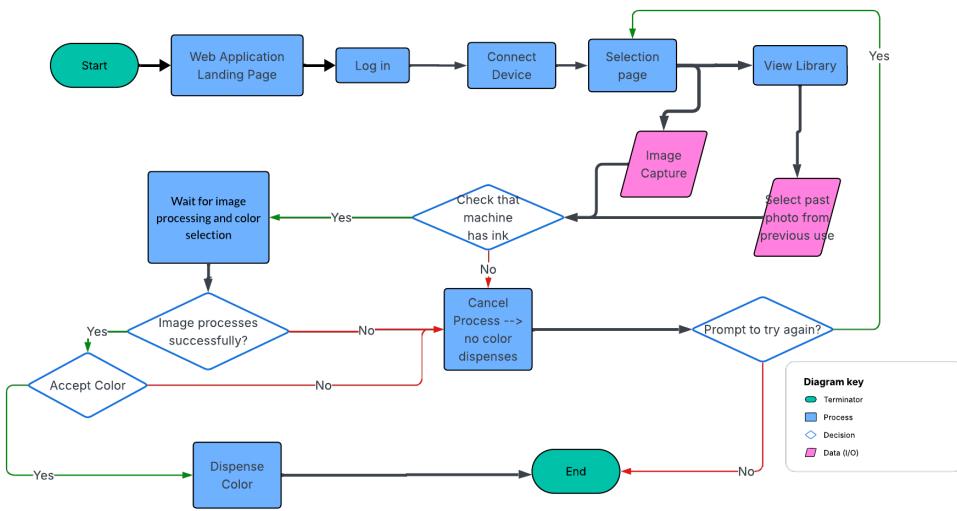


Figure 4.2: User Flow Diagram

¹⁴⁷ **4.1.3 Techstack**

- ¹⁴⁸ Operating systems: Linux on Raspberry Pi
- ¹⁴⁹ Libraries: OpenCV, NumPy, Pandas, Haar Cascade Classifier
- ¹⁵⁰ Frameworks: React
- ¹⁵¹ Languages: Python

152 4.2 Second Iteration

153 4.2.1 Lo-fi/Mid-fi Designs

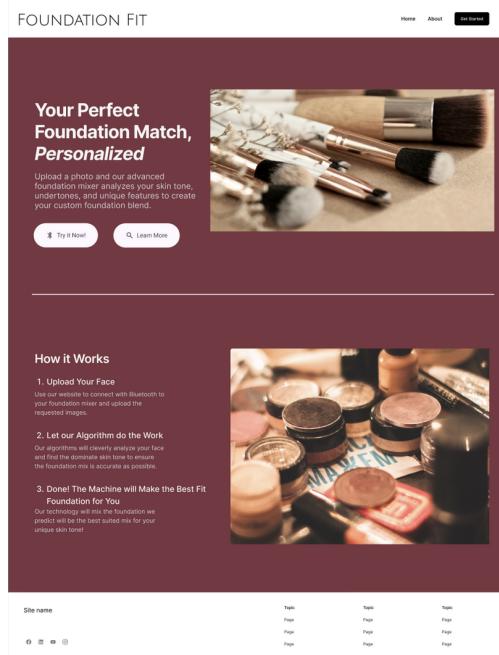


Figure 4.3: Initial Figma Mockup: UI/UX - Landing page.

154 ===== «««< Updated upstream »»»> 9ae2eabe5e6d430ba55097c3c2d1ff9519d985e5

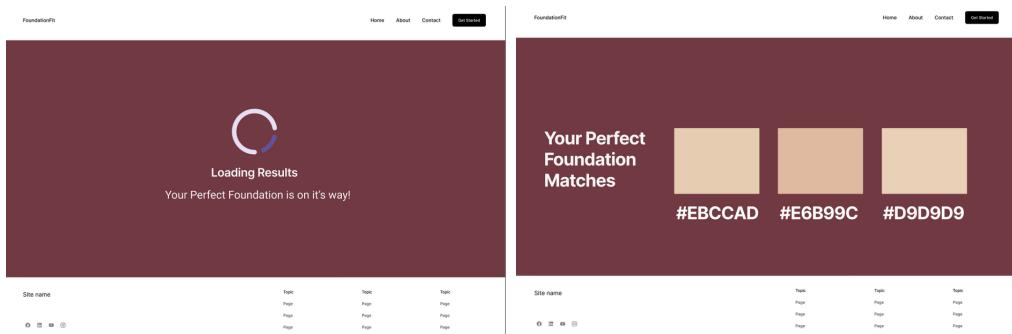


Figure 4.4: Initial Figma Mockup: UI/UX - Loading the foundation shades.

155 4.2.2 Basic Logic and Functionality

```

# 1: Color Bias Adjustment and Sampling
```python
def IMAGE_PROCESSING(SAMPLE_PICTURE) :
 #INPUT: SAMPLE_PICTURE - image captured by camera
 #OUTPUT: LAB_values - color values compatible with paint mixing

 ### Get/Validate image containing skin region
 ### & reference colors/control sheet

 if SAMPLE_PICTURE is EMPTY/NOT_DETECTED :
 RAISE_ERROR "Image captured failed or canceled"
 return NULL

 # initial samples contain gamma-corrected RGB values
 # use OpenCV region of interest rectangle

 SKIN_SAMPLE = list of pixel RGB values that constitute a skin region from SAMPLE_PICTURE
 CONTROL_SAMPLE = list of pixel RGB values that encompass the reference color sheet SAMPLE_PICTURE

 # check whether the reference sheet is present or the image is too dark/light

 if CONTROL_SAMPLE is EMPTY/NOT_DETECTED :
 RAISE_ERROR "Control sheet not detected. Take picture with color reference sheet in a well-lit room."
 return NULL

 # get average gamma-corrected RGB codes for the skin region and control

 SKIN_RGB_AVG = calculate average of SKIN_SAMPLE
 CONTROL_MEASURED_VALUES = list of averages of CONTROL_SAMPLE

 # get linear RGB codes from gamma-corrected RGB codes
 # allows linear operations to be performed on the codes

 SKIN_LINEAR_RGB = apply reverse gamma-correction to SKIN_RGB_AVG
 CONTROL_LINEAR_RGB = list of linearized RGB codes from CONTROL_MEASURED_VALUES

 # find difference in the control input RGB codes and stored RGB codes

 CONTROL_KNOWN_VALUES = load("CONTROL_DATABASE.csv") and linearize the codes
 CONTROL_TRANSFORM = find transformation matrix that makes CONTROL_LINEAR_RGB = CONTROL_KNOWN_VALUES * CONTROL_TRANSFORM

 # apply difference to the values found in the skin region for lighting correction

 XYZ = apply CONTROL_TRANSFORM to SKIN_LINEAR_RGB

 # convert rgb value to lab for later processing

 LAB_VALUES = convert XYZ color space to LAB(XYZ)

 for element in LAB_VALUES :
 if element is OUT_OF_RANGE :
 RAISE_ERROR "color conversion error. take a new picture."
 return NULL

 return LAB_VALUES
```

```

Figure 4.5: Pseudocode: Image Processing Algorithms.

```
# 2: Raspberry Pi Code

```python
def EVENT_HANDLER():
 #extract lab values
 SAMPLE_PICTURE = CAMERA_CAPTURE initiated by BUTTON_PRESS
 LAB_VALUES = IMAGE_PROCESSING(SAMPLE_PICTURE)

 #calculate servo turns for desired recipe
 PAINT_QUANTITIES = assign paint quantities indicating how many
 | | | | | times the servo should turn with TARGET_SHADE

 DEPLOY_TO_RASPBERRY_PI(PAINT_QUANTITIES)
 #servo motors move syringes
```

```

Figure 4.6: Pseudocode: Embedded System.

156 «««< HEAD =====



Figure 4.7: UI/UX - Landing page.

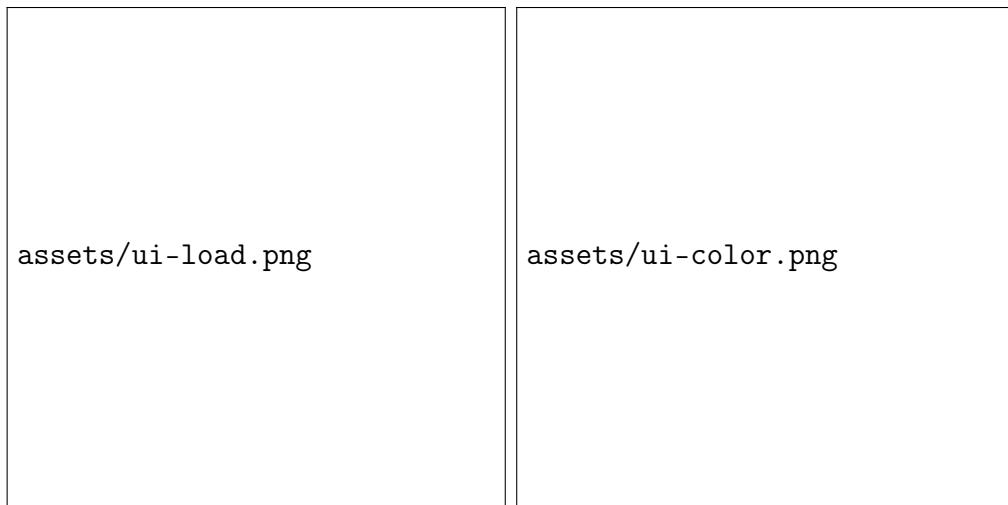


Figure 4.8: UI/UX - Loading the foundation shades.

¹⁵⁷ **4.2.3 User Flow Diagram**

¹⁵⁸ **4.2.4 Techstack**

¹⁵⁹ Operating systems: Linux on Raspberry Pi

¹⁶⁰ Libraries: OpenCV, NumPy, Pandas

¹⁶¹ Frameworks: React, Flask

¹⁶² Languages: Python

¹⁶³

¹⁶⁴ **4.3 Second Iteration**

¹⁶⁵ **4.3.1 Lo-fi/Mid-fi Designs**

¹⁶⁶ **4.3.2 Basic Logic and Functionality**

¹⁶⁷ ===== »»> refs/remotes/origin/main

```

# 1: Color Bias Adjustment and Sampling
```python
def IMAGE_PROCESSING(SAMPLE_PICTURE) :
 #INPUT: SAMPLE_PICTURE - image captured by camera
 #OUTPUT: LAB_values - color values compatible with paint mixing

 ### Get/Validate image containing skin region
 ### & reference colors/control sheet

 if SAMPLE_PICTURE is EMPTY/NOT_DETECTED :
 RAISE_ERROR "Image captured failed or canceled"
 return NULL

 # initial samples contain gamma-corrected RGB values
 # use OpenCV region of interest rectangle

 SKIN_SAMPLE = list of pixel RGB values that constitute a skin region from SAMPLE_PICTURE
 CONTROL_SAMPLE = list of pixel RGB values that encompass the reference color sheet SAMPLE_PICTURE

 # check whether the reference sheet is present or the image is too dark/light

 if CONTROL_SAMPLE is EMPTY/NOT_DETECTED :
 RAISE_ERROR "Control sheet not detected. Take picture with color reference sheet in a well-lit room."
 return NULL

 # get average gamma-corrected RGB codes for the skin region and control

 SKIN_RGB_AVG = calculate average of SKIN_SAMPLE
 CONTROL_MEASURED_VALUES = list of averages of CONTROL_SAMPLE

 # get linear RGB codes from gamma-corrected RGB codes
 # allows linear operations to be performed on the codes

 SKIN_LINEAR_RGB = apply reverse gamma-correction to SKIN_RGB_AVG
 CONTROL_LINEAR_RGB = list of linearized RGB codes from CONTROL_MEASURED_VALUES

 # find difference in the control input RGB codes and stored RGB codes

 CONTROL_KNOWN_VALUES = load("CONTROL_DATABASE.csv") and linearize the codes
 CONTROL_TRANSFORM = find transformation matrix that makes CONTROL_LINEAR_RGB = CONTROL_KNOWN_VALUES * CONTROL_TRANSFORM

 # apply difference to the values found in the skin region for lighting correction

 XYZ = apply CONTROL_TRANSFORM to SKIN_LINEAR_RGB

 # convert rgb value to lab for later processing

 LAB_VALUES = convert XYZ color space to LAB(XYZ)

 for element in LAB_VALUES :
 if element is OUT_OF_RANGE :
 RAISE_ERROR "color conversion error. take a new picture."
 return NULL

 return LAB_VALUES
```

```

Figure 4.9: Pseudocode: Image Processing Algorithms.

```
# 2: Raspberry Pi Code

```python
def EVENT_HANDLER():
 #extract lab values
 SAMPLE_PICTURE = CAMERA_CAPTURE initiated by BUTTON_PRESS
 LAB_VALUES = IMAGE_PROCESSING(SAMPLE_PICTURE)

 #calculate servo turns for desired recipe
 PAINT_QUANTITIES = assign paint quantities indicating how many
 | | | | | times the servo should turn with TARGET_SHADE

 DEPLOY_TO_RASPBERRY_PI(PAINT_QUANTITIES)
 #servo motors move syringes
```

```

Figure 4.10: Pseudocode: Embedded System.

¹⁶⁸ 4.3.3 User Flow Diagram

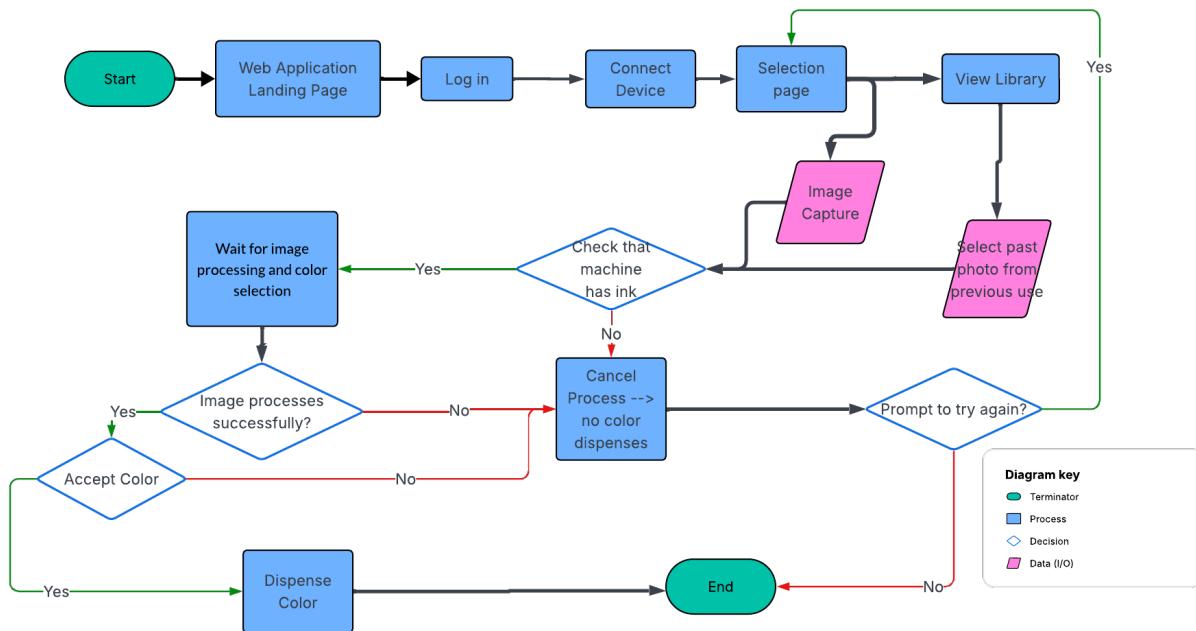


Figure 4.11: User Flow Diagram

169 4.3.4 Techstack

170 Frontend: React, Vite, Typescript, TailwindCSS, Lucide Icons
 171 Storage : IndexedDB (local storage)
 172 Hardware/OS/Hosting : Raspberry Pi (Linux, Pi Local Hosting)
 173 Computer Vision & Data Science : OpenCV, NumPy, Pandas, Haar Cascades
 174 Languages : Typescript, Python
 175 Tooling : Git/Github

176 4.4 Second Iteration

177 4.4.1 Lo-fi/Mid-fi Designs

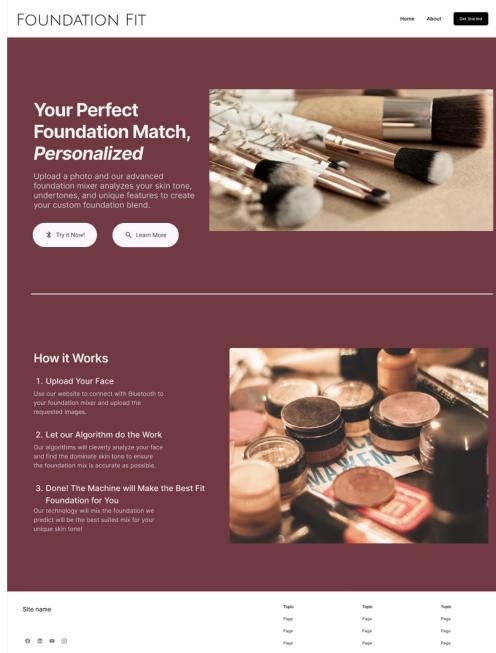


Figure 4.12: Initial Figma Mockup: UI/UX - Landing page.

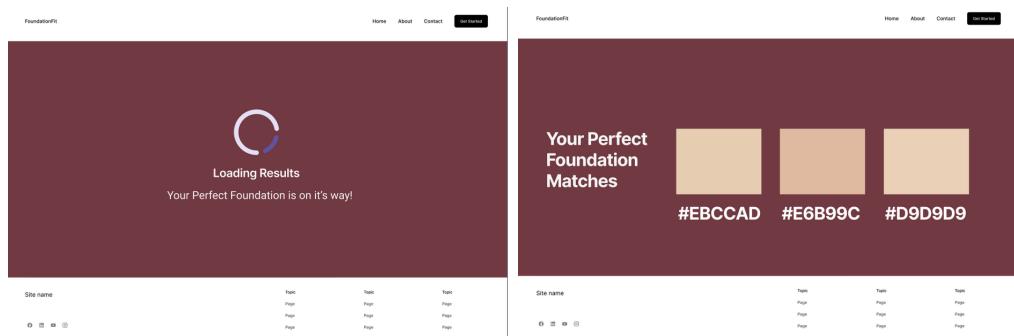


Figure 4.13: Initial Figma Mockup: UI/UX - Loading the foundation shades.

178 4.4.2 Basic Logic and Functionality

179 The backend service is responsible for two core tasks: analyzing a user's skin tone from
180 an input image and triggering the dispenser system for a requested foundation color. It is
181 implemented as a Flask web API with three main endpoints: '/analyze', '/dispense', and
182 '/ping' *app.py*.

183 Skin Tone Analysis Pipeline '/analyze'

184 Image intake (frontend → backend) :

185 Frontend allows the user to view a live feed and takes a picture when it detects the face
186 reference sheet within the yellow box on the screen. The photo is sent to the '/analyze'
187 endpoint. Backend strips, decodes, and loads the data captured into an image object, which
188 is saved for processing.

189 Skin region extraction :

190 The image is converted to a NumPy RGB array and passed to 'define_skin()', which
191 detects and isolates skin pixels. This focuses the analysis on relevant skin regions instead of
192 the background.

193 Color space processing and lighting correction :

194 Skin samples are flattened into a matrix and normalized. 'gamma_to_linear()' converts
195 gamma corrected RGB values to linear RGB. 'lighting_correction()' corrects the for
196 inconsistent lighting and the corrected RGB is converted to XYZ in 'linear_to_xyz()'
197 then to LAB in 'xyz_to_lab()'.

198 Output color encoding :

199 The average LAB color is mapped to an approximate RGB value in 'lab_avg_to_rgb()'
200 and formatted as a hexadecimal color string. This hexadecimal string is returned as '{"re-
201 sult": "<hex_color>"}' to the front end, which uses it for visualization for the user.

202 Dispensing Logic '/dispense'

203 Dispense accepts JSON payload containing the LAB code of the detected color. The
204 endpoint validates that the color is detected and prints a message to let the user know
205 dispensing has started. The endpoint is connected to the Raspberry Pi that drives the
206 stepper motors and pumps.

207 HealthCheck '/ping'

208 The '/ping' endpoint is a lightweight health-check route that returns status updates. It
209 allows the frontend to verify that the backend service is alive and responsive without invoking
210 the full analysis pipeline.

211 <<<< HEAD ===== >>>> Stashed changes >>>> 9ae2eabe5e6d430ba55097c3c2d1ff9519d985e5
212 >>>> refs/remotes/origin/main

213 4.4.3 Core Features of Algorithm

214 The color detection algorithm transforms raw camera input into a corrected, device and
 215 lighting independent representation of the user's skin tone. This corrected color is then
 216 mapped to a cosmetic shade.

217 Macbeth Color Chart Reference Calibration

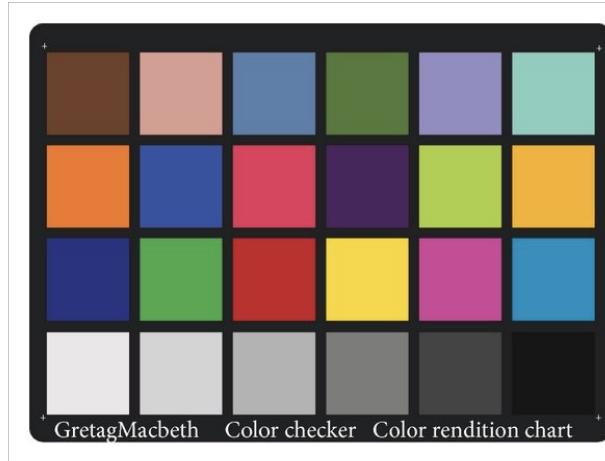


Figure 4.14: Macbeth Color Reference Sheet

218 The algorithm incorporates a Macbeth-style color chart for lighting and camera calibra-
 219 tion. Known RGB or Lab reference values for each chart patch are compared to captured
 220 values:

- 221 • The color chart is detected and segmented from the image.
- 222 • We extract a sample from the middle of the patch.
- 223 • We correct the lighting and distortion

224 Robust Skin Sampling from Facial Regions

225 A Haar-cascade face detector identifies the face region and we extract stable sampling
 226 zones from the mask returned by the Haar-cascade. Regions of interest were defined as the
 227 forehead, nose, and cheeks. Multiple points inside each region are collected to avoid shadows,
 228 edges, or hair interference. These values are aggregated to produce a stable estimate of the
 229 user's skin tone.

230 Gamma Correction and Linear RGB Conversion

231 Since cameras produce gamma-encoded RGB values, the algorithm applies a gamma-to-
 232 linear transformation. Both chart colors and skin samples are linearized before calibration.
 233 Working in linear RGB ensures that subsequent lighting corrections and color transfor-
 234 mations are physically meaningful.

235 Lighting Correction Using Chart Reference Data

236 A lighting correction is computed by comparing captured and reference chart patch col-
237 ors. This correction is applied uniformly to the sampled skin pixels, producing values that
238 approximate standardized, ideal lighting conditions.

239 Transformation into Lab Color Space

240 The corrected linear RGB values are converted into XYZ and then into Lab, a device-
241 independent and perceptually uniform color space. Multiple Lab samples from different
242 regions are averaged to form the final skin-tone vector.

243 Cosmetic Shade Selection and Hex Output

244 The final Lab vector is matched against a database of known cosmetic shades, each
245 precomputed in Lab. The system selects the shade with the smallest perceptual difference
246 (ΔE). The algorithm returns the shade as: a hex code, corrected RGB, and Lab code.

247 Integration with Hardware Dispensing Logic

248 The selected shade is mapped to pigment ratios for the dispenser hardware. The backend
249 provides:

- 250 • `/analyze`: accepts an image and returns the computed shade,
- 251 • `/dispense`: controls Raspberry Pi motors to dispense pigment.
- 252 • `/ping`: returns status updates and process health

253 4.5 Third Iteration**254 4.5.1 Final Product**

255 [Foundation Fix Github](#)

256 Chapter 5

257 Electrical Systems Design

258 5.1 First Iteration

259 5.1.1 Initial Wiring Diagrams

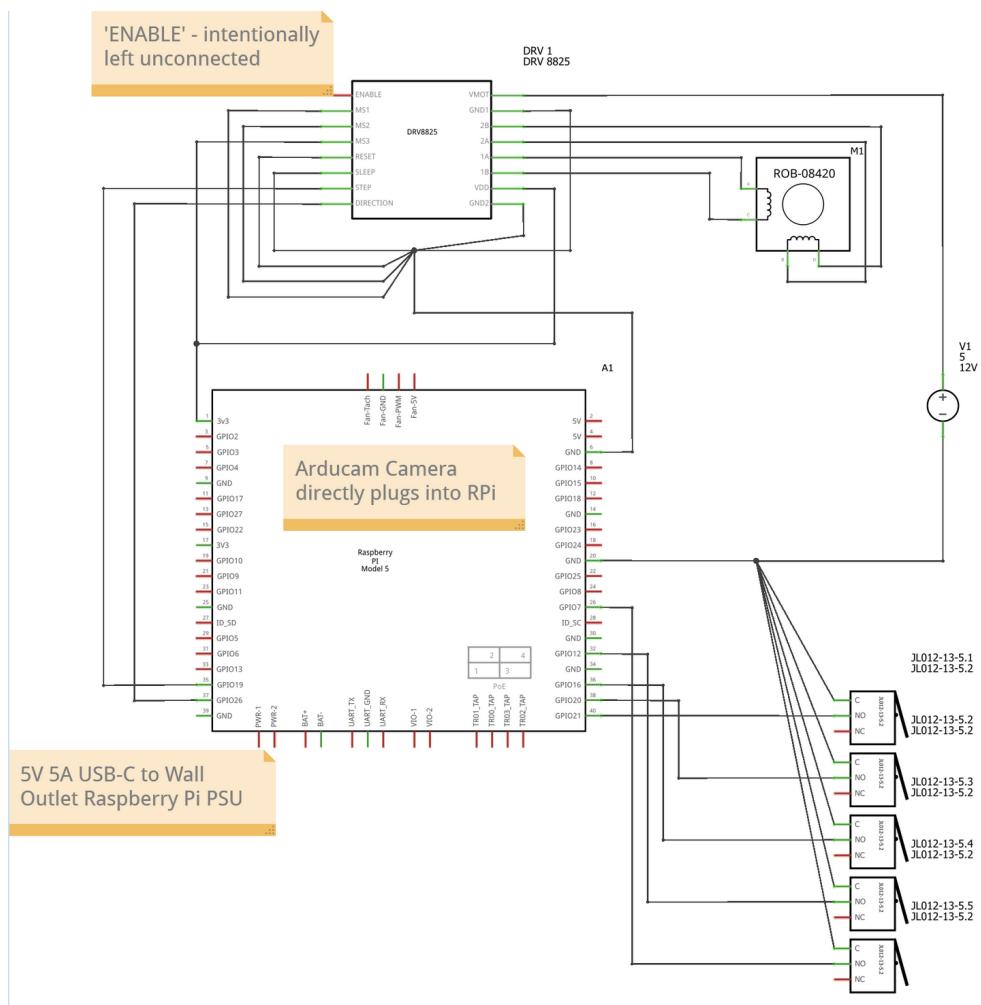


Figure 5.1: Preliminary Wiring Diagram

260 **5.1.2 Initial Power Calculations**

| Parameter | Symbol | Value |
|------------------|--------|-------|
| Phase Current | I | 0.7 |
| Phase Resistance | R | 4.0 |
| Phases | - | 2 |

Table 5.1: Power Calculations

261 $P_{motor} = 2 * I^2 * R$

262 $P_{motor} = 2 * (0.7)^2 * 4.0 = 3.92 * 5 = 19.6W$

263 $P_{RaspberryPi} = 10W$

$P_{total} = 10 + 19.6 = 30W$

264 **5.2 Second Iteration**

265 **5.2.1 Circuit Building**

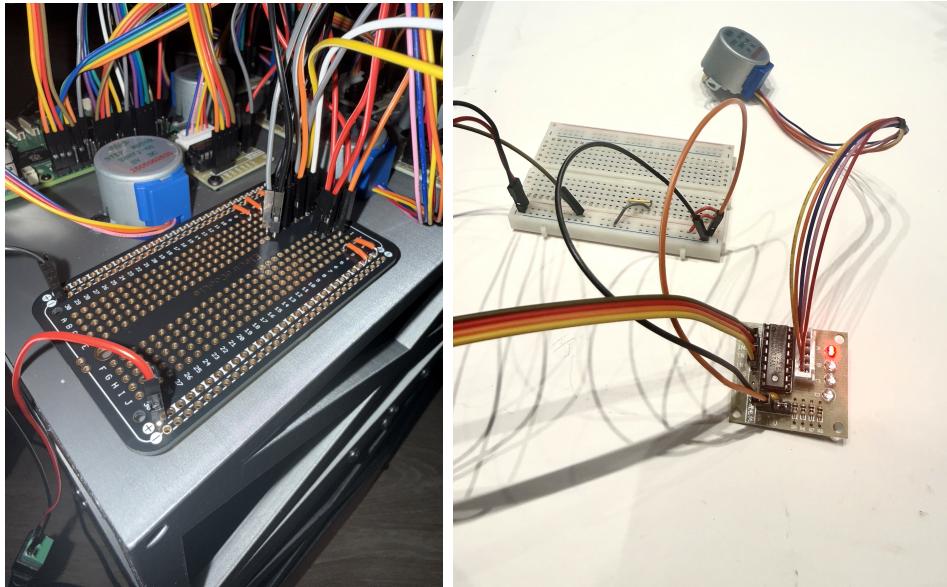


Figure 5.2: Perf Board and Servo Motor to Driver Connection

266 During the second iteration, several key changes were made to the electrical system. The
267 original 12 V NEMA 17 stepper motors and their corresponding drivers were replaced with
268 5V 28BYJ-48 stepper motors paired with ULN2003 driver boards. This allowed the removal
269 of the 12 V power supply and enabled the entire system to operate from a single 5V source.
270 The wiring diagram was updated accordingly to reflect these component substitutions.

271 5.3 Third Iteration

272 5.3.1 Final Wiring Diagrams

273 «««< HEAD ===== «««< HEAD ===== «««< Updated upstream =====
 274 »»»> 9ae2eabe5e6d430ba55097c3c2d1ff9519d985e5 »»»> refs/remotes/origin/main

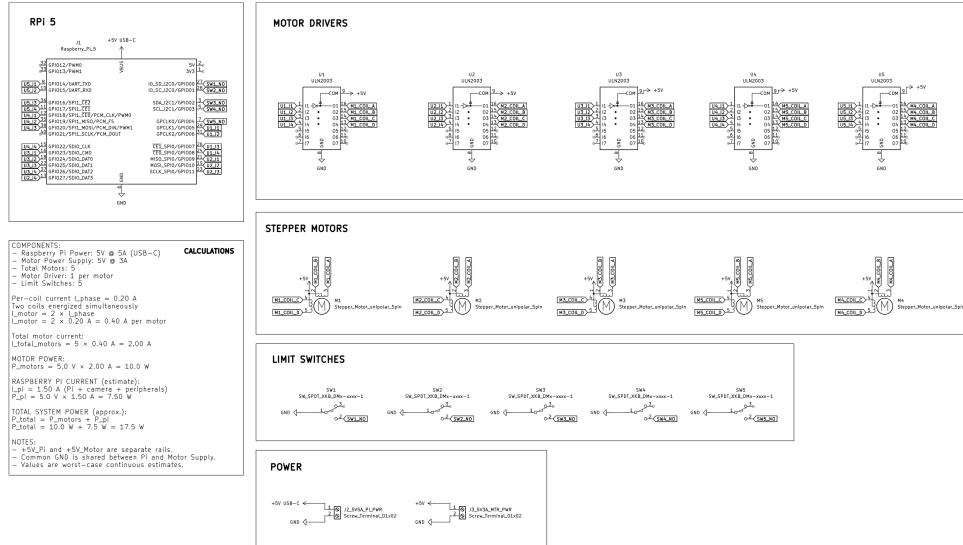


Figure 5.3: Final Wiring Diagrams

275 «««< HEAD =====

276 «««< HEAD ===== »»»> Stashed changes »»»> 9ae2eabe5e6d430ba55097c3c2d1ff9519d985e5
 277 »»»> refs/remotes/origin/main

278 5.3.2 Final Power Calculations

279 *Per Motor:*

280 Approximately 100 mA per coil 2 coils 5 V $\bar{1}$ W

281 *Raspberry Pi 5:*

282 Average consumption of 1.2A 5 V $\bar{6}$ W

283 Total System Power:

$$P_{motor} * 5 + P_{pi} = 1W * 5 = 11W$$

²⁸⁴ 5.3.3 Circuitry

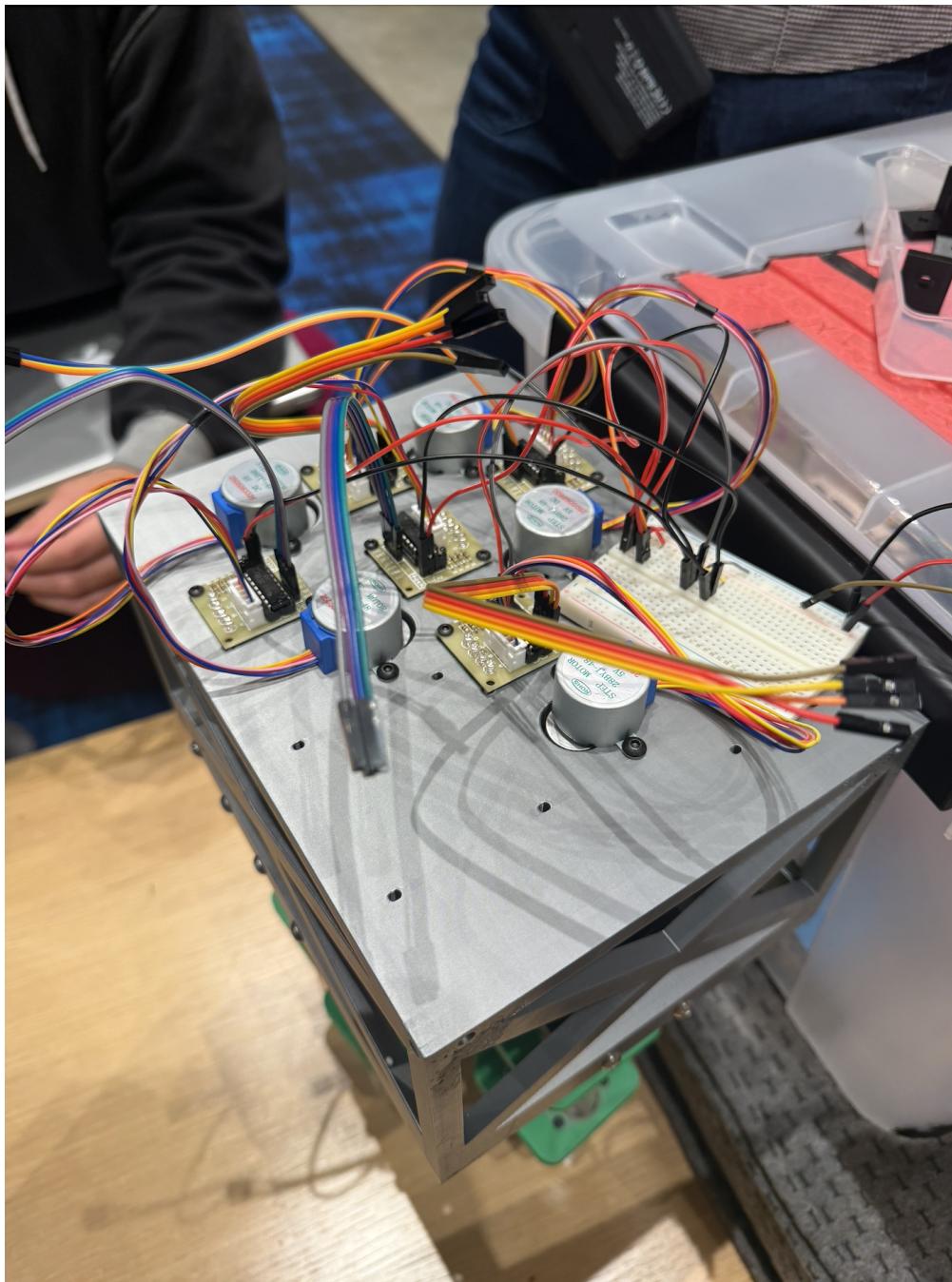


Figure 5.4: Final Circuitry Under Housing Lid

²⁸⁵ The only major change from the prototype stage was the transition from breadboard
²⁸⁶ wiring to permanent soldered connections on a perfboard. This improved durability, reduced
²⁸⁷ wiring instability, and prepared the circuitry for integration into the final enclosure.

²⁸⁸ Chapter 6

²⁸⁹ Final Product



Figure 6.1: Dispenser Systems Fully Assembled

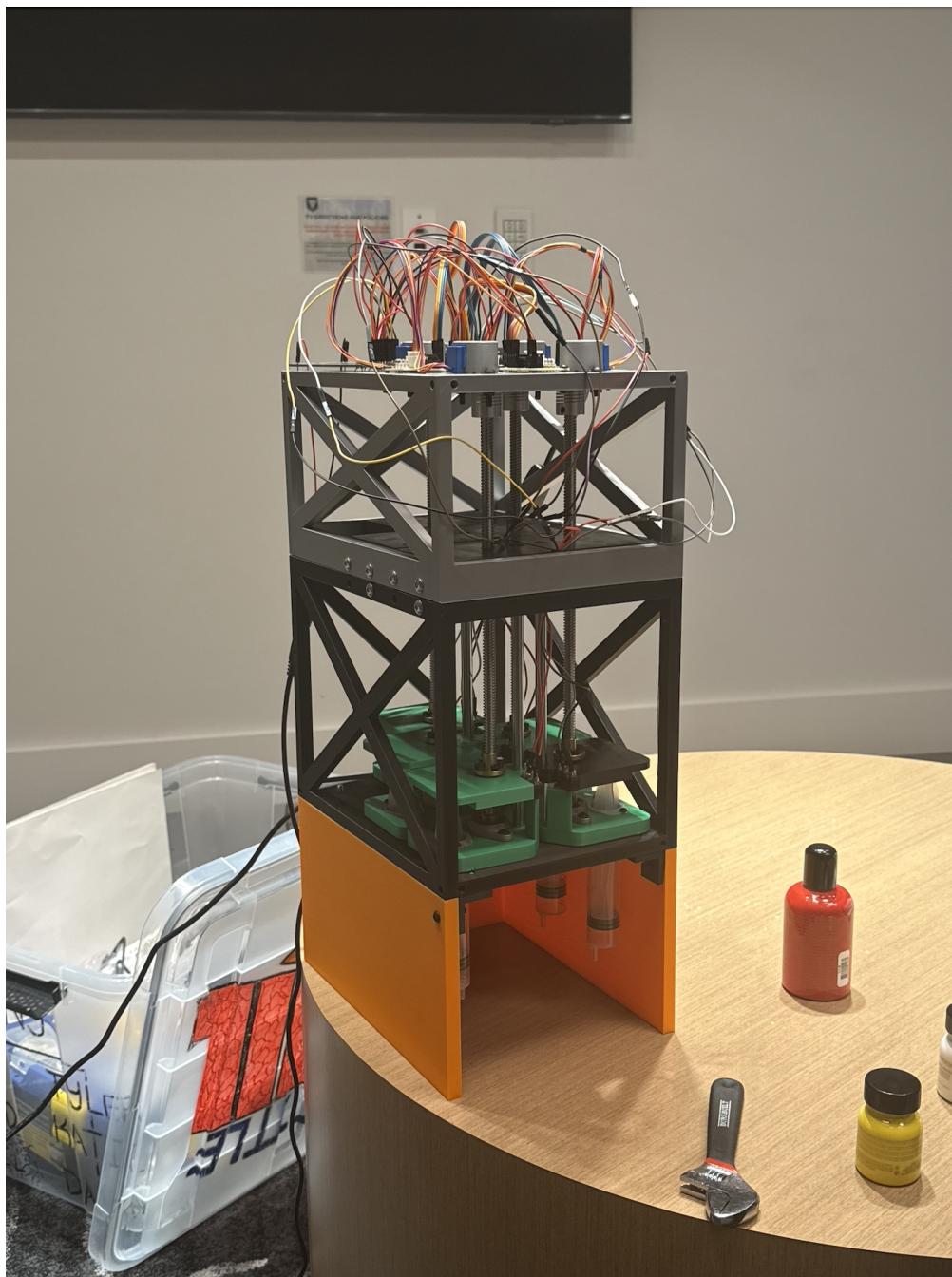


Figure 6.2: Skeleton/Internal Structure

²⁹⁰ **6.1 Testing Protocol**

²⁹¹ **6.2 Integration**

²⁹² **6.3 System Performance**

²⁹³ Chapter 7

²⁹⁴ Discussion

²⁹⁵ 7.1 Limitations

²⁹⁶ 7.2 Future Work

²⁹⁷ **Chapter 8**

²⁹⁸ **Conclusion**

²⁹⁹ **8.1 Bill of Materials**

| Component | Purpose | Qty | Price |
|---|----------------------------|-----|-----------------|
| 2PCS 300mm Tr8x8 Lead Screw + Brass Nut | Linear motion | 3 | \$13.99 |
| 5mm–8mm Lead Screw Coupler (5 pack) | Motor–screw coupling | 1 | \$9.99 |
| Linear Motion Rod Guide 8mm × 200mm | Linear guidance | 3 | \$8.69 |
| 8mm Flange Pillow Block Bearing | Shaft support | 2 | \$8.99 |
| Raspberry Pi 5 (4GB RAM) | Main controller | 1 | \$66.00 |
| 5 Pack Plastic 30mL Syringes | Fluid handling prototype | 1 | \$6.99 |
| Raspberry Pi 5 Power Supply | Power for controller | 1 | \$15.99 |
| Assorted Metric Fasteners (M2–M5) | Mechanical assembly | 1 | \$20.00 |
| Limit Switch (10 pack) | Motion limit detection | 1 | \$5.99 |
| Wiring Kit | Electrical connections | 1 | \$6.98 |
| 5V 3A DC Power Supply | Low-voltage power | 1 | \$7.47 |
| 5 Sets 28BYJ-48 + ULN2003 Driver | Secondary actuation system | 1 | \$14.99 |
| M2 Spacers | Mechanical spacing | 1 | \$6.69 |
| Perfboard / Breadboard | Circuit prototyping | 1 | \$8.79 |
| Mehron Liquid Makeup Colors | Testing material | 1 | \$50.00 |
| Large 3D Prints (>4 hrs) | Structural components | 1 | \$144.00 |
| Total Estimated Cost | | | \$441.12 |

Table 8.1: Updated Bill of Materials.

³⁰⁰ References