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

Contents

List of Figures

List of Tables

Chapter 1

Abstract

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

Chapter 2

Introduction

2.1 Research

2.1.1 Background of Problem

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

The beauty industry produces 120 billion packaging units per year and 95% of this these units are discarded, as opposed to recycled [1]. 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.

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.

2.1.2 Existing Solutions

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 percieved the same depending on the lighting of the room, as well as the high cost of their machine.

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

2.2 Design Choices

2.2.1 Past Considerations and Scrapped Plans

Chapter 3

Hardware Design and Specifications

3.1 First Iteration

3.1.1 Initial CAD Model and Hand Sketches

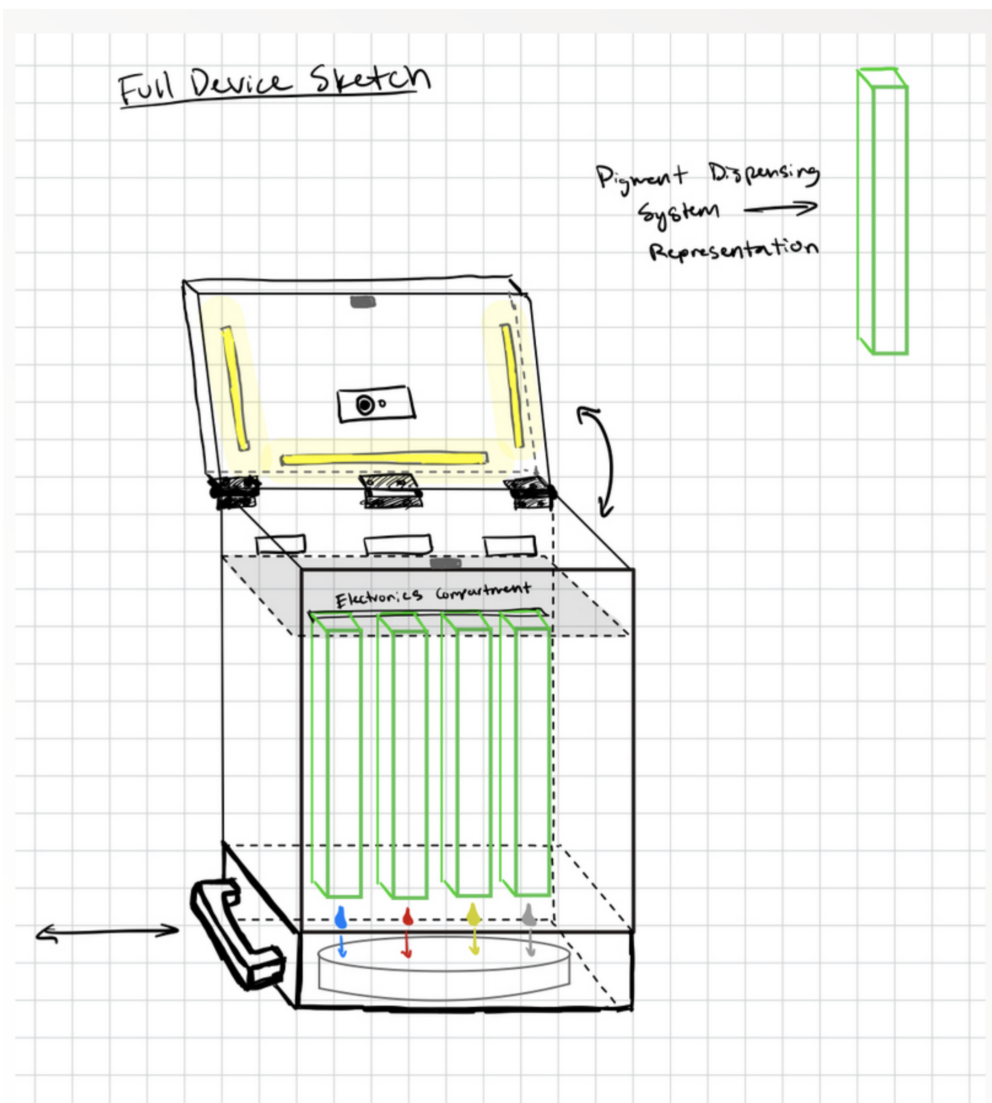

Figure 3.1: First Sketch: Housing

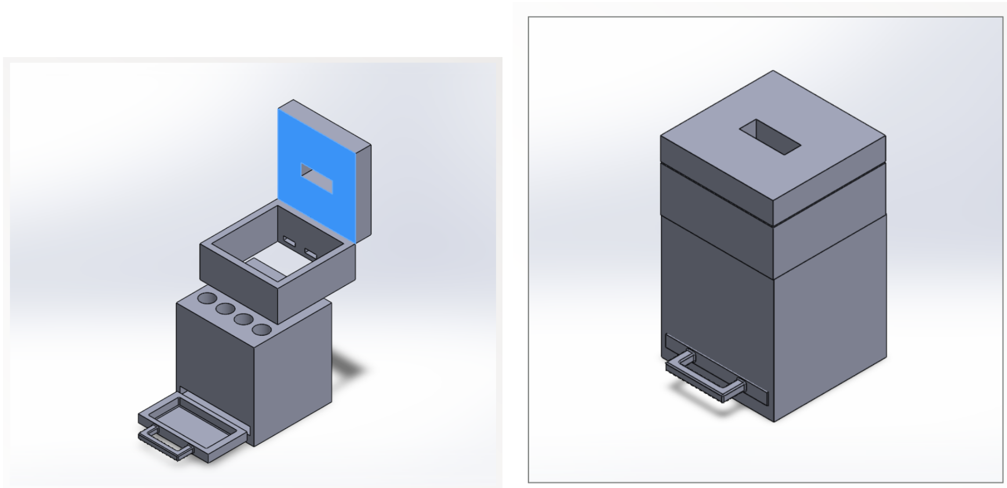

Figure 3.2: First CAD Models: Housing

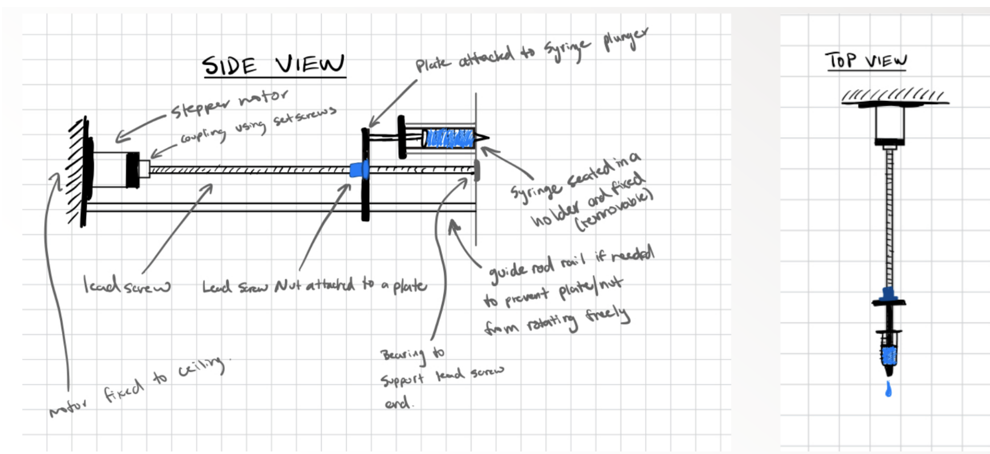

Figure 3.3: First Sketch: Dispenser Systems

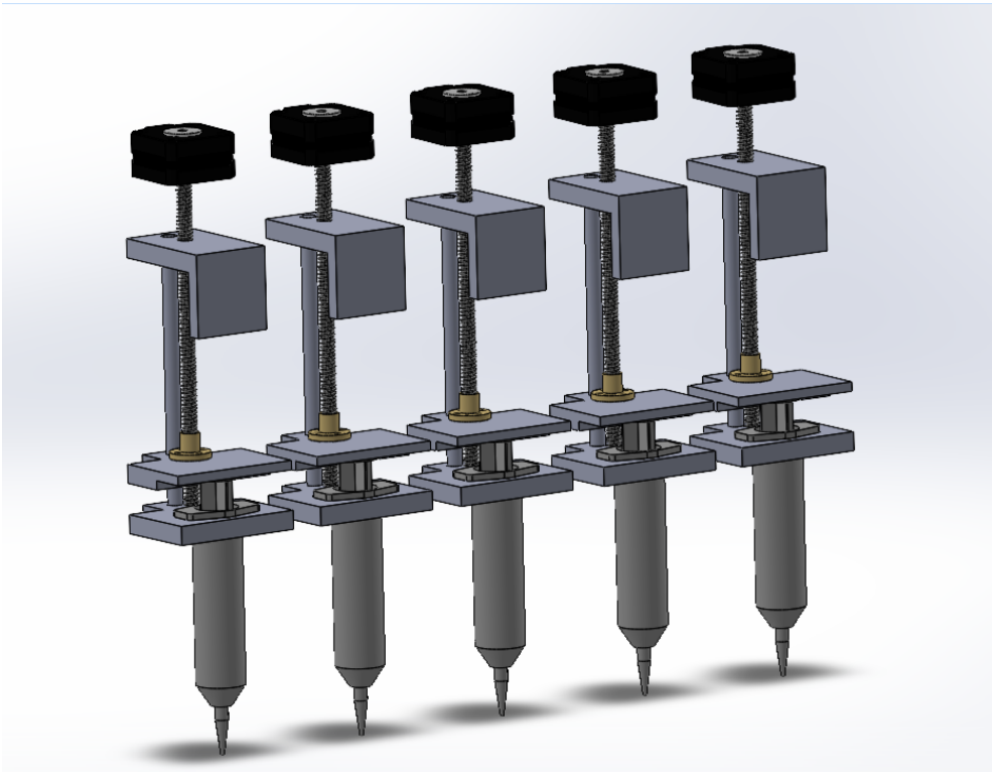

Figure 3.4: First CAD Models: Dispenser Systems

3.1.2 Commentary

3.2 Second Iteration

3.2.1 Updated CAD Model

3.2.2 Commentary

3.3 Third Iteration

3.3.1 Final CAD Model

3.3.2 Final Manufacturing Drawings for Custom Parts

3.3.3 Commentary

Chapter 4

Software Design

4.1 First Iteration

4.1.1 Preliminary Diagrams

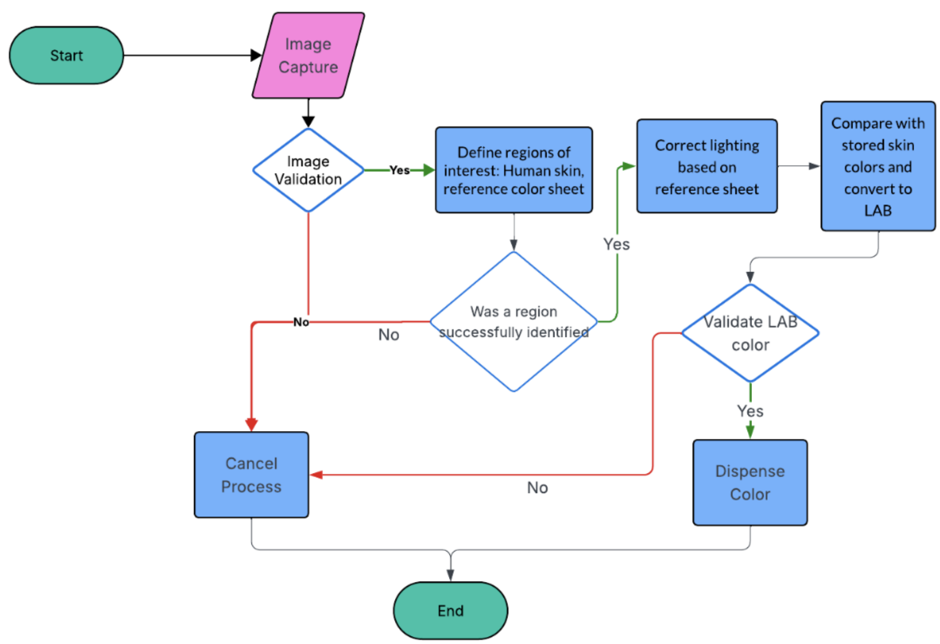

Figure 4.1: Initial Flowchart for Control Logic

````
# 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 "Contol 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_TRANFORM 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 4.2: Pseudocode: Image Processing Algorithms.
````

```
# 2: Raspberry Pi Code

```python
def EVENT_HANDLER():
    #extract lab values
    SAMPLE_PICTURE = CAMERA_CAPUTRE 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 4.3: Pseudocode: Embedded System.

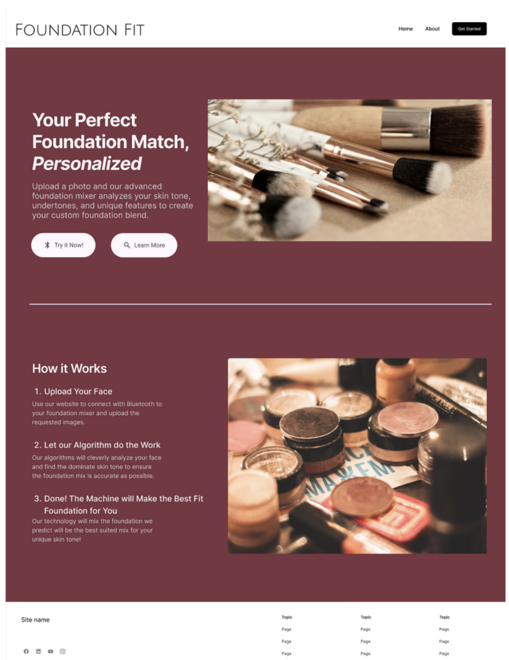

Figure 4.4: UI/UX - Landing page.
```

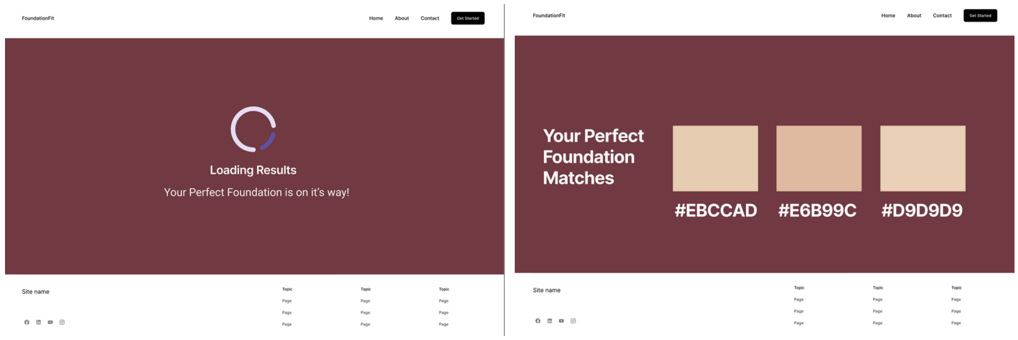

Figure 4.5: UI/UX - Loading the foundation shades.

4.1.2 User Flow Diagram

4.1.3 Techstack

Operating systems: Linux on Raspberry Pi
Libraries: OpenCV, NumPy, Pandas
Frameworks: React, Flask
Languages: Python

4.2 Second Iteration

4.2.1 Lo-fi/Mid-fi Designs

4.2.2 Basic Logic and Functionality

4.2.3 Core Features of Algorithm

4.3 Third Iteration

4.3.1 Final Product

Chapter 5

Electrical Systems Design

5.1 First Iteration

5.1.1 Initial Wiring Diagrams

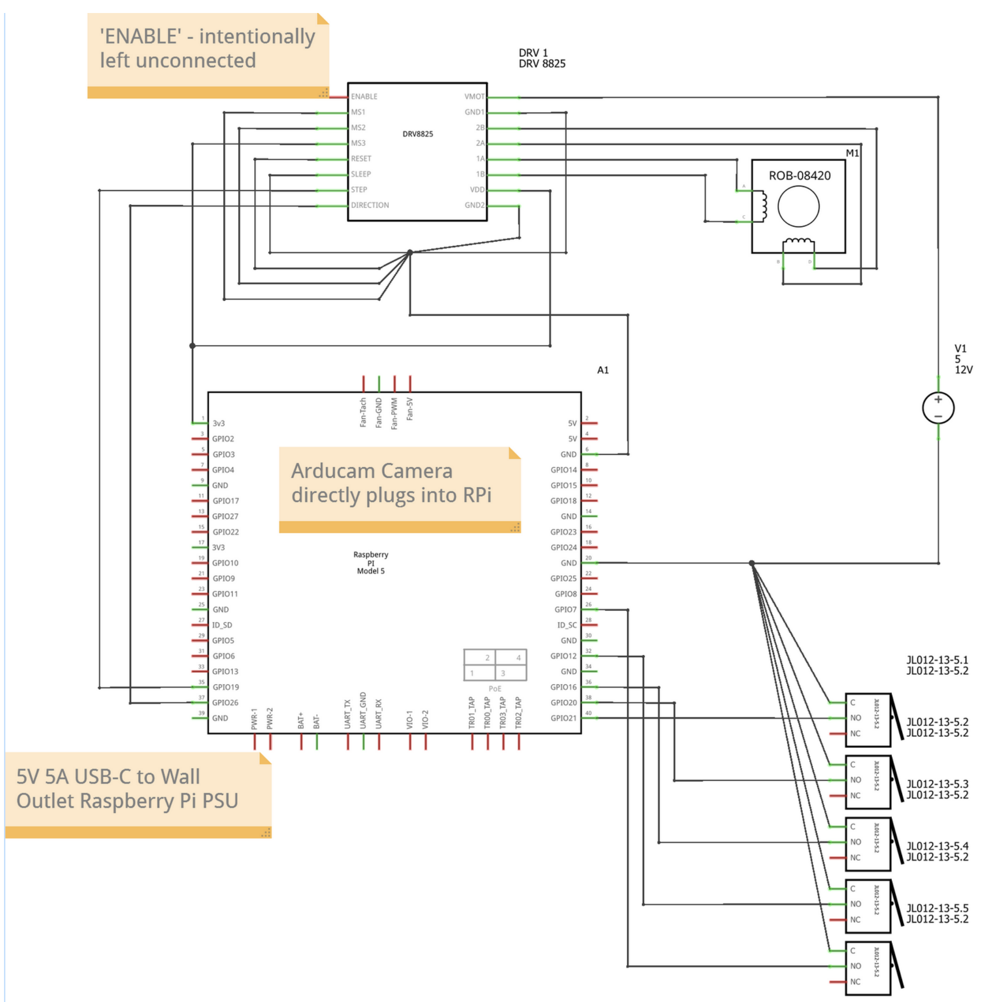

Figure 5.1: Preliminary Wiring Diagram

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

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

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

$$P_{RaspberryPi} = 10W$$

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

5.2 Second Iteration

5.2.1 Circuit Building

5.3 Third Iteration

5.3.1 Final Wiring Diagrams

5.3.2 Final Power Calculations

5.3.3 Circuitry

Chapter 6

Final Product

6.1 Testing Protocol

6.2 Integration

6.3 System Performance

Chapter 7

Discussion

7.1 Limitations

7.2 Future Work

Chapter 8

Conclusion

8.1 Bill of Materials

Component	Purpose	Quantity	Price
2PC T8x8 Lead Screw with Brass Nut	Linear motion conversion	3	$13.99
5-8mm Lead Screw Coupler 5 pack	Connect motor to screw	3	$8.69
8x200mm Shaft Guide	Linear guidance	3	$8.69
8mm Flange Mounted Pillow Block Bearing	Support rotating shaft	3	$8.99
Raspberry Pi 5 (4GB RAM)	Primary controller	1	$66.00
Plastic 30mL Syringes (Pack of 5)	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
Arducam V3 Camera	Vision and monitoring	1	$25.00
Wiring Kit	Electrical connections	1	$5.00
Breadboard / Perfboard	Circuit prototyping	1	$10.00
5V 3A DC Power Supply	General power source	1	$7.47
5 Sets 28BYJ-48 Stepper + ULN2003 Driver	Secondary actuation system	5	$14.99
1.75mm ABS Filament Reel (Black)	Structural printing material	1	$16.99
Total Estimated Cost			**$346.49**

Table 8.1: Key Hardware Elements.

References

Bibliography

[1] Sophie Smith. Billions of beauty packaging goes unrecycled every year - theindustry.beauty. 2024.