ME102B Final Project Report - Sensor-Based Sorting Lid

Zhihan Wang,Kaleb Zhong,Tom Springer

1. Opportunity

Waste sorting has long been a challenge for individuals and businesses aiming to improve recycling efficiency. A new opportunity exists to automate this process with a device that can easily be added to any standard trash can. This device would use sensors to identify the material type, and automatically sort waste into the appropriate compartment.

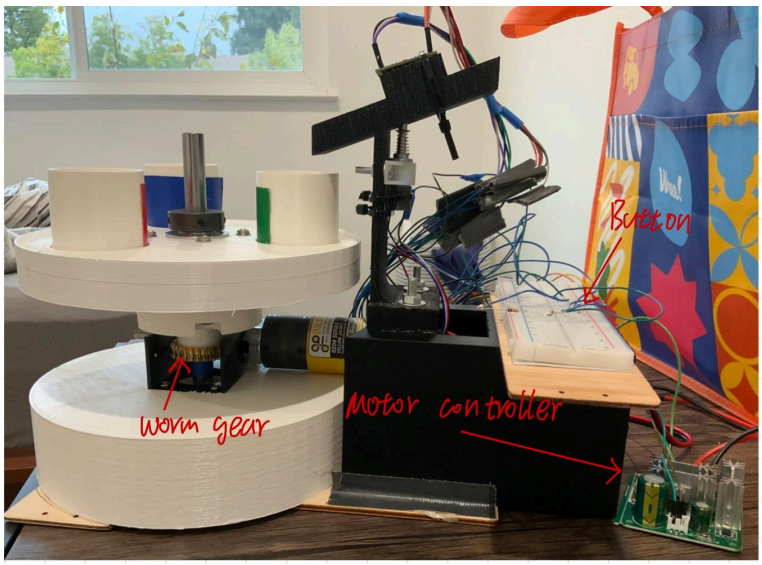
2. High-Level Strategy

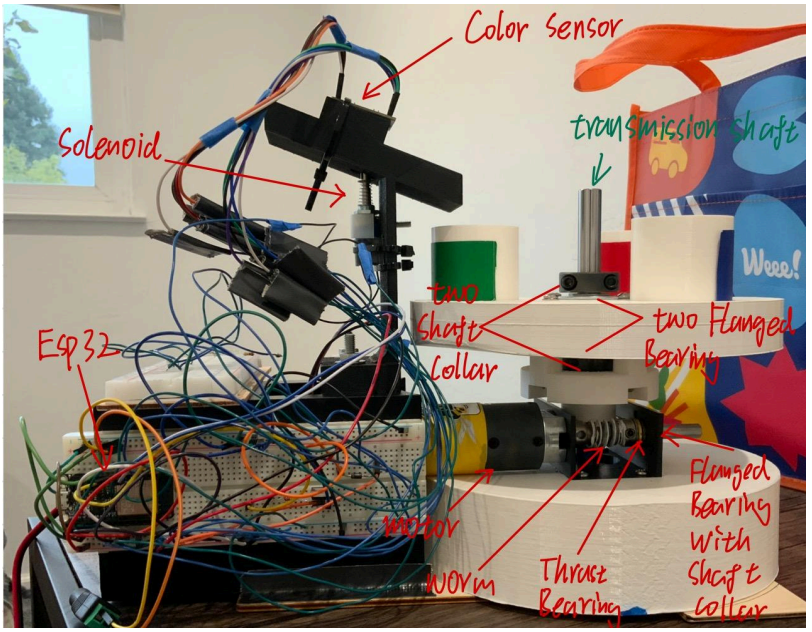
After an initial discussion with the professor, we realized the difficulty of accurately identifying waste. Therefore, we ultimately adopted the professor's suggestion: using chips of different colors to represent different types of waste and shifting our goal to accurately classifying chips of different colors instead of different types of waste.Our initial envisioned functionality was clear: to ensure sufficient accuracy even during long-term operation. Our design is based on a rotatable turntable driven by a motor. Ultimately, we successfully achieved this goal. Our classification system is divided into two subsystems. 1.The detection and release system, consisting of a color sensor and a solenoid. 2.The rotation system, comprising a motor, turntable, worm gear, drive shaft, bearings, and other components.

The basic operational logic of this design is as follows: when colored chips (red, green, or blue) are introduced, the color sensor detects their color and transmits the RGB data to the ESP32. At this stage, the Solenoid remains closed. The three numerical values from the RGB data are compared; for instance, if the red value is smaller than both the green and blue values, we determine it to be red. The same principle is applied for the other colors.We then assign different rotation angles to each color to correspond to the positions of their respective bins. Red rotates counterclockwise by 120 degrees, blue by 240 degrees, and green remains stationary at 0 degrees. After rotating to the designated color position, the Solenoid opens after a one-second delay. It stays open for a set period to ensure the chips have enough time to slide into the bins before closing.Once this process is complete, the turntable continues rotating to return to its initial position (the green bin location). For example, with red, the turntable first rotates 120 degrees to the red bin, then rotates 240 degrees to return to the initial position.If a color other than red, green, or blue is detected, the system outputs "Color not detected."

We use an encoder to monitor the current rotation angle of the turntable. By calculating the encoder counts required for one full rotation based on the gear ratio of the motor-worm gear assembly, we determine the turntable's speed in counts per degree. The encoder counts are checked every 0.5 ms, and the rotation angle is calculated by dividing the current counts by the turntable speed.To ensure the turntable accurately reaches the designated positions, we added parameters and set error margins for adjustments. Ultimately, we achieved our initial goal: the system can automatically sort chips of different colors into their respective bins with extremely high accuracy, even after multiple runs.We also implemented a non-volatile storage (NVS) feature for state saving and recovery. This ensures that even if the ESP32 restarts or loses power during a task, it can resume and complete the previously unfinished task seamlessly.We also implemented turning the color sensor on or off with a 1-second button press.

3. Integrated Device Overview





4.Function-critical Decisions and Calculations

Based on reasonable estimations and accurate modeling data, we have derived the following conditions: M_disk = 1.2 kg , R_disk = 0.1 m , M_gear = 0.2 kg , R_gear = 0.025 m , Reasonably assumed as the turntable's stable speed ω = 1 rad/s , Δt = 0.001 s , Reduction ratio (worm gear): 28:1

$$I_{\text{disk}} = \frac{1}{2} M_{\text{disk}} R_{\text{disk}}^2 = \frac{1}{2} \cdot 1.2 \cdot (0.1)^2 = 0.5 \cdot 1.2 \cdot 0.01 = 0.006\,\text{kg} \cdot \text{m}^2$$

$$I_{\text{gear}} = \frac{1}{2} M_{\text{gear}} R_{\text{gear}}^2 = \frac{1}{2} \cdot 0.2 \cdot (0.025)^2 = 0.5 \cdot 0.2 \cdot 0.000625 = 0.0000625\,\text{kg} \cdot \text{m}^2$$

$$I_{\text{total}} = I_{\text{disk}} + I_{\text{gear}} = 0.006 + 0.0000625 = 0.0060625\,\text{kg} \cdot \text{m}^2$$

Calculate the angular acceleration of the turntable: $\alpha = \frac{\Delta \omega}{\Delta t} = \frac{1\,\text{rad/s}}{0.001\,\text{s}} = 1000\,\text{rad/s}^2$

Calculate the torque required by the motor when the turntable's angular velocity is 1 rad/s.

$$\tau = I \cdot \alpha$$

$$\tau_{\text{needed}} = I_{\text{total}} \cdot \alpha = 0.0060625 \cdot 1000 = 6.0625\,\text{N} \cdot \text{m}$$

$$\tau_{\text{motor}} = \frac{\tau_{\text{needed}}}{28} = \frac{6.0625}{28} \approx 0.2165\,\text{N} \cdot \text{m}$$

Calculate the actual torque output of the motor at a turntable speed of 1 rad/s

$$\text{No-Load Speed} = 435\,\text{RPM} \times \frac{2\pi}{60} \approx 435 \times 0.10472 \approx 45.6\,\text{rad/s}$$

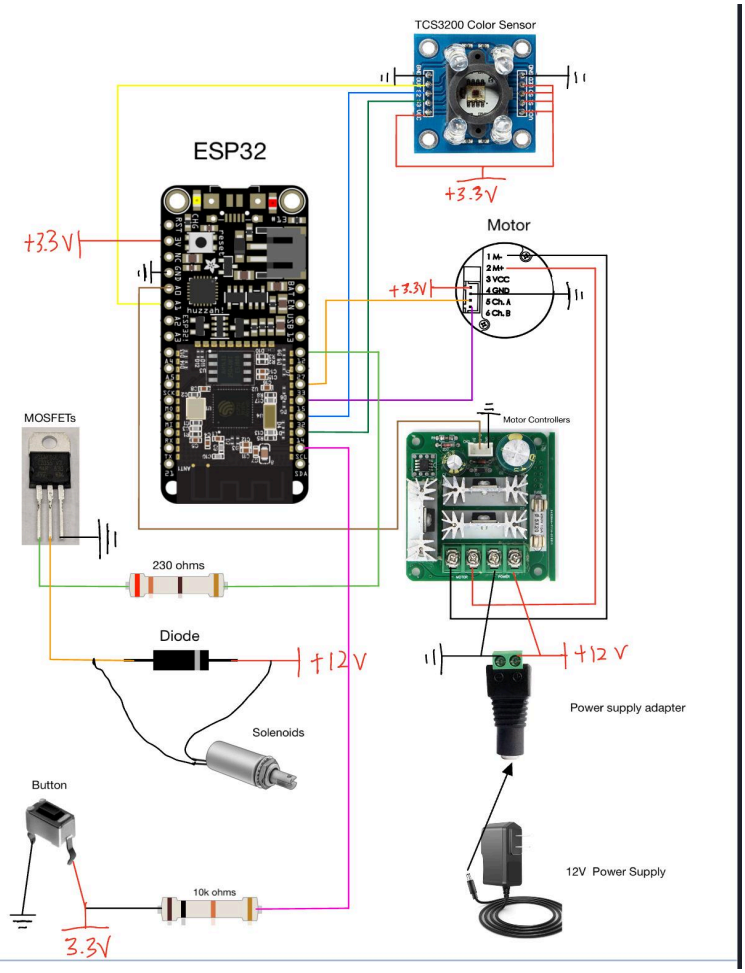
$$\omega_{\text{gearbox output}} = \omega_{\text{turntable}} \times 28 = 1 \times 28 = 28\,\text{rad/s}$$

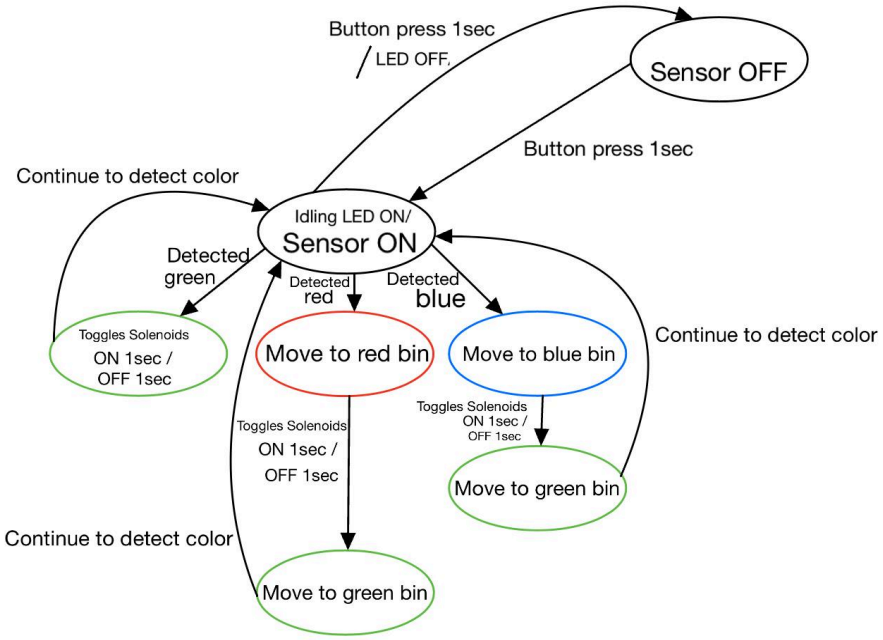
$$T(\omega) = T_{\text{stall}} \left(1 - \frac{\omega}{\omega_{\text{no-load}}}\right)$$

$$T = 1.835 \cdot \left(1 - \frac{28}{45.6}\right) = 1.835 \cdot (1 - 0.614) = 1.835 \cdot 0.386 \approx 0.708\,\text{N} \cdot \text{m}$$

Under the specified conditions, the required torque of 0.2165 Nm is less than the motor's actual output torque of 0.708 Nm. This means that even with the additional torque caused by various frictions, the motor can still meet the torque requirements. Another reason for choosing this motor is its affordability. Although we found motors with torque closer to our required value, they were more expensive. Thus, we ultimately selected this motor. While our design does not have strict speed requirements, our focus is more on precision. Given this scenario, we have ample room to adjust the motor to achieve the speed we deem appropriate.

5. Circuit Diagram and State Transition Diagram





6. Reflection for Future Students

Strategies that worked well for our team included completing the prototype design and manufacturing early, as well as conducting frequent subsystem testing. Precise task allocation and extended collaborative work sessions also contributed to our success. Group brainstorming allowed us to solve problems more efficiently.Our advice for future teams is to allocate more time for integration and debugging. Even if each subsystem works fine individually, numerous issues can still arise during integration. Allowing more time for debugging also means that if design flaws are discovered during the process, there will be enough time to improve the design and rebuild components. Avoid leaving everything until the last moment, as practical implementation often deviates from theoretical plans.

Appendices

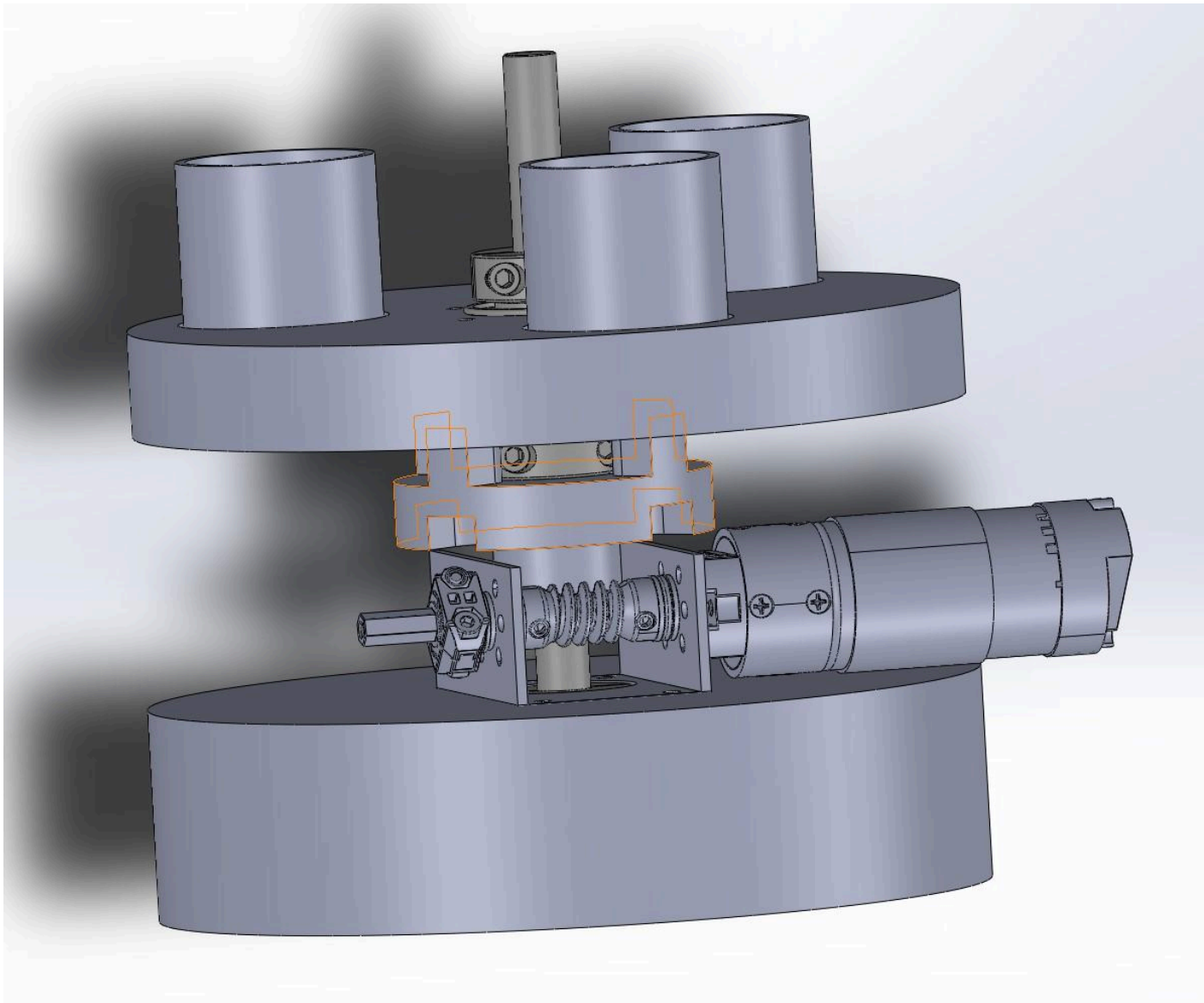
Appendix A: Bill of Materials Images

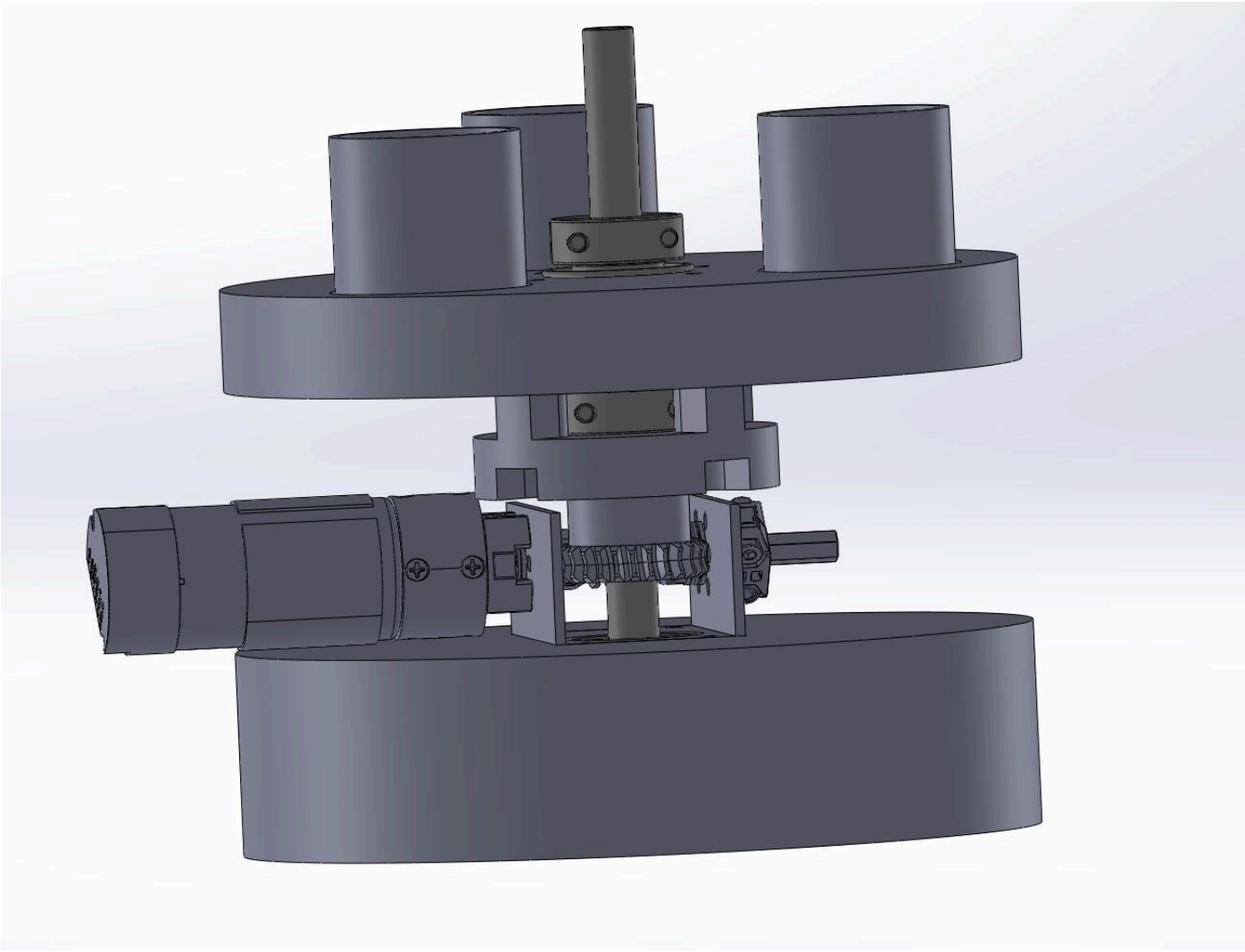
Part Name	Quantity	Cost ($)	Source Link
TCS3200 - Color Sensor	1	22.02	Amazon.com: Teyleten Robot GY-31 TCS3200 TCS230 Color Sensor Module Color Recognition Sensor Module for Arduino : Electronics
cytron md10c r3 motor controllers x 4	1	13.90	10Amp 5V-30V DC Motor Driver (cytron.io)
Ball Bearing	2	34.88	https://www.mcmaster.com/catalog/130/1415/6383K241
Linear Motion Shaft	1	6.59	https://www.mcmaster.com/catalog/130/1356/6061K103
Extra-Grip Clamping Two-Piece Shaft Collar	4	46.76	https://www.mcmaster.com/catalog/130/1466/8386K35
2314 Series Brass, MOD 1.25, Hub Mount Worm Gear	1	16.99	https://www.gobilda.com/2314-series-brass-mod-1-25-hub-mount-worm-gear-14mm-bore-28-tooth/
2313 Series Stainless Steel, MOD 1.25 Worm	1	14.99	https://www.gobilda.com/2313-series-stainless-steel-mod-1-25-worm-6mm-d-bore-34mm-length/
5204 Series Yellow Jacket Planetary Gear Motor	1	46.99	https://www.gobilda.com/5204-series-yellow-jacket-planetary-gear-motor-13-7-1-ratio-80mm-length-8mm-rex-shaft-435-rpm-3-3-5v-encoder/

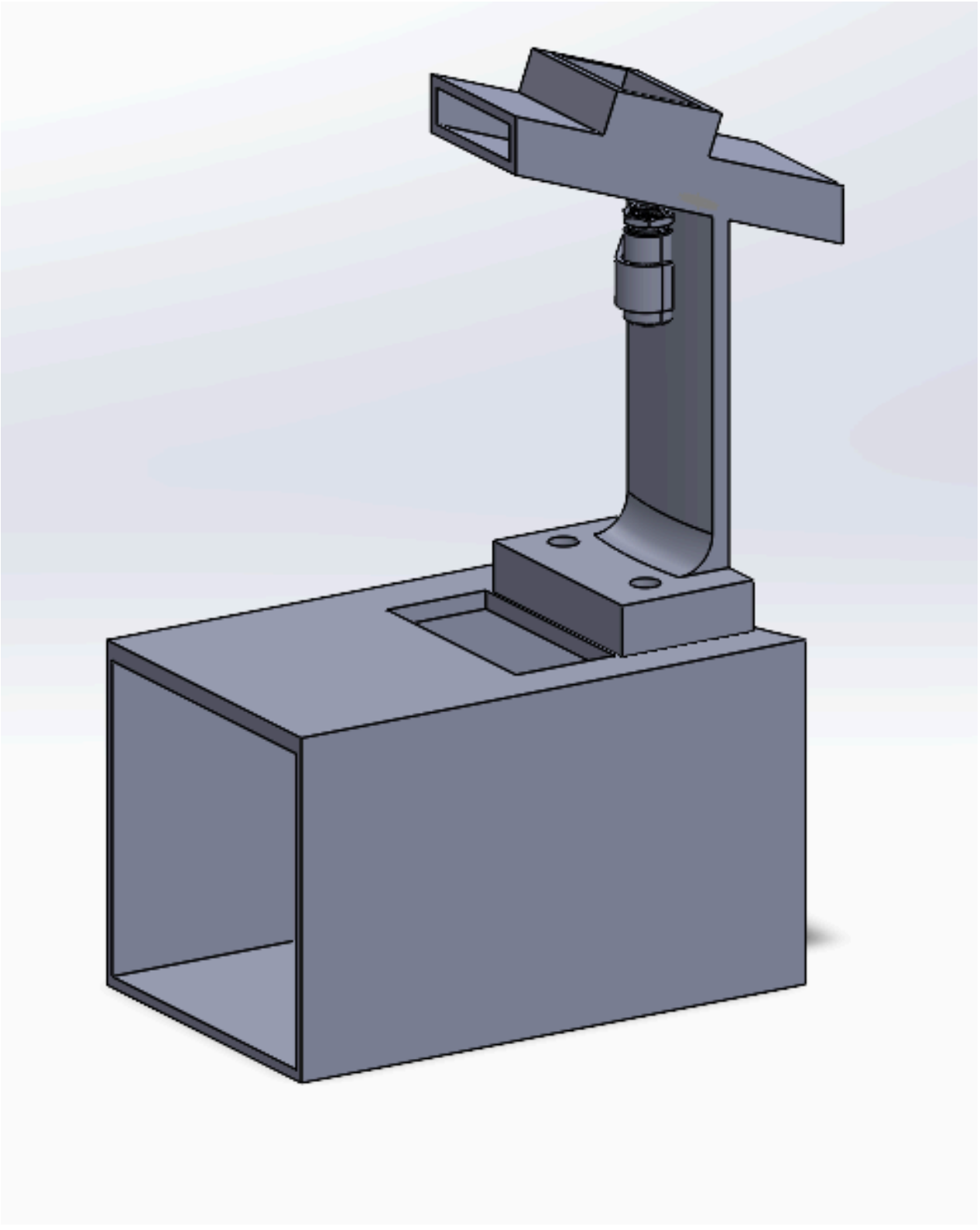
Shim and belleville washer	10	5.99	physical hardware store
Screws and nuts	8	4.99	physical hardware store
Routing Clamp (Holding Solenoid) 3192T44	1	19.98	https://www.mcmaster.com/products/clamps/routing-clamps-2~/plastic-routing-clamps-8/
Solenoid 69905K111	1	41.28	https://www.mcmaster.com/products/solenoids/solenoids~/
Encoder Breakout Cable	1	8.86	

Appendix B: CAD Images

Device integration image

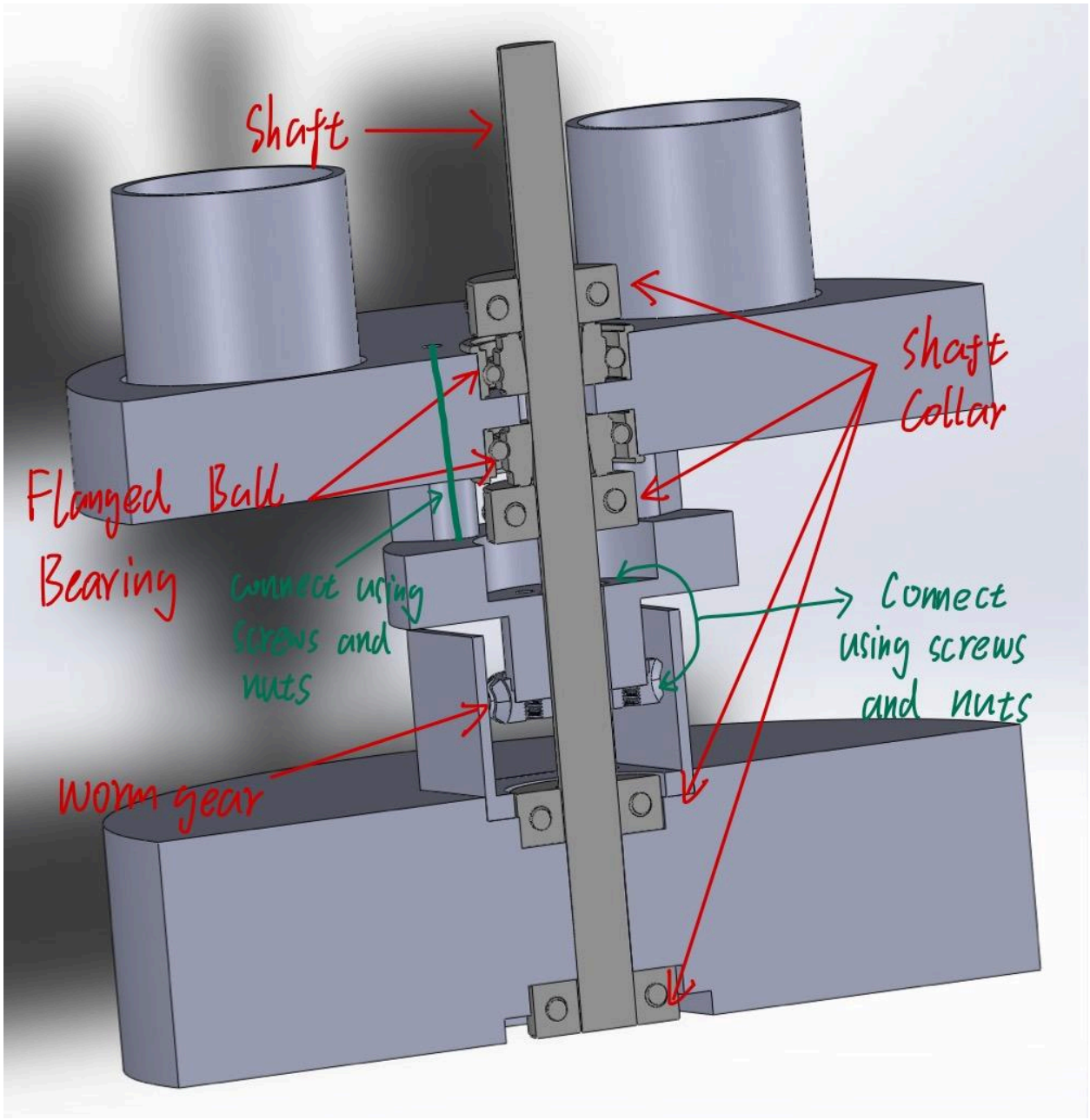




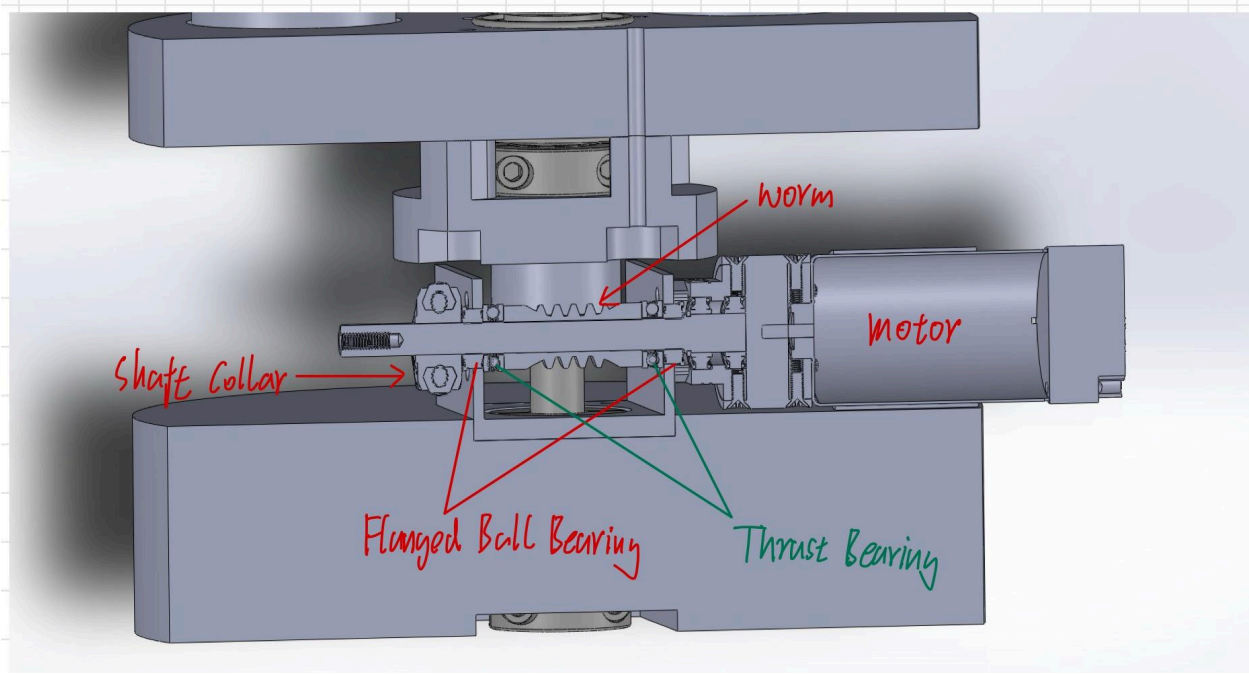


Transmission system details

1.Turntable and gears



2.Motor output shaft



Appendix C: Code Screenshots

```
#include <ESP32Encoder.h>
#include <Preferences.h>

// Motor Control Definitions
#define BIN_1 26
#define LED_PIN 13

// Color Sensor Definitions
#define S2 15
#define S3 32
#define sensorOut 25

// Solenoid Control Pin
#define SOLENOID_PIN 12

// Button Pin
#define BTN 14

ESP32Encoder encoder;
Preferences prefs; // Used for NVS storage

// Debugging and Tuning Constants
const int CPR = 14;                      // counts Per Revolution of the encoder
const float GEAR_RATIO = 383.6;  // Gear ratio =(motor gearbox)13.7 * (worm gear)28 = 383.6
const float COUNTS_PER_DEGREE = CPR * GEAR_RATIO / 360.0; // Encoder counts per degree

// Movement and Position Variables
volatile int currentCounts = 0;
bool motorRunning = false;
unsigned long motorStartTime = 0;
unsigned long motorCheckInterval = 0.5; // Update encoder counts every 0.5 ms
int currentPosition = 0;
int targetPosition = 0;

// Declare color detection variables
int Red = 0;
int Green = 0;
int Blue = 0;
int Frequency = 0;

// Button-related variables
const unsigned long longPressDuration = 1000; // Long press duration threshold
(milliseconds)
unsigned long buttonPressStartTime = 0;       // Button press start time
bool buttonHeld = false;                      // Whether the button is being held down
bool buttonReleased = true;                   // Ensure the button is released before
triggering again
bool sensorEnabled = false;                   // Sensor enabled status

// Independent Rotation Variables
```

```
int initialTargetPosition = 345;              // Initial rotation angle
bool initialRotationDone = false;             // Tracks if the initial rotation is completed

//----------------- New: Hardware timer and flags -----------------
hw_timer_t *timer = NULL;
volatile bool saveStateFlag = false;

void IRAM_ATTR onTimer() {
 // Timer interrupt callback function, sets a flag, execution happens in the loop
 saveStateFlag = true;
}

//----------------- New: State save and restore functions -----------------
void saveCurrentStateToNVS() {
 prefs.putInt("currentPosition", currentPosition);
 prefs.putInt("targetPosition", targetPosition);
 prefs.putBool("sensorEnabled", sensorEnabled);
 prefs.putBool("motorRunning", motorRunning);
 // More state information can be stored if needed
 Serial.println("State saved to NVS.");
}

void restoreStateFromNVS() {
 currentPosition = prefs.getInt("currentPosition", 0);
 targetPosition = prefs.getInt("targetPosition", 0);
 sensorEnabled = prefs.getBool("sensorEnabled", false);
 motorRunning = prefs.getBool("motorRunning", false);
 Serial.println("State restored from NVS.");
}

//-------------------------------------------------------

void setup() {
 Serial.begin(115200);

 // Motor initialization
 ESP32Encoder::useInternalWeakPullResistors = puType::up;
 encoder.attachHalfQuad(27, 33);
 encoder.setCount(0); // Ensure encoder starts at 0
 pinMode(BIN_1, OUTPUT);
 digitalWrite(BIN_1, LOW);

 // Solenoid initialization
 pinMode(SOLENOID_PIN, OUTPUT);
 digitalWrite(SOLENOID_PIN, LOW);

 // Color sensor initialization
 pinMode(S2, OUTPUT);
 pinMode(S3, OUTPUT);
 pinMode(sensorOut, INPUT);

 // Button initialization
```

```
 pinMode(BTN, INPUT_PULLUP);

 pinMode(LED_PIN, OUTPUT);
 digitalWrite(LED_PIN, LOW);

 //----------------- New: Initialize NVS and restore state -----------------
 prefs.begin("myStorage", false);
 restoreStateFromNVS();

 //----------------- New: Initialize hardware timer -----------------
 timer = timerBegin(0, 80, true); // Divider 80, counting frequency 1MHz
 timerAttachInterrupt(timer, &onTimer, true);
 timerAlarmWrite(timer, 1000000, true); // 1-second interrupt
 timerAlarmEnable(timer);

 Serial.println("Setup complete. Performing initial rotation...");
  // Perform initial rotation
 performInitialRotation();
}

void loop() {
 unsigned long currentTime = millis();

 // If the timer interrupt set the state save flag, execute state save here
 if (saveStateFlag) {
   saveStateFlag = false;
   saveCurrentStateToNVS();
 }

 // Check button state
 int buttonState = digitalRead(BTN);
 if (buttonState == LOW) {
   if (!buttonHeld && buttonReleased) {
     buttonHeld = true;
     buttonPressStartTime = currentTime;
   } else if (buttonHeld && (currentTime - buttonPressStartTime) >= longPressDuration) {
     sensorEnabled = !sensorEnabled;
     digitalWrite(LED_PIN, sensorEnabled ? HIGH : LOW);
     Serial.println(sensorEnabled ? "Sensor ON" : "Sensor OFF");
     buttonReleased = false;
     buttonHeld = false;
   }
 } else {
   buttonHeld = false;
   buttonReleased = true;
 }

 // Check motor state
 if (motorRunning) {
   if (currentTime - motorStartTime >= motorCheckInterval) {
     motorStartTime = currentTime;
     currentCounts = encoder.getCount();
```

```
    currentPosition = (int)(currentCounts / COUNTS_PER_DEGREE) % 360;
    if (currentPosition < 0) currentPosition += 360;

    if (abs(currentPosition - targetPosition) < 0.1) {
      motorRunning = false;
      digitalWrite(BIN_1, LOW);
      Serial.println("Motor reached target position.");
      encoder.setCount(0);
      currentPosition = 0;
      postRotationActions();
    }
  }
}

if (!sensorEnabled || motorRunning) {
  return;
}

// Step 1: Get raw frequency values for Red, Green, and Blue
Red = getRed();
Green = getGreen();
Blue = getBlue();

// Debugging information
Serial.print("Raw Red = ");
Serial.print(Red);
Serial.print("    ");
Serial.print("Raw Green = ");
Serial.print(Green);
Serial.print("    ");
Serial.print("Raw Blue = ");
Serial.println(Blue);

// Step 2: Determine action using switch-case
int detectedColor = -1;
switch (detectedColor) {
  case 0:
    if (Red < Green && Red < Blue && Red < 100) {
      Serial.println("Red detected");
      targetPosition = 120;
      startMotor(targetPosition);
    }
    break;

  case 1:
    if (Green < Red && Green < Blue && Green < 100) {
      Serial.println("Green detected");
      delay(1000); // Wait before solenoid activation
      activateSolenoid(); // Directly activate solenoid without rotation
    }
    break;
```

```
   case 2:
     if (Blue < Red && Blue < Green && Blue < 80) {
       Serial.println("Blue detected");
       targetPosition = 240;
       startMotor(targetPosition);
     }
     break;

   default:
     Serial.println("No color detected.");
     return;
 }
}

void startMotor(int target) {
 targetPosition = target;
 motorRunning = true;
 motorStartTime = millis();
 digitalWrite(BIN_1, HIGH);
 Serial.print("Motor started. Target position: ");
 Serial.println(targetPosition);
}

int getRed() {
 digitalWrite(S2, LOW);
 digitalWrite(S3, LOW);
 return pulseIn(sensorOut, LOW);
}

int getGreen() {
 digitalWrite(S2, HIGH);
 digitalWrite(S3, HIGH);
 return pulseIn(sensorOut, LOW);
}

int getBlue() {
 digitalWrite(S2, LOW);
 digitalWrite(S3, HIGH);
 return pulseIn(sensorOut, LOW);
}

void postRotationActions() {
 static bool firstRotationComplete = false;
 if (!firstRotationComplete) {
   delay(1000);
   activateSolenoid();
   firstRotationComplete = true;
   targetPosition = 360 - targetPosition;
   startMotor(targetPosition);
 } else {
   firstRotationComplete = false;
```

```
   Serial.println("Rotation cycle complete.");
 }
}

void performInitialRotation() {
 int initialTargetCounts = initialTargetPosition * COUNTS_PER_DEGREE;
 digitalWrite(BIN_1, HIGH);
 while (true) {
   currentCounts = encoder.getCount();
   if (abs(currentCounts) >= initialTargetCounts) {
     break;
   }
 }
 digitalWrite(BIN_1, LOW);
 encoder.setCount(0);
 Serial.println("Initial 340° rotation complete.");
 initialRotationDone = true;
}

void activateSolenoid() {
 digitalWrite(SOLENOID_PIN, HIGH);
 delay(2000);
 digitalWrite(SOLENOID_PIN, LOW);
 Serial.println("Solenoid activated.");
}
```