```
#include <Servo.h>
#include <math.h>
// --------------------------
// Switch Definition
// --------------------------
const int startSwitchPin = 2; // Switch wired to GND when closed
bool wasSwitchClosed = false; // For edge-detection
// --------------------------
// Color Sensor Definitions
// --------------------------
const int S0 = A0;
const int S1 = A1;
const int S2 = A2;
const int S3 = A3;
const int sensorOut = 3; // TCS3200 output pin
// --------------------------
// Servos// --------------------------
Servo bigServo; // MG996R on pin 6
Servo smallServo; // Ramp servo on pin 5
// --------------------------
// Ramp
// --------------------------
const int rampOrigin = 70; // “Green” home position
// --------------------------
// Color Calibration Data
// --------------------------
struct ColorData {
const char* name;
int red, green, blue;
float rRatio, gRatio, bRatio;
};
ColorData colors[] = {
{"Red", 194, 364, 292},
{"Green", 250, 257, 302},
{"Yellow", 147, 196, 273},{"Purple", 328, 420, 393},
{"Orange", 146, 285, 294}
};
const int numColors = sizeof(colors) / sizeof(colors[0]);
// ------------------------------------------------
// Read Vcc (in mV) by measuring the 1.1V reference
// ------------------------------------------------
long readVcc() {
ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1); //
internal 1.1V
delay(2);
ADCSRA |= _BV(ADSC);
while (ADCSRA & _BV(ADSC));
int result = ADC;
return (1100L * 1023L) / result;
}
void setupBaselineRatios() {
for (int i = 0; i < numColors; i++) {
float sum = colors[i].red + colors[i].green + colors[i].blue;
colors[i].rRatio = colors[i].red / sum;colors[i].gRatio = colors[i].green / sum;
colors[i].bRatio = colors[i].blue / sum;
}
}
unsigned long readColor(int s2State, int s3State) {
digitalWrite(S2, s2State);
digitalWrite(S3, s3State);
return pulseIn(sensorOut, LOW);
```

```
}
String detectSingleSample() {
unsigned long redReading = readColor(LOW, LOW);
unsigned long blueReading = readColor(LOW, HIGH);
unsigned long greenReading = readColor(HIGH, HIGH);
Serial.print("Raw R: "); Serial.print(redReading);
Serial.print(" G: "); Serial.print(greenReading);
Serial.print(" B: "); Serial.println(blueReading);
int maxReading = max(redReading, max(greenReading, blueReading));int minReading = min(redReading,
min(greenReading, blueReading));
if ((maxReading - minReading) < 50) {
return "None";
}
// -- OVERRIDE: only if red channel is the strongest and above 200
if (redReading > 200
&& redReading >= greenReading
&& redReading >= blueReading) {
Serial.println("Override → Red (highest red channel)");
return "Red";
}
float sum = redReading + greenReading + blueReading;
float rRatio = redReading / sum;
float gRatio = greenReading / sum;
float bRatio = blueReading / sum;
// -- nearest-neighbor on ratio space --
float minDistance = 1e9;
int minIndex = -1;for (int j = 0; j < numColors; j++) {
float dr = rRatio - colors[j].rRatio;
float dg = gRatio - colors[j].gRatio;
float db = bRatio - colors[j].bRatio;
float dist = sqrt(dr*dr + dg*dg + db*db);
if (dist < minDistance) {
minDistance = dist;
minIndex = j;
}
}
return (minIndex >= 0) ? String(colors[minIndex].name) : "None";
}
String detectColorConsecutive() {
const int MAX_SAMPLES = 10;
String lastColor = "None";
int consecutiveCount = 0;
for (int i = 1; i <= MAX_SAMPLES; i++) {
String found = detectSingleSample();
Serial.print("Sample "); Serial.print(i);Serial.print(" => "); Serial.println(found);
if (found == lastColor && found != "None") {
consecutiveCount++;
} else {
lastColor = found;
consecutiveCount = 1;
}
if (lastColor != "None" && consecutiveCount >= 2) {
Serial.print("Detected "); Serial.print(lastColor);
Serial.println(" 2× in a row → confirmed");
return lastColor;
}
delay(500);
}
Serial.println("No valid color detected consecutively.");
return "None";
}void moveRampToColor(const String& color) {
```

```
if (color == "Green") smallServo.write(rampOrigin);
else if (color == "Purple") smallServo.write(rampOrigin + 30);
else if (color == "Orange") smallServo.write(rampOrigin + 70);
else if (color == "Yellow") smallServo.write(rampOrigin - 40);
else if (color == "Red") smallServo.write(rampOrigin - 80);
else Serial.println("No valid color → no movement");
}
void setup() {
Serial.begin(9600);
pinMode(startSwitchPin, INPUT_PULLUP);
pinMode(S0, OUTPUT); pinMode(S1, OUTPUT);
pinMode(S2, OUTPUT); pinMode(S3, OUTPUT);
pinMode(sensorOut, INPUT);
digitalWrite(S0, HIGH);
digitalWrite(S1, LOW);
setupBaselineRatios();
Serial.println("Setup complete. Awaiting switch ON...");}
void loop() {
bool switchClosed = (digitalRead(startSwitchPin) == LOW);
if (!switchClosed) {
if (wasSwitchClosed) {
bigServo.detach();
smallServo.detach();
Serial.println("Switch OFF → servos detached.");
}
wasSwitchClosed = false;
delay(100);
return;
}
if (!wasSwitchClosed) {
bigServo.attach(6);
smallServo.attach(5);
Serial.println("Switch ON → servos attached, starting cycle.");
long Vcc_mV = readVcc();Serial.print("Measured Vcc: ");
Serial.print(Vcc_mV / 1000.0, 2);
Serial.println(" V");
}
wasSwitchClosed = true;
// — Sorting cycle —
Serial.println("\nRotation 1: to 0° (drop)");
bigServo.write(0); delay(1000);
Serial.println("Rotation 2: to 173° (dispense)");
bigServo.write(173); delay(1000);
Serial.println("Rotation 3: to 87° (sense)");
bigServo.write(87); delay(500);
Serial.println("Detecting color…");
String finalColor = detectColorConsecutive();
Serial.print("Final color: "); Serial.println(finalColor);
moveRampToColor(finalColor);
delay(300);Serial.println("Returning to 0° and dropping…");
bigServo.write(0); delay(300);
Serial.println("Cycle complete.");
delay(50);
}
```