Table of Contents

1.0 Introduction

The Skittles Separator illustrates some of the basic fundamentals of C++ programming language elements implemented into a microcontroller, the Arduino Uno. This project consists of a mechanism that is able to separate Skittles candy based on color into designated compartments. The project is also programmed to keep track of how many skittles of each color were passed through a color sensor. The user places a random sample of skittles into a funnel on the top of the machine and then the machine is initiated via turning a rotary dial and the separation process begins. This project applies both software and hardware engineering attributes.

1.1 Electrical Components

There are two input devices and two output devices used in this project:

Input	Output
TCS230 Color Sensor	Servo Motor
Rotary Dial	LCD Screen

The color sensor is the fundamental component of how this project will operate. It is a programmable light-to-frequency converter that uses an 8x8 array of photodiodes to obtain square waves readings that can be read by the Arduino board to determine the color. Refer to appendix A for in-depth explanation.

A rotary dial is used to initiate the machine. The rotary dial is used as a digital input rather than an analog input. So, it only reads true or false with one full turn to the left for the system to operate, and one turn back to stop it.

The LCD screen is used to print out the detected color of the individual pieces. It prints out a warning message that reads “WARNING! STANDBY MODE” when the device is yet to be initiated. Once the rotary dial is turned, the detection process begins and the screen prints the color name depending on the color sensor detection. Once the funnel is empty, a message prints notifying the user which appears for a specific time if the funnel is not reloaded. If the tunnel is not refilled within a certain time range, a report will be printed showing the count of skittles for each color.

The main mechanical component in this project is the servo motor. It is used to provide the necessary motion to move the skittle pieces from the funnel tube to under the color sensor. Also, another servo motor is used to move the angle the skittles pieces into different cups depending on color.

1.2 Mechanism

The machine is box-shaped with an open front face that contains three main shelves, a moving skittles holder piece to transport the skittles to the color sensor and a slanted plank to transport the skittles to their designated compartment. All parts were sketched on AutoCAD and then laser cut out of 2 mm foam board. The top shelf is a housing compartment for the color sensor with a circular slot for a long plastic tube to fit and hold the skittles vertically in place. The middle shelf houses a servo motor that transports each skittle under the color sensor for reading. The last shelf contains a second servo motor that moves a slanted plank to transports each skittle to a designated cup based on the skittles color. All parts and shelves are hot glued in place, as shown in *Figure 1*.

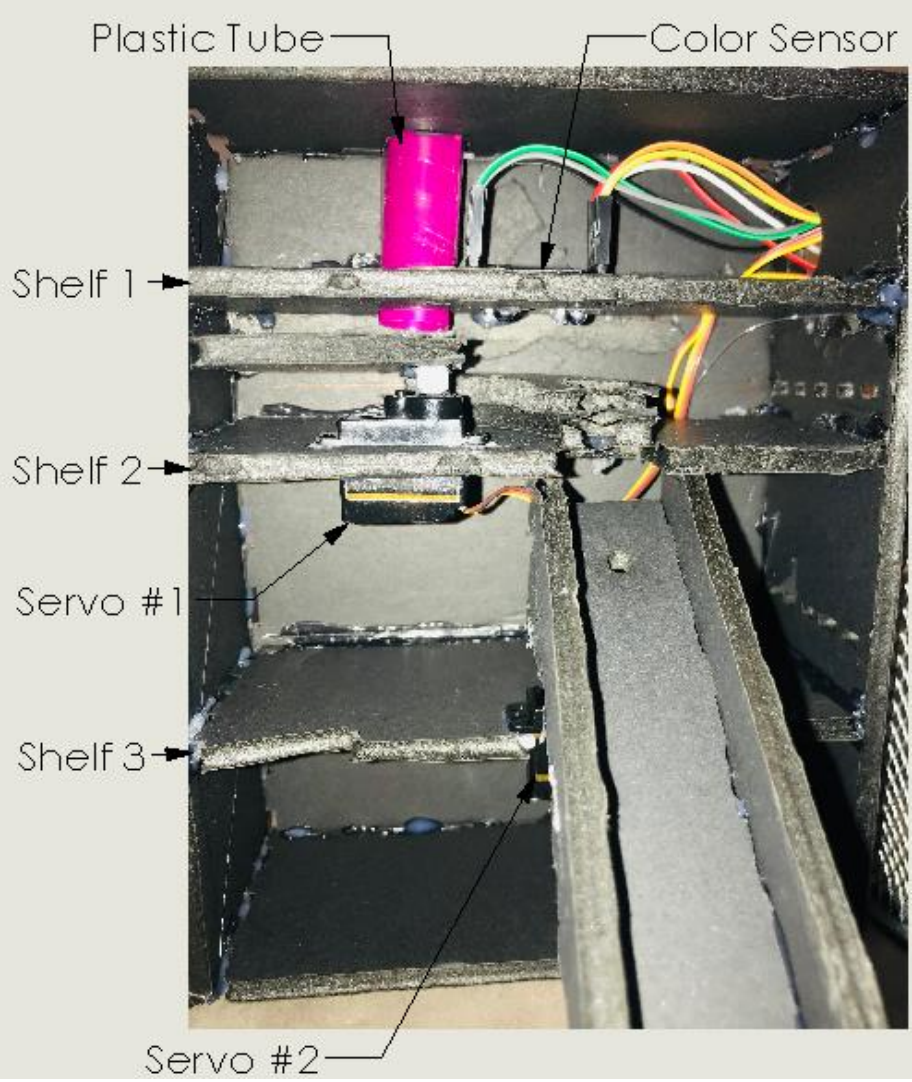


Figure 1: Skittles Separator final assembly

2.0 Algorithms

After setting up the compartment for the device with all the components, an algorithm was developed of how the color sensing mechanism, as well as the servo movement, will function. The algorithm is developed for two main aspects of the device, which are:

1- Getting the frequency reading from the color sensor, and detecting the color based off that reading.

2- Rotate the two servo motors accordingly depending on the color detected to dispense the colored objects in separate containers.

An algorithm for the LCD screen is also considered to print out various messages.

2.1 Pseudo Code

The code design was first approached by designing the pseudo code for the main functions implemented in the design. *Figure 2* shows the pseudo code of the main while loop that controls all the different functions.

```
      while(true){
      rotaryDial=Read(port 2)
            if(rotaryDial=1){
                  servoMovement();
}
            else{
                  idleScreen();
                  valueReset(red, yellow, orange, green, purple, empty);
                  topServo(calibrate);
                  bottomServo(calibrate);}
}
```

Figure 2: Pseudo code of the main design code

A digital read function constantly gets feedback from a specific port where the rotary dial is connected to. If the reading is 1, it executes the servoMovement() function which is the main function that controls the servo motors and the color sensor. If the reading is not 1, the device is in standby mode which prints out the idleScreen() message to the LCD and reset the count values for the colors to zero. Also, the two servo motors are calibrated to the default position.

The pseudo code for the servoMovement() function is shown in *Figure 3*. This function controls the servo motors based off the reading from the getColor() function. A for loop is implemented to move top servo motor to position the skittles piece under the photodiode array of the color sensor. A delay is set to give time for the sensor to obtain the frequency reading. Then, a switch function is implemented to execute a set of statements based off color detected. For example, if the sensor detects the color red, the counter for the red color is incremented by one and "Red" is printed to the LCD screen and the background light color changes to red. The bottom servo is moved depending on a predetermined angle of where the containers are placed. Then, the case for red color is stopped by using the break function.

```
      Void servoMovement(){
            for(angle between a and b){
                  topServo(angle);}
            color = getColor();
            delay(1500);
      switch (color){
            case red:
                  increment red counter
                  moveBottomServo;
                  screenPrint("red");
                  change Backlight color to red;
                  break;

            case orange:
                  //same process as red
}
}
```

Figure 3: Pseudo code showing the servoMovement() function.

3.0 Test Plan

A test plan was developed for the major functionalities of the device. It serves as a blueprint to conduct code debugging if the code is not executing properly, or the result is not as expected. The list below specifies how each function of the design will be tested.

getColor():

Before making the assembly of the device compartment, the color sensor testes on a breadboard to determine if the different frequency readings are detected properly. The serial monitor functionality will be used to print out the readings for each color hue (red, green, and blue). If the color sensor is detecting a different reading when different objects are placed on the photodiodes, then the color sensor is functioning properly and is able to distinguish between color shades. Then, when all the components are assembled, the range of each color is obtained through a trial and error process. A set of 20 pieces from each color is put through the device and from the serial monitor output, a range can be determined by analyzing the maximum and minimum detected values for each color.

ServoMovement():

For the top servo motor, the testing process will be trial and error. The Skittles are placed on the transportation piece and the servo motor moves the piece of skittles under the color sensor. Initial values for the angles are set and then the servo operates to see if the Skittles piece is moved directly under the color sensor. The angles are altered until exact positioning under the color sensor is achieved. For the top servo motor, the degree of freedom is 180 degrees horizontally. To accommodate for all five different Skittle colors, the 180 is divided by 5 to achieve equal spacing between all five compartments. Then, a test is run by inserting different colored skittles to observe the movement of the servo motor.

idleScreen():

This function prints out to the screen when the device is in standby mode which is activated by a rotary dial. It can be tested by observing if the code executes properly and prints out the specific message to the LCD screen when it is activated by the rotary dial (when reading is true).

valueReset():

The main purpose of this function is to reset the color counter every time a run is over, and the rotary dial is turned to read false and stop the device. It can be tested by observing if the counter printed out to the screen is reset every time.

4.0 Implementation and Testing

4.1 Component Schematic and Revision

All the components are connected to the Arduino board for testing. Powering the color sensor correctly was done by referring to the datasheet of the particular model number. There are 3 input pins that get the RGB reading and one output pin. The configuration and connections of all the components used are shown in *Figure 4.*

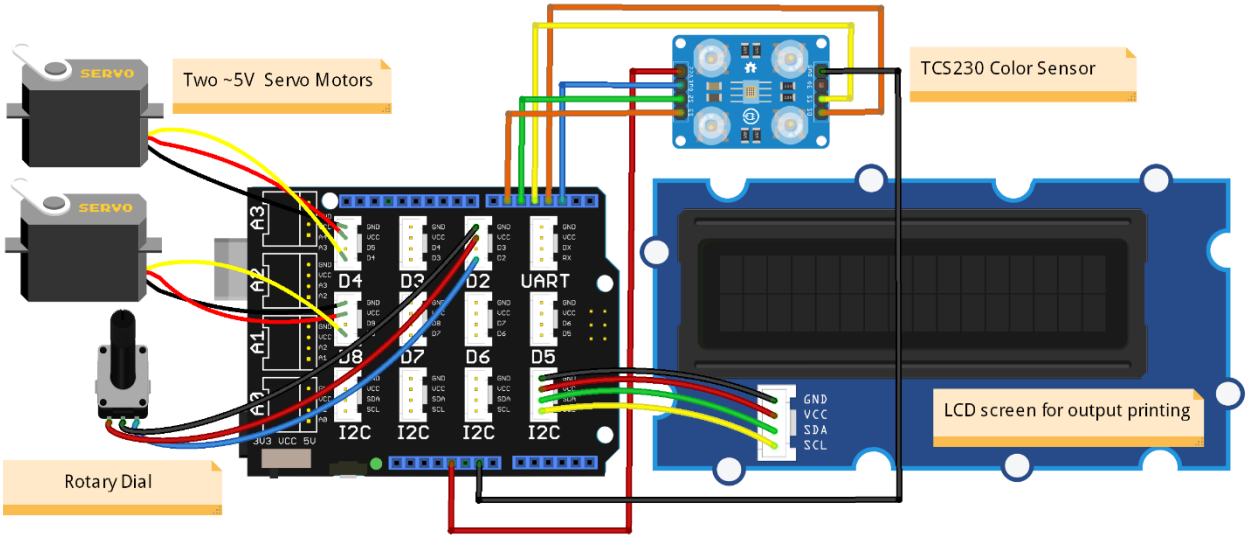


Figure 4: Diagram showing different components used in the project

Two hardware components were changed compared to the original project proposal. First, the 9V battery was discarded because it did not provide the system with enough current. Also, it caused signal issues between the Arduino board and the servo motors. So, a connection using a USB cable to the computer was a better alternative. Second, the push button was changed to a rotary dial due to some programming issues with making the input of the push button always true until pressed again. Therefore, the rotatory dial allowed that feature by rotating the dial to alternate between true and false.

4.2 Code Testing and Result Analysis

The list below shows the results of the conducted test plan with the results and the iteration process. The full code can be found in *Appendix B*.

getColor():

Using the serial print function, a range for each color was set. 20 pieces of each color were tested for frequency values and the maximum and minimum values were set as the range. Using IF statement, when the color frequency falls within the range, it must detect the color as appropriate. Testing the function through trial and error results in a successful trial with minor errors due to lighting conditions and the sensor frequency.

```
1.    /*
2.         These conditions are what determines the color under the sensor.
3.         There is a range for each condition because of the uncertainty
4.         and accuracy of the sensor.
5.      */
6.      if (R <= 25 & R >= 22 & G <= 39 & G >= 33 & B <= 28 & B >= 23) {
7.        color = 1; // Detects red
8.      }
9.      if (R <= 20 & R >= 16 & G <= 34 & G >= 28 & B <= 27 & B >= 21) {
10.       color = 2; // Detects Orange
11.     }
12.     if (R <= 30 & R >= 22 && G <= 30 & G >= 21 && B <= 26 & B >= 22) {
13.       color = 3; // Detects Green
14.     }
15.     if (R <= 20 & R >= 15 & G <= 25 & G >= 20 & B <= 24 & B >= 15) {
16.       color = 4; // Detects Yellow
17.     }
18.     if (R <= 32 & R >= 26 & G <= 43 & G >= 35 & B <= 31 & B >= 26) {
19.       color = 5; // Detects purple
20.     }
21.     if (R <= 38 & R >= 34 & G <= 43 & G >= 37 & B <= 31 & B >= 26)
22.       color = 6; // Detects Nothing
23.
24.     return color; //return the color number to the servoMovement() function
```

ServoMovement():

For the top servo motor, three angles were set for three positions. The first angle positions the skittles holder directly under the plastic tube to reload with a piece of skittles, the second angle places the piece of skittles under the color sensor for reading, and the third angle positions the skittles piece to drop through a hole onto the slanted plank to place the piece the designated compartment. After, a series of trial an error, the three angles were determined to be: first angle 125 (under the plastic tube), second angles 73 (under color sensor), and third angle 20 (through the hole onto the plank).

```
1.    for (int i = 73; i > 20; i--) {
2.        topServo.write(i);
3.        delay(2);
4.      }
5.      delay(200);
6.
7.      for (int i = 20; i < 125; i++) {
8.        topServo.write(i);
9.        delay(2);
10.     }
```

These three movements are looped for the machine to operate continuously until stopped by turning the rotary dial.

For the bottom servo motor, the five angles for the different compartment positions are stored in an array.

```
int servoAngles[] = {25, 50, 80, 110, 140}
```

Then, the values are passed in from the array to the servo motor function to move the slanted plank depending on which case in the SWITCH function is getting executed which depends on the color detected. The testing process consisted of multiple tries to get the right delay between the time the color sensor takes to detect the color and the time for the bottom servo to move the plank to the angle of the color detected compartment.

```
11.   switch (color) {
12.       case 1:
13.         redCounter++;  //increases the counter value by 1 each time it's detected
14.         bottomServo.write(servoAngles[0]);  //gets angle from array and moves servo
15.         lcdPrint("      RED");  //prints color name to the lcd
16.         lcdMoveCursor(0, 1);    // moves cursor to second line
17.         lcdPrint("skittles # ");
18.         lcdPrint(redCounter);  //prints out the number of skittles
19.         lcdBacklightColour(255, 0, 0);  //changes lcd backlight color
20.         delay(1500);  // sets a delay of 1.5 seconds
21.         lcdClear();  //clears lcd screen
22.         break;  //breaks the execution sequence for case 1
```

idleScreen():

Using an IF statement, the function detects when the device is idle (rotary dial reading is false), and prints out a fading warning screen. Two FOR loops were used to create the fading effect – one to fade down to 0 RGB reading and one to fade up to 255 RGB reading. Testing the function was simply done by physically rotating the dial and observing whether the screen print function gets executed. The test was successful after the modification of the fading FOR loop by only changing the red RGB reading in changeBackLightColor function.

```
for (int i = 255; i > 0; i--) //first for loop runs from 255 to 0
{
lcdBacklightColour(i, 0, 0);
}
for (int i = 0; i < 255; i++) //second for loop runs from 0 to 255
{
lcdBacklightColour(i, 0, 0);
}
```

valueReset():

Using pass-by-reference, each color counter parameter is passed into the valueReset() function. Inside the function each parameter gets reassigned to zero when the condition of the rotary dial is false. Testing the function is done by completing a run, then stopping the device. When turned on again, the values were reset to zero, indicating a successful trial.

```
// This function is used to reset the counter values to zero after run is complete
void valuesReset(int& redCounter, int& yellowCoarunter, int& orangeCounter
                  , int& greenCounter, int& purpleCounter, int& emptyHole) {

   redCounter = 0;
   yellowCounter = 0;
   orangeCounter = 0;
   greenCounter = 0;
   purpleCounter = 0;
   emptyHole = 0;
   }
```

5.0 Reflection and Conclusion

Ultimately, after a long debugging and trial error process, the device is successfully able to separate the skittles pieces based off color into the designated cups. A small percentage of error is still present due to various aspects – ambient light and the accuracy of the color sensor. Overall, the design went through multiple iteration processes. The main changes to the implementation was modifying the ranges for each color case to reduce error. The design is heavily dependent on the perfect alignments by the servo motors. So, during the implementation of the code, the angles were modified until the system was able to run smoothly with minimal errors.

The list below illustrates all the various functions implemented into the design.

- **Expressions and functions**: The code consists of a variety of functions that each perform certain tasks. For example, one function consists of IF statements to determine the color determined by the color sensor. Another function moves the servo motor depending on the detected color**.** Expressions are used to evaluate the angles and frequency reading from the color sensor.

- **If statements and the switch function**: If statements are used to set a range of frequencies read by the color sensor for the 5 identified colors (red, orange, green, yellow, and purple) and given a number from 1 to 5. Once that is obtained, the switch function is then used to operate the servo motors to move at the required angle depending on the case number (1-5) to drop the skittles pieces in the designated container.

- **While and for loops**: It is essential that the code runs in a continuous loop. A while loop is used to render the code always true (running) unless the rotary dial is turned, which terminates the program. For loops are used to move the servo motors to the certain predefined angles.

- **Arrays**: The sorting mechanism is dependent on a servo motor that moves to certain predetermined angles. These angles are stored in an array of size of 5 (each angle depends on where the container is) and then the designated angle can be retrieved from the array by passing the address into the servo motor.

- **Pass-by-Reference:** One feature of the design is the ability to keep count of each piece by color and then print out a report to the LCD screen with all the color numbers stored. The counter needs to reset every time a new run is started which is possible to achieve by passing in all the counter values by reference, then resigning all the values to zero. If the values were passing in by value, it would only create a copy of each variable and not change the original values.

6.0 Appendices

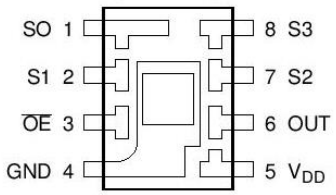
6.1 Appendix A: TCS230 color light-to-frequency converter datasheet

TAOS TEXAS ADVANCED OPTOELECTRONIC SOLUTIONS™

TCS230
PROGRAMMABLE
COLOR LIGHT-TO-FREQUENCY CONVERTER
TAOS046 - FEBRUARY 2003

- **High-Resolution Conversion of Light Intensity to Frequency**
- **Programmable Color and Full-Scale Output Frequency**
- **Communicates Directly With a Microcontroller**
- **Single-Supply Operation (2.7 V to 5.5 V)**
- **Power Down Feature**
- **Nonlinearity Error Typically 0.2% at 50 kHz**
- **Stable 200 ppm/°C Temperature Coefficient**
- **Low-Profile Surface-Mount Package**

SOIC PACKAGE
(TOP VIEW)

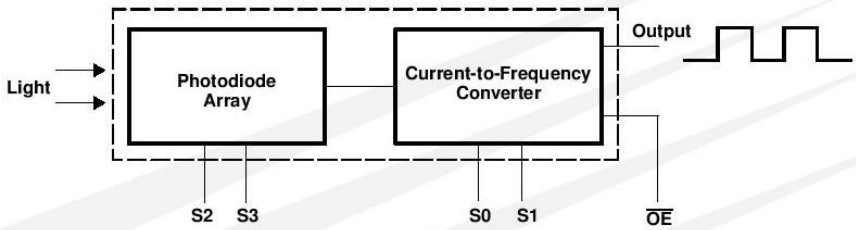


Description

The TCS230 programmable color light-to-frequency converter combines configurable silicon photodiodes and a current-to-frequency converter on single monolithic CMOS integrated circuit. The output is a square wave (50% duty cycle) with frequency directly proportional to light intensity (irradiance). The full-scale output frequency can be scaled by one of three preset values via two control input pins. Digital inputs and digital output allow direct interface to a microcontroller or other logic circuitry. Output enable ($\overline{OE}$) places the output in the high-impedance state for multiple-unit sharing of a microcontroller input line.

The light-to-frequency converter reads an 8 x 8 array of photodiodes. Sixteen photodiodes have blue filters, 16 photodiodes have green filters, 16 photodiodes have red filters, and 16 photodiodes are clear with no filters. The four types (colors) of photodiodes are interdigitated to minimize the effect of non-uniformity of incident irradiance. All 16 photodiodes of the same color are connected in parallel and which type of photodiode the device uses during operation is pin-selectable. Photodiodes are 120 µm x 120 µm in size and are on 144-µm centers.

Functional Block Diagram



Terminal Functions

TERMINAL NAME	TERMINAL NO.	I/O	DESCRIPTION
GND	4		Power supply ground. All voltages are referenced to GND.
$\overline{OE}$	3	I	Enable for f_o (active low).
OUT	6	O	Output frequency (f_o).
S0, S1	1, 2	I	Output frequency scaling selection inputs.
S2, S3	7, 8	I	Photodiode type selection inputs.
V_{DD}	5		Supply voltage

Table 1. Selectable Options

S0	S1	OUTPUT FREQUENCY SCALING (f_o)		S2	S3	PHOTODIODE TYPE
L	L	Power down		L	L	Red
L	H	2%		L	H	Blue
H	L	20%		H	L	Clear (no filter)
H	H	100%		H	H	Green

Available Options

DEVICE	T_A	PACKAGE - LEADS	PACKAGE DESIGNATOR	ORDERING NUMBER
TCS230	- 25°C to 85° C	SOIC-8	D	TCS230D

Absolute Maximum Ratings over operating free-air temperature range (unless otherwise noted)†

Supply voltage, V_{DD} (see Note 1) . . . . . 6 V
Input voltage range, all inputs, V_I . . . . . - 0.3 V to V_{DD} + 0.3 V
Operating free-air temperature range, T_A . . . . . 0°C to 70°C
Storage temperature range . . . . . - 25°C to 85°C
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds . . . . . 260°C

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: All voltage values are with respect to GND.

Recommended Operating Conditions

		MIN	NOM	MAX	UNIT
Supply voltage, V_{DD}		2.7	5	5.5	V
High-level input voltage, V_{IH}	V_{DD} = 2.7 V to 5.5 V	2		V_{DD}	V
Low-level input voltage, V_{IL}	V_{DD} = 2.7 V to 5.5 V	0		0.8	V
Operating free-air temperature range, T_A		0		70	°C

APPLICATION INFORMATION

Power supply considerations

Power-supply lines must be decoupled by a 0.01-μF to 0.1-μF capacitor with short leads mounted close to the device package.

Input interface

A low-impedance electrical connection between the device $\overline{OE}$ pin and the device GND pin is required for improved noise immunity.

Output interface

The output of the device is designed to drive a standard TTL or CMOS logic input over short distances. If lines greater than 12 inches are used on the output, a buffer or line driver is recommended.

Photodiode type (color) selection

The type of photodiode (blue, green, red, or clear) used by the device is controlled by two logic inputs, S2 and S3 (see Table 1).

Output frequency scaling

Output-frequency scaling is controlled by two logic inputs, S0 and S1. The internal light-to-frequency converter generates a fixed-pulsewidth pulse train. Scaling is accomplished by internally connecting the pulse-train output of the converter to a series of frequency dividers. Divided outputs are 50%-duty cycle square waves with relative frequency values of 100%, 20%, and 2%. Because division of the output frequency is accomplished by counting pulses of the principal internal frequency, the final-output period represents an average of the multiple periods of the principle frequency.

The output-scaling counter registers are cleared upon the next pulse of the principal frequency after any transition of the S0, S1, S2, S3, and $\overline{OE}$ lines. The output goes high upon the next subsequent pulse of the principal frequency, beginning a new valid period. This minimizes the time delay between a change on the input lines and the resulting new output period. The response time to an input programming change or to an irradiance step change is one period of new frequency plus 1 μS. The scaled output changes both the full-scale frequency and the dark frequency by the selected scale factor.

The frequency-scaling function allows the output range to be optimized for a variety of measurement techniques. The scaled-down outputs may be used where only a slower frequency counter is available, such as low-cost microcontroller, or where period measurement techniques are used.

Measuring the frequency

The choice of interface and measurement technique depends on the desired resolution and data acquisition rate. For maximum data-acquisition rate, period-measurement techniques are used.

Output data can be collected at a rate of twice the output frequency or one data point every microsecond for full-scale output. Period measurement requires the use of a fast reference clock with available resolution directly related to reference clock rate. Output scaling can be used to increase the resolution for a given clock rate or to maximize resolution as the light input changes. Period measurement is used to measure rapidly varying light levels or to make a very fast measurement of a constant light source.

Maximum resolution and accuracy may be obtained using frequency-measurement, pulse-accumulation, or integration techniques. Frequency measurements provide the added benefit of averaging out random- or high-frequency variations (jitter) resulting from noise in the light signal. Resolution is limited mainly by available counter registers and allowable measurement time. Frequency measurement is well suited for slowly varying or constant light levels and for reading average light levels over short periods of time. Integration (the accumulation of pulses over a very long period of time) can be used to measure exposure, the amount of light present in an area over a given time period.

6.2 Appendix B: Full Code

```
34.  //including required header files
35.  #include <Wire.h>
36.  #include <SoftwareSerial.h>
37.  #include <Servo.h>
38.  #include <seeed-kit.h>
39.
40.  //defining each port in the color sensor
41.
42.  const int S0 = 8;
43.  const int S1 = 9;
44.  const int S2 = 10;
45.  const int S3 = 11;
46.  const int sensorOut = 12;
47.
48.  //initializing various global identifiers to zero
49.
50.  int frequency = 0;
51.  int color = 0;
52.  int servoAngles[] = {25, 50, 80, 110, 140}; //defining an array with 5 angle readings
53.  int redCounter = 0;
54.  int yellowCounter = 0;
55.  int orangeCounter = 0;
56.  int greenCounter = 0;
57.  int purpleCounter = 0;
58.  int emptyHole = 0;
59.
60.  //initializing servo motors
61.  Servo topServo;
62.  Servo bottomServo;
63.
64.  /*
65.     void setup is where the function declaration and port initializing
66.     is found. The code inside is executed only once at the beginning of
67.     the program and then never again.
68.  */
69.  void setup() {
70.    /*
71.        the pinMode funtion is used to identify each port
72.        on the arduino board as an input or output port.
73.    */
74.    pinMode(S0, OUTPUT);
75.    pinMode(S1, OUTPUT);
76.    pinMode(S2, OUTPUT);
77.    pinMode(S3, OUTPUT);
78.    pinMode(sensorOut, INPUT);
79.
80.    lcdInit(); //initializes lcd screen
81.
82.
83.    /*
84.        according to the TC230 color sensor datasheet,
85.        setting pin S0 and S1 to HIGH will set the reading
86.        frequency scale to 100%
87.    */
88.    digitalWrite(S0, HIGH);
89.    digitalWrite(S1, HIGH);
90.
91.    //attaching each servo motor to certain digital port
```

```
92.     topServo.attach(5);
93.     bottomServo.attach(4);
94.     Serial.begin(9600);  // begins the serial monitor function to 9600 baud
95.   }
96.
97.   /*
98.      the void loop function loops the code inside the block consecutively,
99.      which allows the program to change and respond. It is used to actively
100.     control the Arduino board.
101.  */
102.  void loop() {
103.
104.    int rotaryDial;
105.    /*
106.       The while loop is the main function of the program and connects
107.       everything together.
108.    */
109.    while (true) {
110.
111.      rotaryDial = digitalRead(2); //the value of the rotary dial is constantly being read
112.
113.      if (rotaryDial > 0) {
114.        lcdClear();  //clears the screen from previous output
115.        servoMovement();  //executes the servoMovement function below
116.      }
117.      /*
118.         the else function contains functions that are executed
119.         when the sorting mechanism is turned off.
120.      */
121.      else {
122.        warningScreen();  //executes the warning screen function
123.        valuesReset(redCounter, yellowCounter, orangeCounter
124.                    , greenCounter, purpleCounter, emptyHole); //resets the color count val
                                ues to zero
125.        topServo.write(125); //moves the top servo to the middle
126.        bottomServo.write(90);  //moves the bottom servo to the middle
127.      }
128.    }
129.
130.  }
131.  /*
132.     This function is what controls the servo motors based off
133.     the color reading.
134.  */
135.  void servoMovement() {
136.    delay(1500);
137.
138.    //
139.    for (int i = 125; i > 73; i--) {
140.      topServo.write(i);
141.      delay(2);
142.    }
143.    delay(1500);
144.
145.    color = readColor(); //calls the readColor function which detects the color
146.    delay(10);
147.    /*
148.       The switch functions executes a certain set of code
149.       based off the color determined. Here, there are 6 cases.
150.    */
```

```
151.    switch (color) {
152.      case 1:
153.        redCounter++;  //increases the counter value by 1 each time it's detected
154.        bottomServo.write(servoAngles[0]);  //gets angle from array and moves servo
155.        lcdPrint("      RED");  //prints color name to the lcd
156.        lcdMoveCursor(0, 1);    // moves cursor to second line
157.        lcdPrint("skittles # ");
158.        lcdPrint(redCounter);  //prints out the number of skittles
159.        lcdBacklightColour(255, 0, 0);  //changes lcd backlight color
160.        delay(1500);  // sets a delay of 1.5 seconds
161.        lcdClear();  //clears lcd screen
162.        break;  //breaks the execution sequence for case 1
163.
164.      case 2:
165.        orangeCounter++;
166.        bottomServo.write(servoAngles[1]);
167.        lcdPrint("     ORANGE");
168.        lcdMoveCursor(0, 1);
169.        lcdPrint("skittles # ");
170.        lcdPrint(orangeCounter);
171.        lcdBacklightColour(255, 100, 0);
172.        delay(1500);
173.        lcdClear();
174.
175.        break;
176.
177.      case 3:
178.        greenCounter++;
179.        bottomServo.write(servoAngles[2]);
180.        lcdPrint("     GREEN");
181.        lcdMoveCursor(0, 1);
182.        lcdPrint("skittles # ");
183.        lcdPrint(greenCounter);
184.        lcdBacklightColour(0, 255, 0);
185.        delay(1500);
186.        lcdClear();
187.
188.        break;
189.
190.      case 4:
191.        yellowCounter++;
192.        bottomServo.write(servoAngles[3]);
193.        lcdPrint("     YELLOW");
194.        lcdMoveCursor(0, 1);
195.        lcdPrint("skittles # ");
196.        lcdPrint(yellowCounter);
197.        lcdBacklightColour(255, 255, 0);
198.        delay(1500);
199.        lcdClear();
200.
201.        break;
202.
203.      case 5:
204.        purpleCounter++;
205.        bottomServo.write(servoAngles[4]);
206.        lcdPrint("     PURPLE");
207.        lcdMoveCursor(0, 1);
208.        lcdPrint("skittles # ");
209.        lcdPrint(purpleCounter);
210.        lcdBacklightColour(128, 0, 128);
211.        delay(1500);
```

```
212.       lcdClear();
213.
214.       break;
215.
216.     case 6:
217.       /*
218.          if the compartment where the skittles is supposed
219.          to be is empty for 2 runs, it will execute the a report
220.          for the number of skittles for each color
221.       */
222.       if (emptyHole == 2) {
223.         colorRunDown();
224.         delay(10000);
225.         lcdClear();
226.       }
227.       emptyHole++;
228.       bottomServo.write(servoAngles[2]);
229.       lcdMoveCursor(2, 0);
230.       lcdPrint("compartment");
231.       lcdMoveCursor(4, 1);
232.       lcdPrint("is empty");
233.       lcdBacklightColour(255, 255, 255);
234.       delay(1500);
235.       lcdClear();
236.
237.     case 0:
238.       break;
239.   }
240.   delay(300);
241.
242.   for (int i = 73; i > 20; i--) {
243.     topServo.write(i);
244.     delay(2);
245.   }
246.   delay(200);
247.
248.   for (int i = 20; i < 125; i++) {
249.     topServo.write(i);
250.     delay(2);
251.   }
252.   color = 0;
253. }
254.
255. /*
256.    The following function detects the color based off the
257.    frequency reading. The digitalWrite value depends on
258.    what color hue is being read from the data sheet.
259. */
260. int readColor() {
261.   // Setting red filtered photodiodes to be read
262.   digitalWrite(S2, LOW);
263.   digitalWrite(S3, LOW);
264.   // Reading the output equency
265.   frequency = pulseIn(sensorOut, LOW);
266.   int R = frequency;
267.   // Printing the value on the serial monitor
268.   Serial.print("R= ");//printing name
269.   Serial.print(frequency);//printing RED color frequency
270.   Serial.print("  ");
271.   delay(50);
272.
```

```
  // Setting Green filtered photodiodes to be read
  digitalWrite(S2, HIGH);
  digitalWrite(S3, HIGH);
  // Reading the output frequency
  frequency = pulseIn(sensorOut, LOW);
  int G = frequency;
  // Printing the value on the serial monitor
  Serial.print("G= ");//printing name
  Serial.print(frequency);//printing RED color frequency
  Serial.print("  ");
  delay(50);

  // Setting Blue filtered photodiodes to be read
  digitalWrite(S2, LOW);
  digitalWrite(S3, HIGH);
  // Reading the output frequency
  frequency = pulseIn(sensorOut, LOW);
  int B = frequency;
  // Printing the value on the serial monitor
  Serial.print("B= ");//printing name
  Serial.print(frequency);//printing RED color frequency
  Serial.println("  ");
  delay(50);

  /*
     These conditions are what determines the color under the sensor.
     There is a range for each condition because of the uncertainty
     and accuracy of the sensor.
  */
  if (R <= 25 & R >= 22 & G <= 39 & G >= 33 & B <= 28 & B >= 23) {
    color = 1; // Detects red
  }
  if (R <= 20 & R >= 16 & G <= 34 & G >= 28 & B <= 27 & B >= 21) {
    color = 2; // Detects Orange
  }
  if (R <= 30 & R >= 22 && G <= 30 & G >= 21 && B <= 26 & B >= 22) {
    color = 3; // Detects Green
  }
  if (R <= 20 & R >= 15 & G <= 25 & G >= 20 & B <= 24 & B >= 15) {
    color = 4; // Detects Yellow
  }
  if (R <= 32 & R >= 26 & G <= 43 & G >= 35 & B <= 31 & B >= 26) {
    color = 5; // Detects purple
  }
  if (R <= 38 & R >= 34 & G <= 43 & G >= 37 & B <= 31 & B >= 26)
    color = 6; // Detects Nothing

  return color; //return the color number to the servoMovement() function
}

// this function prints out to the screen when device is idle
void warningScreen() {

  lcdMoveCursor(4, 0);
  lcdPrint("WARNING:");
  lcdMoveCursor(0, 1);
  lcdPrint("STANDSTILL MODE!");

  /*
     the for loop is used to add a red fading affect to the lcd screen
  */
```

```
334.    for (int i = 255; i > 0; i--) //first for loop runs from 255 to 0
335.    {
336.      lcdBacklightColour(i, 0, 0);
337.    }
338.    for (int i = 0; i < 255; i++) //second for loop runs from 0 to 255
339.    {
340.      lcdBacklightColour(i, 0, 0);
341.    }
342.
343.  }
344.  // function for printing out the color report at the end of each run
345.  void colorRunDown() {
346.    lcdPrint("R=");
347.    lcdPrint(redCounter);
348.    lcdMoveCursor(6, 0);
349.    lcdPrint("G=");
350.    lcdPrint(greenCounter);
351.    lcdMoveCursor(13, 0);
352.    lcdPrint("Y=");
353.    lcdPrint(yellowCounter);
354.    lcdMoveCursor(4, 1);
355.    lcdPrint("P=");
356.    lcdPrint(purpleCounter);
357.    lcdMoveCursor(10, 1);
358.    lcdPrint("O=");
359.    lcdPrint(orangeCounter);
360.  }
361.  // This function is used to reset the counter values to zero after run is complete
362.  void valuesReset(int& redCounter, int& yellowCoarunter, int& orangeCounter
363.                   , int& greenCounter, int& purpleCounter, int& emptyHole) {
364.
365.    redCounter = 0;
366.    yellowCounter = 0;
367.    orangeCounter = 0;
368.    greenCounter = 0;
369.    purpleCounter = 0;
370.    emptyHole = 0;
371.
372.  }
```