COLOR SORTING USING ARDUINO and

COLOR SENSOR TCS230

**A PROJECT REPORT**

***Submitted by***

170040320 J.V.N.D.S.R.PRASAD

170040335 K.SAI RAM

170040339 K.SADHVIK

170040639 P.SIVA SAI DINESH

170040998 K.PREM KUMAR

Under the guidance of

**Mr. K. UDAY KIRAN SIR**

**ASSISTANT PROFESSOR**



**K L E F**

**Green Fields, VADDESWARAM- 522502**

**Guntur District, A.P., INDIA.**

**2017-18**

**DEPARTMENT OF BASIC ENGINEERING SCIENCES-1**

**BASIC ENGINEERING MEASUREMENTS (17 GN 1003)**

CERTIFICATE

## This is to certify that J.V.N.D.S.R.PRASAD, K.SAI RAM, K.SADHVIK, P.SIVA SAI DINESH, K.PREM KUMAR bearing Univ. Regd no. 170040320, 170040335,170040339,170040639,170040998 of section 22 studying I/IV B. Tech in KLEF has satisfactorily completed project in the semester II during the academic year 2017 – 2018.

**Signature of the HOD Signature of Course Coordinator**

**TABLEOFCONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTERNO.** | **TITLE** | **PAGENO.** |

**1 ABSTRACT 4**

**2 INTRODUCTION 5**

**3 COMPONENTS 10**

**4 SOFTWARE REQUIREMNTS 22**

**5 WORKING PRINCIPLE 29**

**6 RESULTS 34**

**7 7.1 ADVANTAGES 35**

**7.2 DISADVANTAGES 35**

**7.3 FUTURE ENHANCEMENTS 35**

**8 CONCLUSION 36**

**ABSTRACT**

Sorting of products is a very difficult industrial process. Continuous manual sorting creates consistency issues. This paper describes a working prototype designed for automatic sorting of objects based on the color. TCS230 sensor was used to detect the color of the product and the Arduino Nano microcontroller was used to control the overall process. The identification of the color is based on the frequency analysis of the output of TCS230 sensor. Two sliders were used, each controlled by separate Servo motors. The first slider is for placing the product to be analyzed by the color sensor, and the second slider is for moving the container, having separated compartments, in order to separate the products. The experimental results promise that the prototype will fulfill the needs for higher production and precise quality in the field of automation.

First we identify the color of the object using TCS230 color sensor.

Then sort the objects based on their colors using Arduino Nano microcontroller and Servo motors.

**CHAPTER 1**

**INTRODUCTION**

Machines can perform highly repetitive tasks better than humans. Worker fatigue on assembly lines can result in reduced performance, and cause challenges in maintaining product quality. An employee who has been performing an inspection task over and over again may eventually fail to recognize the color of product. Automating many of the tasks in the industries may help to improve the efficiency of manufacturing system. The purpose of this model is to design and implement a system which automatically separates products based on their color. This machine consists of three parts: sliders, color sensor, and servo motor. The output and input of these parts was interfaced using Arduino Nano microcontroller.

To reduce human efforts on mechanical maneuvering different types of sorting machines are being developed. These machines are too costly due to the complexity in the fabrication process. A common requirement in the field of color sorting is that of color sensing and identification.

## Color Sensing and Identification

Color sensor systems are increasingly being used in automated applications to detect automation errors and monitor quality at the speed of production line. They are used in assembly lines to identify and classify products by color. The objectives of their usage include to check the quality of products, to facilitate sorting and packaging, to assess the equality of products in storage, and to monitor waste products. Consequently, there is an abundance of color sensors and the choice is often application-driven. Low cost and simple color sensors are preferred over sophisticated solutions for less demanding applications where the top priority is cost and power consumption.

Color names can be used and conjure reasonably consistent perceptions. There have eleven basic color names that have been identified such as white, gray, black, red, yellow, green, blue, orange, purple, pink, and brown. Most or all colors can be described in terms of variations and combinations of these colors. Due to the fact that human color vision is accomplished in part by three different types of cone cells in the retina, it follows that three values are necessary and sufficient to define any color. Color theory describes that there are three values that can be thought of as coordinates of a point in three-dimensional space, giving rise to the concept of color space. Hue, saturation, luminance is one such color co-ordinate system, or color space.

## Color Sorting

Bickman, et al described in the article about automated color-sorting using optical technology that has evolved from early designs intended to remove ceramic contaminants. The system configuration is similar to automated ceramic removal equipment, but color-sorting equipment used a different light source. Automated systems can generally be instructed to remove any one or a combination of the three glass colors. Industrial applications require some sort of automated visual processing and classification of items placed on a moving conveyor. Bozma and Yal-cin state that items may be randomly positioned and oriented while moving on a slider. A camera located above the conveyor views the items orthographically. Boukouvalas et al describes an integrated system developed for the detection of defects on color ceramic tiles and for the color grading of defect-free tiles. The integrated system developed under the ASSIST project (automatic system for surface inspection and sorting of tiles) is used for the detection of defects on color tiles and for the color grading of defect-free tiles. Many have proposed advanced solutions for the sorting of recyclable packaging towards process automation. Mattone et al had explained about a technique for detecting and classifying objects. Most of the authors prefer to use 2D Vision techniques to separate the objects from the known belt background and to get some of their geometrical parameters.

.

**CHAPTER 2**

**BLOCK DIAGRAM**

Microcontroller

ARDUINO NANO

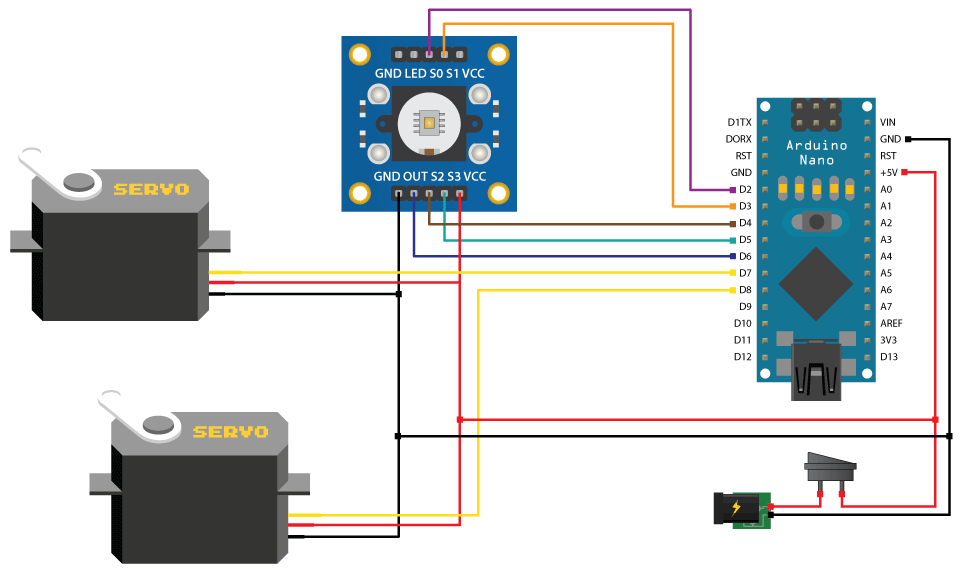
Color Sensor

TCS 230/3200

Objects are sorted based on their color

2 SERVO MOTORS and Sliders

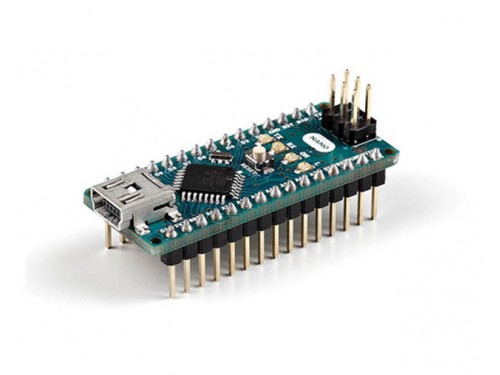
**CIRCUIT DIAGRAM**



**CHAPTER 3**

**COMPONENTS**

**ARDUINO NANO**



The Arduino Nano is a small, complete, and breadboard-friendly board based on the ATmega328P (Arduino Nano 3.x). It has more or less the same functionality of the Arduino Duemilanove, but in a different package. It lacks only a DC power jack, and works with a Mini-B USB cable instead of a standard one.

|  |  |
| --- | --- |
| Microcontroller | ATmega328 |
| Architecture | AVR |
| Operating Voltage | 5 V |
| Flash Memory | 32 KB of which 2 KB used by bootloader |
| SRAM | 2 KB |
| Clock Speed | 16 MHz |
| Analog IN Pins | 8 |
| EEPROM | 1 KB |
| DC Current per I/O Pins | 40 mA (I/O Pins) |
| Input Voltage | 7-12 V |
| Digital I/O Pins | 22 (6 of which are PWM) |
| PWM Output | 6 |
| Power Consumption | 19 mA |
| PCB Size | 18 x 45 mm |
| Weight | 7 g |
| Product Code | A000005 |

### **Power**

The Arduino Nano can be powered via the Mini-B USB connection, 6-20V unregulated external power supply (pin 30), or 5V regulated external power supply (pin 27). The power source is automatically selected to the highest voltage source.

### **Memory**

The ATmega328P has 32 KB, (also with 2 KB used for the bootloader. The ATmega328P has 2 KB of SRAM and 1 KB of EEPROM.

### **Input and Output**

Each of the 14 digital pins on the Nano can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

* Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the FTDI USB-to-TTL Serial chip.
* External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
* PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.
* SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language.
* LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Nano has 8 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the analogReference() function. Analog pins 6 and 7 cannot be used as digital pins. Additionally, some pins have specialized functionality:

* I2C: 4 (SDA) and 5 (SCL). Support I2C (TWI) communication using the Wire library (documentation on the Wiring website).

There are a couple of other pins on the board:

* AREF. Reference voltage for the analog inputs. Used with analogReference().
* Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

### **Communication**

The Arduino Nano has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328P provide UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An FTDI FT232RL on the board channels this serial communication over USB and the FTDI drivers (included with the Arduino software) provide a virtual com port to software on the computer. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the FTDI chip and USB connection to the computer (but not for serial communication on pins 0 and 1). A SoftwareSerial library allows for serial communication on any of the Nano's digital pins. The ATmega328P also support I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus. To use the SPI communication, please see ATmega328P datasheet.

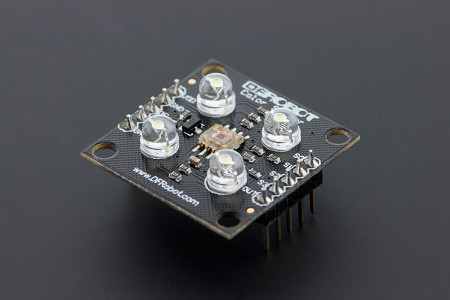
### **Programming**

The Arduino Nano can be programmed with the Arduino software ([download](http://www.arduino.org/software)). Select "Arduino Duemilanove or Nano w/ ATmega328P" from the Tools > Board menu (according to the microcontroller on your board). The ATmega328P on the Arduino Nano comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol. You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header using Arduino ISP or similar.

### **Automatic (Software) Reset**

Rather then requiring a physical press of the reset button before an upload, the Arduino Nano is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the FT232RL is connected to the reset line of the ATmega328P via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload. This setup has other implications. When the Nano is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Nano. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

**COLOR SENSOR**



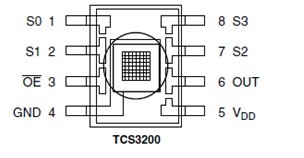
T[**CS3200 GBB Color Sensor For Arduino**](https://www.dfrobot.com/product-540.html) is a complete color detector, including a TAOS TCS3200 RGB sensor chip and 4 white LEDs. The TCS3200 can detect and measure a nearly limitless range of visible colors. Applications include test strip reading, sorting by color, ambient light sensing and calibration, and color matching, to name just a few.

The TCS3200 GBB Color Sensor For Arduino has an array of photo detectors, each with either a red, green, or blue filter, or no filter (clear). The filters of each color are distributed evenly throughout the array to eliminate location bias among the colors. Internal to the device is an oscillator which produces a square-wave output whose frequency is proportional to the intensity of the chosen color.

## Specifications

* Single-Supply Operation (2.7V to 5.5V)
* High-Resolution Conversion of Light Intensity to Frequency
* Programmable Color and Full-Scale Output Frequency
* Power Down Feature
* Communicates Directly to Microcontroller
* S0~S1: Output frequency scaling selection inputs
* S2~S3: Photodiode type selection inputs
* OUT Pin: Output frequency
* OE Pin: Output frequency enable pin (active low), can be impending when using
* Support LED lamp light supplement control
* Size: 28.4x28.4mm

## PinOut

[](https://www.dfrobot.com/wiki/index.php/File:Color_Sensor_1.jpg)

|  |  |  |
| --- | --- | --- |
| **Pin Name** | **I/O** | **DESCRIPTION** |
| GND(4) |  | Power supply ground. All voltages are referenced to GND |
| OE(3) | I | Enable for fo (active low). |
| OUT | O | Output frequency (fo). |
| S0,S1（1，2） | I | Output frequency scaling selection inputs. |
| S2,S3（7，8） | I | Photodiode type selection inputs |
| VDD（5） |  | Supply voltage |

|  |  |
| --- | --- |
| **Wiring instructions** | |
| VCC——5V | GND——GND |
| S0——D3 | S1——D4 |
| S2——D5 | S3——D6 |
| OUT——D2 |  |

### **S0,S1,S2,S3**

To TCS3002D, when choose a color filter, it can allow only one particular color to get through and prevent other color. For example, when choose the red filter, Only red incident light can get through, blue and green will be prevented. So we can get the red light intensity. Similarly ,when choose other filters we can get blue or green light.

TCS3002D has four photodiode types. Red , blue, green and clear, reducing the amplitude of the incident light uniformity greatly, so that to increase the accuracy and simplify the optical. When the light project to the TCS3002D we can choose the different type of photodiode by different combinations of S2 and S3. Look at the form as follows.

|  |  |  |
| --- | --- | --- |
| **S0** | **S1** | **OUTPUT FREQUENCY SCALING (fo)** |
| L | L | Power down |
| L | H | 2% |
| H | L | 20% |
| H | H | 100% |

TCS3002D can output the frequency of different square wave (occupies emptiescompared 50%),different color and light intensity correspond with different frequency of square wave. There is a relationship between the output and light intensity. The range of the typical output frequency is 2HZ~500KHZ. We can get different scaling factor by different combinations of S0 and S1. Look at the form as follows.

|  |  |  |
| --- | --- | --- |
| **S2** | **S3** | **PHOTODIODE TYPE** |
| L | L | RED |
| L | H | BLUE |
| H | L | Clear (no filter) |
| H | H | GREEN |

**SERVO MOTOR**



A **servomotor** is a [rotary actuator](https://en.wikipedia.org/wiki/Rotary_actuator) or [linear actuator](https://en.wikipedia.org/wiki/Linear_actuator) that allows for precise control of angular or linear position, velocity and acceleration.[[1]](https://en.wikipedia.org/wiki/Servomotor#cite_note-1) It consists of a suitable motor coupled to a sensor for position feedback. It also requires a relatively sophisticated controller, often a dedicated module designed specifically for use with servomotors.

Servomotors are not a specific class of motor although the term *servomotor* is often used to refer to a motor suitable for use in a [closed-loop control](https://en.wikipedia.org/wiki/Closed-loop_control) system.

Servo implies an error sensing feedback control which is utilized to correct the performance of a system. It also requires a generally sophisticated controller, often a dedicated module designed particularly for use with servomotors. Servo motors are DC motors that allows for precise control of angular position. They are actually DC motors whose speed is slowly lowered by the gears. The servo motors usually have a revolution cutoff from 90° to 180°. A few servo motors also have revolution cutoff of 360° or more. But servo motors do not rotate constantly. Their rotation is limited in between the fixed angles.

The servo motor is actually an assembly of four things: a normal DC motor, a gear reduction unit, a position-sensing device and a control circuit. [The DC motor](http://www.edgefxkits.com/four-quadrant-dc-motor-control-without-microcontroller) is connected with a gear mechanism which provides feedback to a position sensor which is mostly a potentiometer. From the gear box, the output of the motor is delivered via servo spline to the servo arm. For standard servo motors, the gear is normally made up of plastic whereas for high power servos, the gear is made up of metal.

A servo motor consists of three wires- a black wire connected to ground, a white/yellow wire connected to control unit and a red wire connected to power supply.

The function of the servo motor is to receive a control signal that represents a desired output position of the servo shaft and apply power to its DC motor until its shaft turns to that position.

It uses the position sensing device to figure out the rotational position of the shaft, so it knows which way the motor must turn to move the shaft to the instructed position. The shaft commonly does not rotate freely around similar to a DC motor, however rather can just turn 200 degrees.

From the position of the rotor, a rotating magnetic field is created to efficiently generate toque. Current flows in the winding to create a rotating magnetic field. The shaft transmits the motor output power. The load is driven through the transfer mechanism. A high-function rare earth or other permanent magnet is positioned externally to the shaft. The optical encoder always watches the number of rotations and the position of the shaft.

### Working of a Servo Motor

The Servo Motor basically consists of a DC Motor, a Gear system, a position sensor and a control circuit. [The DC motors get powered from a battery and run at high speed and low torque.](http://www.edgefxkits.com/four-quadrant-dc-motor-speed-control-with-microcontroller)The Gear and shaft assembly connected to the DC motors lower this speed into sufficient speed and higher torque. The position sensor senses the position of the shaft from its definite position and feeds the information to the control circuit. The control circuit accordingly decodes the signals from the position sensor and compares the actual position of the motors with the desired position and accordingly controls the direction of rotation of the DC motor to get the required position. The Servo Motor generally requires DC supply of 4.8V to 6 V.

### Controlling a Servo Motor

A servo motor is controlled by controlling its position using Pulse Width Modulation Technique. The width of the pulse applied to the motor is varied and send for a fixed amount of time.

The pulse width determines the angular position of the servo motor. For example a pulse width of 1 ms causes a angular position of 0 degrees, whereas a pulse width of 2 ms causes a angular width of 180 degrees.

**TO INTERFACE SERVO MOTOR**

## Servo library

|  |
| --- |
| This library allows an Arduino board to control RC (hobby) servo motors. Servos have integrated gears and a shaft that can be precisely controlled. Standard servos allow the shaft to be positioned at various angles, usually between 0 and 180 degrees. Continuous rotation servos allow the rotation of the shaft to be set to various speeds.  The Servo library supports up to 12 motors on most Arduino boards and 48 on the Arduino Mega. On boards other than the Mega, use of the library disables analogWrite() (PWM) functionality on pins 9 and 10, whether or not there is a Servo on those pins. On the Mega, up to 12 servos can be used without interfering with PWM functionality; use of 12 to 23 motors will disable PWM on pins 11 and 12. |
| This library allows an Arduino board to control RC (hobby) servo motors. Servos have integrated gears and a shaft that can be precisely controlled. Standard servos allow the shaft to be positioned at various angles, usually between 0 and 180 degrees. Continuous rotation servos allow the rotation of the shaft to be set to various speeds.  The Servo library supports up to 12 motors on most Arduino boards and 48 on the Arduino Mega. On boards other than the Mega, use of the library disables analogWrite() (PWM) functionality on pins 9 and 10, whether or not there is a Servo on those pins. On the Mega, up to 12 servos can be used without interfering with PWM functionality; use of 12 to 23 motors will disable PWM on pins 11 and 12.  Circuit  Servo motors have three wires: power, ground, and signal. The power wire is typically red, and should be connected to the 5V pin on the Arduino board. The ground wire is typically black or brown and should be connected to a ground pin on the Arduino board. The signal pin is typically yellow, orange or white and should be connected to a digital pin on the Arduino board. Note that servos draw considerable power, so if you need to drive more than one or two, you'll probably need to power them from a separate supply (i.e. not the +5V pin on your Arduino). Be sure to connect the grounds of the Arduino and external power supply together. |

**CHAPTER 4**

**SOFTWARE REQUIREMENTS**

**ARDUINO SOFTWARE**

A program for Arduino hardware may be written in any [programming language](https://en.wikipedia.org/wiki/Programming_language) with compilers that produce binary machine code for the target processor. Atmel provides a development environment for their [AVR](https://en.wikipedia.org/wiki/Atmel_AVR) and [ARM Cortex-M](https://en.wikipedia.org/wiki/ARM_Cortex-M) based microcontrollers: AVR Studio (older) and Atmel Studio (newer).

### IDE

The Arduino [integrated development environment](https://en.wikipedia.org/wiki/Integrated_development_environment) (IDE) is a [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) application (for [Windows](https://en.wikipedia.org/wiki/Windows), [macOS](https://en.wikipedia.org/wiki/MacOS), [Linux](https://en.wikipedia.org/wiki/Linux)) that is written in the programming language [Java](https://en.wikipedia.org/wiki/Java_(programming_language)). It originated from the IDE for the languages [*Processing*](https://en.wikipedia.org/wiki/Processing_(programming_language)) and [*Wiring*](https://en.wikipedia.org/wiki/Wiring_(development_platform)). It includes a code editor with features such as text cutting and pasting, searching and replacing text, automatic indenting, [brace matching](https://en.wikipedia.org/wiki/Brace_matching), and [syntax highlighting](https://en.wikipedia.org/wiki/Syntax_highlighting), and provides simple *one-click* mechanisms to compile and upload programs to an Arduino board. It also contains a message area, a text console, a toolbar with buttons for common functions and a hierarchy of operation menus. The source code for the IDE is released under the [GNU General Public License](https://en.wikipedia.org/wiki/GNU_General_Public_License), version 2.

The Arduino IDE supports the languages [C](https://en.wikipedia.org/wiki/C_(programming_language)) and [C++](https://en.wikipedia.org/wiki/C%2B%2B) using special rules of code structuring. The Arduino IDE supplies a [software library](https://en.wikipedia.org/wiki/Software_library) from the [Wiring](https://en.wikipedia.org/wiki/Wiring_(development_platform)) project, which provides many common input and output procedures. User-written code only requires two basic functions, for starting the sketch and the main program loop, that are compiled and linked with a program stub *main()* into an executable [cyclic executive](https://en.wikipedia.org/wiki/Cyclic_executive)program with the [GNU toolchain](https://en.wikipedia.org/wiki/GNU_toolchain), also included with the IDE distribution. The Arduino IDE employs the program *avrdude* to convert the executable code into a text file in hexadecimal encoding that is loaded into the Arduino board by a loader program in the board's firmware.

### Sketch**[**[**edit**](https://en.wikipedia.org/w/index.php?title=Arduino&action=edit&section=8)**]**

A program written with the IDE for Arduino is called a *sketch*. Sketches are saved on the development computer as text files with the file extension *.ino*. Arduino Software (IDE) pre-1.0 saved sketches with the extension *.pde*.

A minimal Arduino C/C++ program consist of only two functions:

* *setup()*: This function is called once when a sketch starts after power-up or reset. It is used to initialize variables, input and output pin modes, and other libraries needed in the sketch.
* *loop()*: After *setup()* has been called, function *loop()* is executed repeatedly in the main program. It controls the board until the board is powered off or is reset.
* /\* Arduino Project - Color Sorting Machine
* #include <Servo.h>
* #define S0 2
* #define S1 3
* #define S2 4
* #define S3 5
* #define sensorOut 6
* Servo topServo;
* Servo bottomServo;
* **int** frequency = 0;
* **int** color=0;
* **void** setup() {
* pinMode(S0, OUTPUT);
* pinMode(S1, OUTPUT);
* pinMode(S2, OUTPUT);
* pinMode(S3, OUTPUT);
* pinMode(sensorOut, INPUT);
* // Setting frequency-scaling to 20%
* digitalWrite(S0, HIGH);
* digitalWrite(S1, LOW);
* topServo.attach(7);
* bottomServo.attach(8);
* Serial.begin(9600);
* }
* **void** loop() {
* topServo.write(115);
* delay(500);
* for(**int** i = 115; i > 65; i--) {
* topServo.write(i);
* delay(2);
* }
* delay(500);
* color = readColor();
* delay(10);
* **switch** (color) {
* **case** 1:
* bottomServo.write(50);
* **break**;
* **case** 2:
* bottomServo.write(75);
* **break**;
* **case** 3:
* bottomServo.write(100);
* **break**;
* **case** 4:
* bottomServo.write(125);
* **break**;
* **case** 5:
* bottomServo.write(150);
* **break**;
* **case** 6:
* bottomServo.write(175);
* **break**;
* **case** 0:
* **break**;
* }
* delay(300);
* for(**int** i = 65; i > 29; i--) {
* topServo.write(i);
* delay(2);
* }
* delay(200);
* **for**(**int** i = 29; i < 115; i++) {
* topServo.write(i);
* delay(2);
* }
* color=0;
* }
* // Custom Function - readColor()
* **int** readColor() {
* // Setting red filtered photodiodes to be read
* digitalWrite(S2, LOW);
* digitalWrite(S3, LOW);
* // Reading the output frequency
* frequency = pulseIn(sensorOut, LOW);
* **int** R = frequency;
* // Printing the value on the serial monitor
* Serial.print("R= ");//printing name
* Serial.print(frequency);//printing RED color frequency
* Serial.print(" ");
* delay(50);
* // 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);
* if(R<45 & R>32 & G<65 & G>55){
* color = 1; // Red
* }
* **if**(G<55 & G>43 & B<47 &B>35){
* color = 2; // Orange
* }
* if(R<53 & R>40 & G<53 & G>40){
* color = 3; // Green
* }
* **if**(R<38 & R>24 & G<44 & G>30){
* color = 4; // Yellow
* }
* **if**(R<56 & R>46 & G<65 & G>55){
* color = 5; // Brown
* }
* **if** (G<58 & G>45 & B<40 &B>26){
* color = 6; // Blue
* }
* **return** color;

So, we need to include the “Servo.h” library, define the pins to which the color sensor will be connected, create the servo objects and declare some variables needed for the program. In the setup section we need to define the pins as Outputs and Inputs, set the frequency-scaling for the color sensor, define the servo pins and start the serial communication for printing the results of the color read on the serial monitor.

In the loop section, our program starts with moving the top servo motor to the position of the skittle charger. Note that this value of 115 suits to my parts and my servo motor, so you should adjust this value as well as the following values for the servo motors according to your build.

Next using the “for” loop we will rotate and bring the skittle to the position of the color sensor. We are using a “for” loop so that we can control the speed of the rotation by changing the delay time in loop.

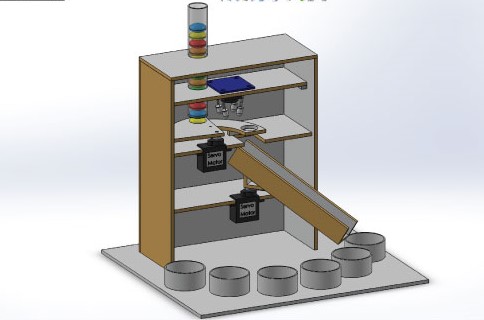
Next, after half a second delay, using the custom made function, readColor() we will read the color of the skittle. Here’s the code of the custom function. Using the four control pins and the frequency output pin of the color sensor we read color of the skittle. The sensor reads 3 different values for each skittle, Red, Green and Blue and according to these values we tell what the actual color is. For more details how the TCS3200 color sensor works you can check my previous detailed tutorial about it.

Here are the RGB values that I got from the sensor for each skittle. Note that these values can vary because the sensors isn’t always accurate. Therefore, using these “if” statements we allow the sensor an error of around +-5 of the tested value for the particular color. So for example if we have a Red skittle, the first “if” statement will be true and the variable “color” will get the value 1. So that’s what the readColor() custom function does and after that using a “switch-case” statement we rotate the bottom servo to the particular position. At the end we further rotate the top servo motor until the skittle drops into the guide rail and again send it back to the initial position so that the process can repeated.

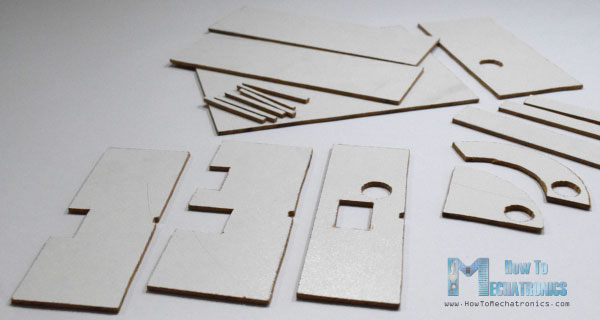
**CHAPTER 5**

**WORKING PRINCIPLE**

All we need for this [Arduino project](https://howtomechatronics.com/arduino-projects/) is one color sensor (TCS3200) and two hobbyist servo motors, which makes this project quite simple but yet very fun to build it. In the first place, using the Solidworks 3D modeling software I made the design of the color sorter and here’s its working principle:



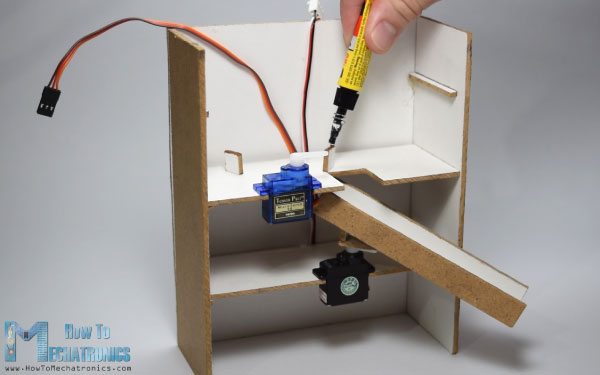
* Initially, the colored skittles which are held in the charger drop into the platform attached on the top servo motor.
* Then the servo motor rotates and brings the skittle to the color sensor which detects its color.
* After that the bottom servo motor rotates to the particular position and then the top servo motor rotates again till the skittle drop into the guide rail.
* The material that I used for this project is a 3 mm tick fiberboard. I redraw the parts on the fiberboard according to the drawings and using a small hand saw cut all the parts to size.



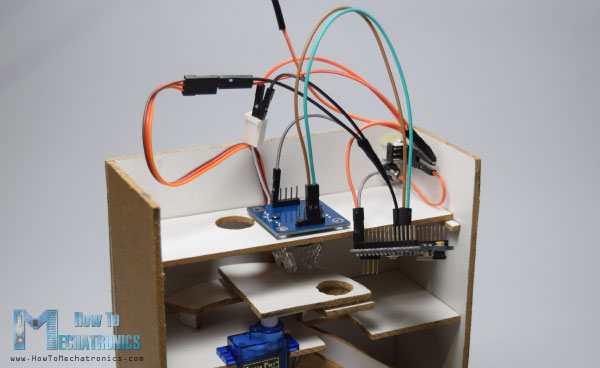
* Once I got all the parts ready, I started assembling them. First I assembled the outer parts using a glue gun.



* Then using all-purpose glue I glued the two servo motors on their platforms and attached them to the assembly.
* After that again using a glue I attached the guide rail on the bottom servo motor as well as the support and the platform needed for the top servo motor.



* Next, I inserted a switch and a power jack for powering the Arduino with a 5V adapter and on the third platform I inserted the color sensor.



**CHAPTER 6**

**RESULTS**

We have developed a sorting machine using Arduino Nano for automatic color sorting, taking in to consideration three colors namely Green, Red and Black. We consumed two months to produce the prototype. This figure shows different stages involved in the process. You may note that the green object and the red object lying in different sections of the container placed on the second conveyor belt.

**Expected result**

****

**CHAPTER 7**

ADVANTAGES:

Automatic sorting of objects based on the color can be done by using this.

Continuous manual sorting creates consistency issues. We can overcome this using this model.

DISADVANTAGES:

Color sensor doesn’t work properly.

.

FUTURE ENHANCEMENTS:

• We can add a load cell for measurement and control of weight of the product

• We can also add a counter for counting the number of products

• Speed of the system can be increased accounting to the speed of production

• The system can be used as a quality controller by adding more sensors

• The sensor can be changed according to the type of product

**CHAPTER 8**

**CONCLUSION**

We have developed a working model of automated Color Sorting machine working model using Arduino NANO as microcontroller, TCS 230 color sensor, two Servo motors, Cardboard. This is a working model of Color sorting machine we can further develop this for future enhancements. It can be used in many of the industry applications. Finally we had completed the working model.

**References:**

[**https://howtomechatronics.com/projects/arduino-color-sorter-project/**](https://howtomechatronics.com/projects/arduino-color-sorter-project/)