VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELGAVI-590014

A Project Report
On

"Color Sorting Machine"

Submitted in partial fulfillment of the requirements for the 8th semester of ***Bachelor of Engineering in Electrical and Electronics Engineering*** *of Visvesvaraya Technological University, Belagavi*

Submitted by:

RAGHAV AMBASHTA	**1RN18EE030**
SHALU PRIYAMVADA	**1RN18EE035**
ANUBHAV AYUSHYAMAN	**1RN18EE006**
ATULYA MODI	**1RN18EE008**

Under the Guidance of:
Dr. Sumathi S
Professor & HOD
Dept. of EEE

Department of Electrical and Electronics Engineering

RNS Institute of Technology

Channasandra, Dr. Vishnuvardhan Road, RR Nagar Post, Bengaluru – 560 098

2021-2022

RNS Institute of Technology

Channasandra, Uttarahalli - Kengeri Main Road,
Bengaluru - 560 098

Department of Electrical and Electronics Engineering

CERTIFICATE

Certified that the mini project work entitled **"Color Sorting Machine"** has been successfully carried out by **Raghav Ambashta (1RN18EE030), Shalu Priyamvada** (**1RN18EE035)**, **Anubhav Ayushyaman** (**1RN18EE006)** and **Atulya Modi** (**1RN18EE008)** bonafide students of **RNS Institute of Technology** in partial fulfillment of the requirements for the **8th semester** of **Bachelor of Engineering** in **Electrical and Electronics Engineering** of **Visvesvaraya Technological University**, Belagavi, during the academic year 2021-2022. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the requirements of **8th semester BE, EEE**.

Signature of Guide	**Signature of HoD** **(Dept. of EEE)**	**Signature of Principal**
Dr. Sumathi S	Dr. Sumathi S	Dr. M K Venkatesha

RNS Institute of Technology
Channasandra, Uttarahalli - Kengeri Main Road,
Bengaluru - 560 098

Department of Electrical and Electronics Engineering

DECLARATION

We hereby declare that the entire work embodied in this project report titles “Color Sorting Machine” submitted to Visvesvaraya Technological University, Belagavi, is carried out by us at the Department of Electrical and Electronics Engineering, RNS Institute of Technology, Bengaluru under the guidance of Dr. Sumathi S, Professor and HoD, Electrical and Electronics Engineering, RNS Institute OF Technology. This project has not been submitted in part of full for thew award of any diploma or degree of this or any other University.

Name	USN	Signature
RAGHAV AMBASHTA	1RN18EE030	
SHALU PRIYAMVADA	1RN18EE035	
ANUBHAV AYUSHYAMAN	1RN18EE006	
ATULYA MODI	1RN18EE008	

ACKNOWLEDGEMENT

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped us in carrying out this project work.

We would like to take this opportunity to thank them all. We would firstly like to thank **Visvesvaraya Technological University** for giving us the opportunity to work on a technical project.

We are grateful to **Dr. M K Venkatesha,** Principal, RNSIT, Bangalore, for his support towards completing our project.

We deeply express our sincere gratitude to our guides **Dr. Sumathi S, Professor & HOD, Department of EEE, RNSIT, Bangalore** for the able guidance, regular source of encouragement and assistance throughout this project.

We would like to thank all the teaching and non-teaching staff of Department of Electrical and Electronics Engineering RNSIT, Bangalore for their constant support and encouragement.

Date: 21/06/2022	**Raghav Ambashta**	**1RN18EE030**
Place: Bengaluru	**Shalu Priyamvada**	**1RN18EE035**
	Anubhav Ayushyaman	**1RN18EE006**
	Atulya Modi	**1RN18EE008**

ABSTRACT

In the modern scenario of manufacturing, quality holds an important component for achievement. It is essential to increase manufacturing pace, lower the labor charge and perform the task with utmost precision and efficiency. Sorting of object is an essential process in manufacturing. Merchandise should be taken care of in manufacturing and manual sorting is time consuming and labor extensive.

Color sorting machines are most commonly used in the sorting of agricultural grain and rice, as well as in the processing of food products, such as coffee, nuts and oil crops. The optical sorter separates any stones, mouse droppings and discolored, toxic or otherwise unacceptable items. Manual work can create consistency problems. Machines can perform assignments superior to human beings.

This project aims at making an automatic sorting tool which helps the sorting mechanism based on color. For sensing TCS3200 coloration sensor has been used. For actuation, micro servo motors have been used. With the aid of reading the frequency of the output of the sensor, color primarily based absolutely sorting is completed.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCION

Problem Statement:

"Arduino Based Color Sorting Machine using TCS3200 colour sensor"

Sorting of objects based on color is an essential mechanical process in the manufacturing industry. Manually doing this task consumes a lot of time, money and energy. Product consistency and quality is also very poor. To obtain high quality product with low cost and time investment, a machine which can automate this task is the need of the day. This project aims at solving this problem by making an Arduino based color sorting machine using the TCS3200 color sensor. The implementation makes use of micro servo motors for actuation, color sensor for determining color and microcontroller for logic-based control of all the components. For the construction of mechanical structure, sturdy whiteboards will be used. The application of this machine can be found in lots of manufacturing industries, especially the food processing industry. Rice, coffee, grains and vegetables are the best suited products for sorting based on color. By determining the color of grains, the freshness and quality of product can be obtained. The defective items can then be removed.

CHAPTER 2

WORKING PRINCIPLE

2.1 WORKING PRINCIPLE

This project works on the principle that every color is composed of 3 fundamental colors- red, green and blue. By determining the values of these 3 components, colors can be differentiated from one another. For this task the TCS3200 color sensor has been used. It is basically a light to frequency converter that converts the incident light on its photodiode array into frequencies of RGB. Using a pre-programmed logic in the microcontroller, these values are then compared with pre-defined values to determine the color. Using micro servo-motors, the objects are then placed in respective containers.

The process of sorting begins at the hopper where the objects are placed. A mechanical arm controls the dropping of objects and then guides them along a guiding rail to the color sensor platform. Here, the color determines the RGB frequencies of object and sends this information to the microcontroller. The microcontroller then determines the color of the object and then actuated the bottom servo motor to place it in the respective container.

The whole system can be simplified into 4 units:

2.1.1 Input Unit

The essential employment of this unit is to transport the object to sensor unit, while the object goes to the sensor unit the manual rail desires to prevent. The hopper and top servomotor form this unit.

2.1.2 Processing Unit

This unit consists of the TCS3200 color sensor which determines the color of input object. It passes this information to the control unit i.e the Arduino Uno board. It is fed by the input unit and passes the object to output unit.

2.1.3 Output Unit

This unit consists of bottom servomotor and the sorted boxes of items. Based on color of the object in processing unit the bottom servomotor moves in required direction facing towards the respective box containing sorted objects. The object then falls into it and the process is complete.

2.1.4 Control Unit

This unit consists of Arduino Uno board which is the heart of the system. It controls all the above units.

2.2 SCHEMATIC DIAGRAM

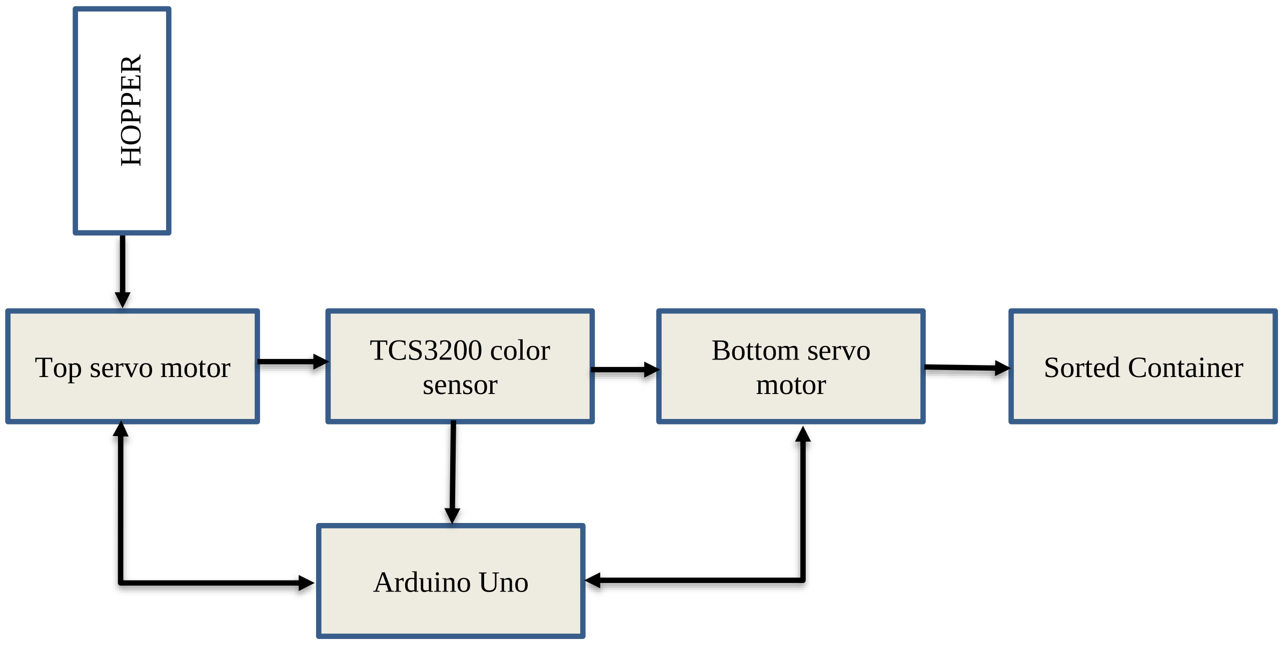


Fig 2.1 Basic Block Diagram

2.2.1 Hopper

It is a long cylindrical plastic tube which contains the objects to be sorted.

2.2.2 Top servo motor

It is a SG90 micro servomotor which actuates the which controls dropping of object from hopper and takes it to the color sensing platform.

2.2.3 TCS3200 color sensor

It determines the color of the object and passes this information to the microcontroller.

2.2.4 Bottom servo motor

Another SG90 micro servo motor that guides the object to its respective sorted container.

2.2.5 Sorted container

They contain objects of every particular color

2.2.6 Arduino Uno

It is a microcontroller board which contains the program of machine and controls all the components of the machine.

2.3 CIRCUIT DIAGRAM

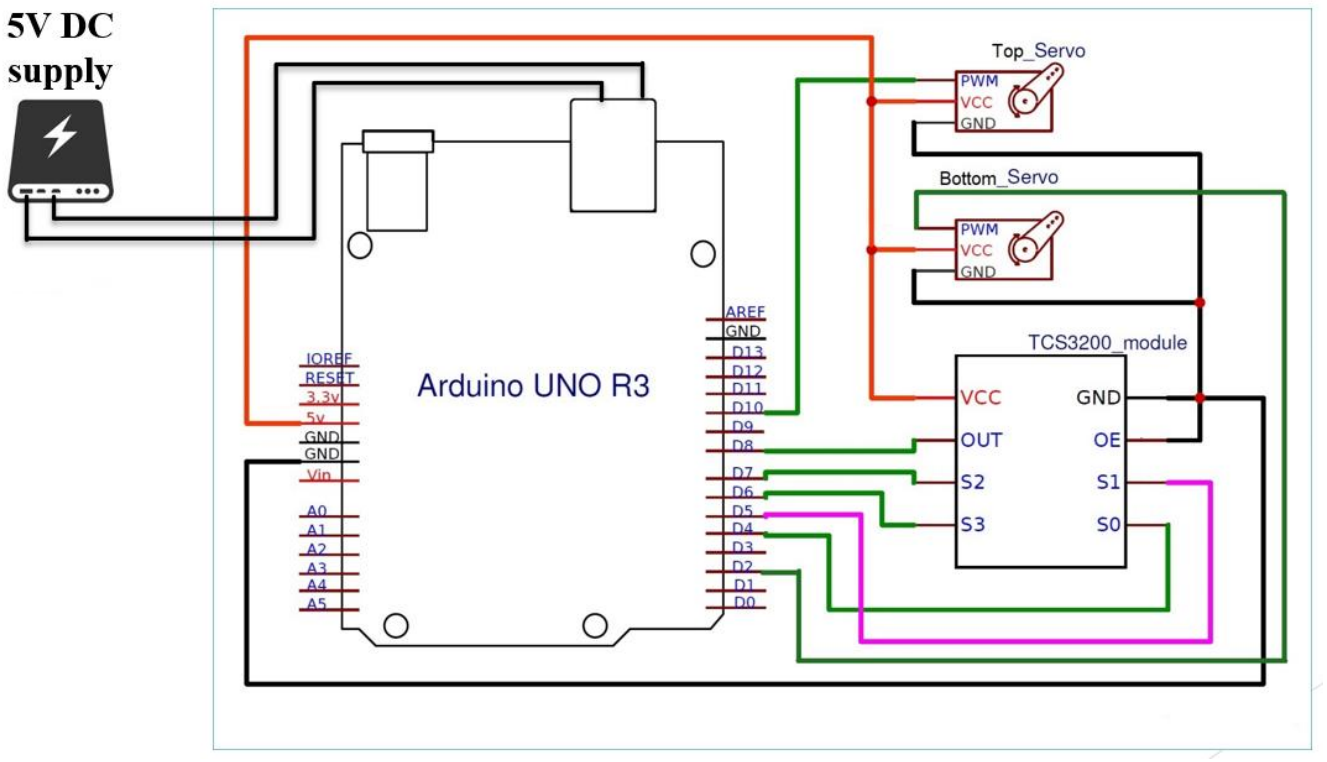


Fig 2.2 Circuit Diagram

CHAPTER 3

COMPONENT LIST

3.1 ARDUINO UNO

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. A worldwide community of makers - students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount of accessible knowledge that can be of great help to novices and experts alike.

Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. As soon as it reached a wider community, the Arduino board started changing to adapt to new needs and challenges, differentiating its offer from simple 8-bit boards to products for IoT applications, wearable, 3D printing, and embedded environments. All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The software, too, is open-source, and it is growing through the contributions of users worldwide

its simple and accessible user experience, Arduino has been used in thousands of different projects and applications. The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux. Teachers and students use it to build low cost scientific instruments, to prove chemistry and physics principles, or to get started with programming and robotics. Designers and architects build interactive prototypes, musicians and artists use it for installations and to experiment with new musical instruments. Makers, of course, use it to build many of the projects exhibited at the Maker Faire, for example. Arduino is a key tool to learn new things. Anyone - children, hobbyists, artists, programmers - can start tinkering just following the step-by-step instructions of a kit, or sharing ideas online with other members of the Arduino community.

There are many other microcontrollers and microcontroller platforms available for physical computing. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, and many others offer similar functionality. All of these tools take the messy details of microcontroller programming and wrap it up in an easy-to-use package. Arduino also simplifies the process of working with microcontrollers, but it offers some advantage for teachers, students, and interested amateurs over other systems:

1. **Inexpensive** - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than Rs. 1000

2. **Cross-platform** - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.

3. **Simple, clear programming environment** - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with how the Arduino IDE works.

4. **Open source and extensible software** - The Arduino software is published as open-source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.

5. **Open source and extensible hardware** - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

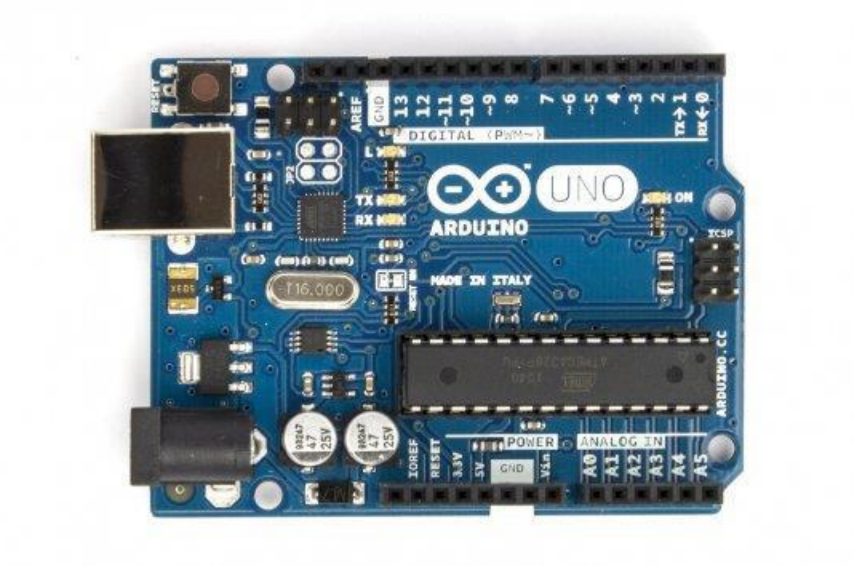


Fig 3.1 Arduino UNO

3.1.1 SPECIFICATIONS

Microcontroller:	ATmega328
Operating Voltage:	5V
Input Voltage (recommended):	7-12V
Input Voltage (limits):	6-20V
Digital I/O Pins:	14 (6 provide PWM output)
Analog Input Pins:	6
DC Current per I/O Pin:	40 mA
DC Current for 3.3V Pin:	50 mA
Flash Memory:	32 KB
SRAM:	2 KB
EEPROM:	1 KB
Clock Speed:	16 MHz

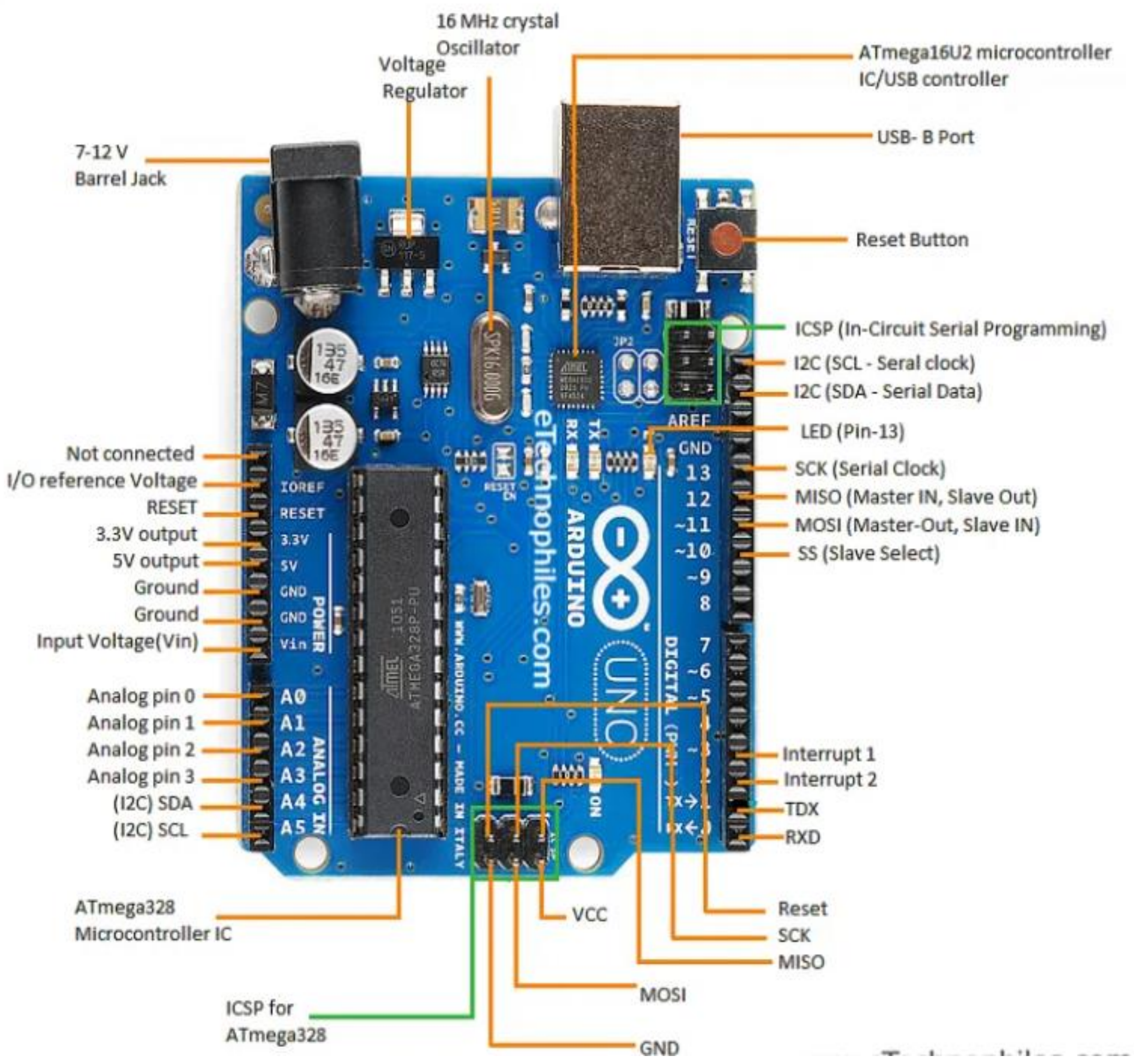


Fig 3.2 Arduino UNO board pin diagram

3.1.2 General Pin functions:

1. **LED**: There is a built-in LED driven by digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
2. **VIN**: The input voltage to the Arduino/Genuino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

3. **5V**: This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 20V), the USB connector (5V), or the VIN pin of the board (7-20V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage the board.
4. **3V3**: A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA
5. **GND**: Ground pins.
6. **IOREF**: This pin on the Arduino/Genuino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs to work with the 5V or 3.3V.
7. **Reset**: Typically used to add a reset button to shields which block the one on the board.

3.1.2 Special Pin Functions

Each of the 14 digital pins and 6 Analog pins on the Uno 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 20 mA as recommended operating condition and has an internal pull-up resistor (disconnected by default) of 20-50k ohm. A maximum of 40mA is the value that must not be exceeded on any I/O pin to avoid permanent damage to the microcontroller. The Uno has 6 analog inputs, labeled A0 through A5, 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 AREF pin and the analogReference() function.

3.1.3 In addition, some pins have specialized functions:

1. **Serial**: pins 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 ATmega8U2 USB-to-TTL Serial chip.

2. **External Interrupts**: pins 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.

3. **PWM**(**P**ulse **W**idth **M**odulation) 3, 5, 6, 9, 10, and 11 Can provide 8-bit PWM output with the analogWrite() function.

4. **SPI**(**S**erial **P**eripheral **I**nterface): 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.

5. **TWI**(**T**wo **W**ire **I**nterface): A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

6. **AREF**(Analog Reference): Reference voltage for the analog inputs.

3.1.4 Communication:

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino board, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The 16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a .inf file is required.

The Arduino Software (IDE) includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1). A SoftwareSerial library allows serial communication on any of the Uno's digital pins.

3.1.5 Automatic (Software) Reset:

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno board is designed in a way that allows it to be reset by software running on a connected to a computer. One of the hardware flow control lines (DTR) of the ATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nano farad capacitor.

When this line is asserted (taken low), the reset line drops long enough to reset the chip. This setup has other implications. When the Uno 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 Uno. 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.

3.2 TCS 3200 COLOUR SENSOR

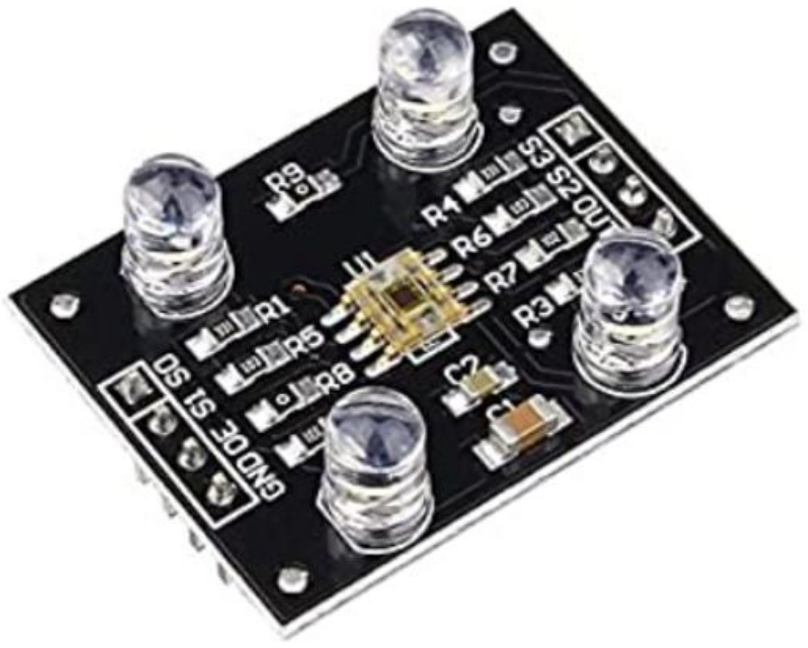


Fig 3.3 TCS3200 Colour Sensor

3.2.1 How Color Sensors Work

white light is made up of three primary colors (Red, green and blue), which have different wavelengths. These colors combine with each other to form different shades of color. When white light falls on any surface, some wavelengths of light are absorbed and some are reflected, depending on the properties of the surface material. The color we see is a result of which wavelengths are reflected back into our eyes.

Now coming back to the sensor, a typical color sensor includes a high-intensity white LED that projects a modulated light onto the object. To detect the color of reflected light, almost all the color sensors consists of a grid of color-sensitive filter, also known as ‘**Bayer Filter**‘ and an array of photodiodes underneath, as shown in the picture below.

A single pixel is made up of 4 filters, one red, one blue, one green and one clear filter (no filter). This pattern is also known as the ‘**Bayer Pattern**‘. Each filter passes light of just a single color to the photodiode beneath, while the clear filter passes light as it is, as shown below. This extra light passed through the clear filter is a major advantage in low light conditions.

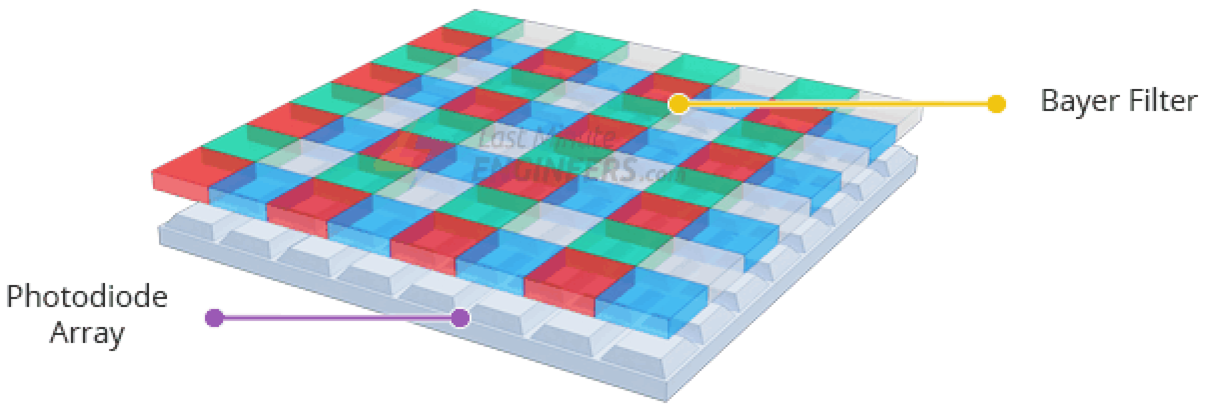


Fig 3.4 Photodiode Array

The processing chip then addresses each photodiode (one color at a time), and measures the intensity of the light. As there is an array of photodiodes, the results are first averaged and then sent out for processing. By measuring the relative level of red, green and blue light, the color of he object is determined.

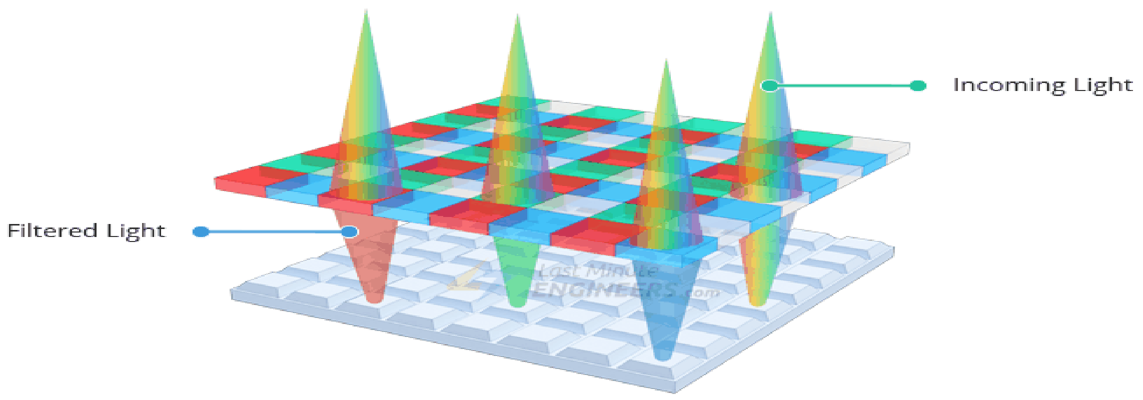


Fig 3.5 Working of Photodiode Array

3.2.2 TCS3200 Design

At the heart of the module is an inexpensive RGB sensor chip from Texas Advanced Optoelectronic Solutions – TCS230. The TCS230 Color Sensor is a complete color detector that can detect and measure an almost infinite range of visible colors. The sensor itself can be seen at the center of the module, surrounded by the four white LEDs. The LEDs light up when the module is powered up and are used to illuminate the object being sensed. Thanks to these LEDs, the sensor can also work in complete darkness to determine the color or brightness of the object. The TCS230 operates on a supply voltage of 2.7 to 5.5 volts and provides TTL logic-level outputs.

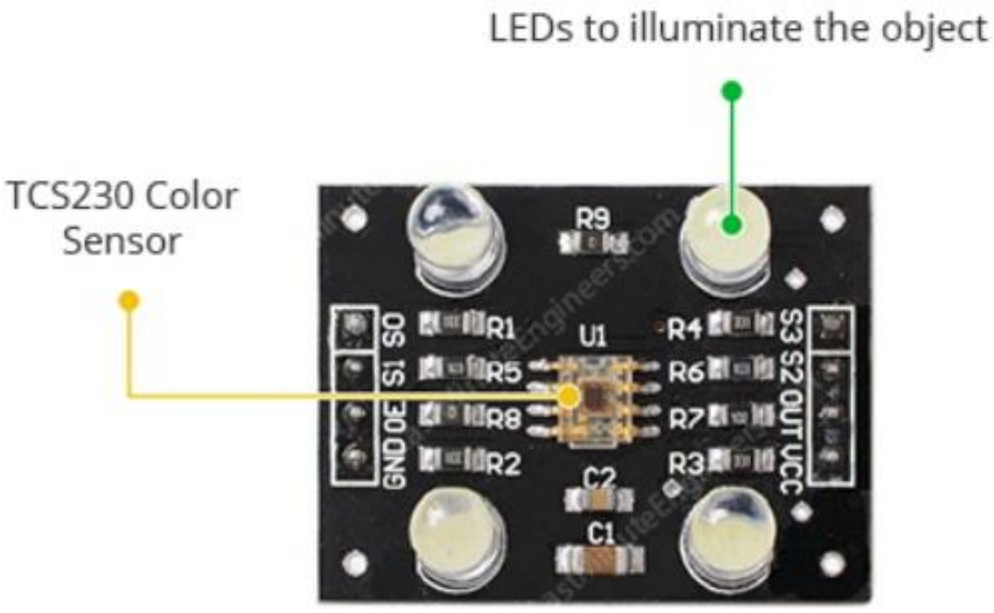


Fig 3.6 TCS3200 Design

3.2.3 Operation

The output of TCS3200 is a square wave with 50% duty cycle. TCS3200 can be interfaced with any microcontroller directly using digital input and digital output pins. Using two control input pins, the full-scale output frequency can be scaled. From the four different types of filter covered photodiodes, each diode can be activated using S2 and S3 selection inputs.

When the diodes with red filter are chosen, only red incident light is measured and the remaining green and blue light are blocked. Then by measuring the frequency, the intensity of the red light can be detected. Using S0, S1 select inputs, frequency scaling factor can be set.

To measure the frequency, the 6th pin is used. This pin is connected to the microcontroller. To determine the color of the object, first set all the input pins as input and output pins as output. Here there is no use of analog pins. Next set the desired frequency scaling by setting the pins S0, S1 either high or low.

Now to detect the color of the object, activate each type of filter using S2 and S3 selection lines. After activation of filter measure the frequency generated on the 6th pin using a microcontroller. When both S2 and S3 pins are low, Red color filters are activated and the intensity of red components of the object is detected. Similarly, driving S2 low and S3 high detects Blue component and driving both S2, S3 to High detects Green components of the object. All these values are collected and compared to get the actual color of the object. Ambient light can cause variations in the measurements so sensor and object should be protected from the ambient light during measurement.
0.01μF to 0.1μF capacitors should be used for decoupling the power supply lines. For input noise immunity a low impedance electrical connection is required at the device output and device ground. A buffer is required if lines greater than 12 inches are used at the output.

The TCS230 detects colour with the help of an 8 x 8 array of photodiodes, of which sixteen photodiodes have red filters, 16 photodiodes have green filters, 16 photodiodes have blue filters, and remaining 16 photodiodes are clear with no filters. If you look closely at the sensor, you can actually see these filters.

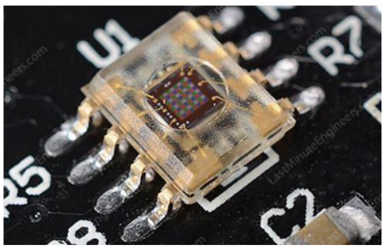
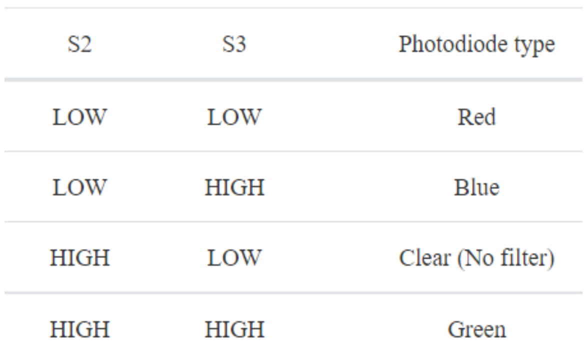


Fig 3.7 TCS3200 Filter

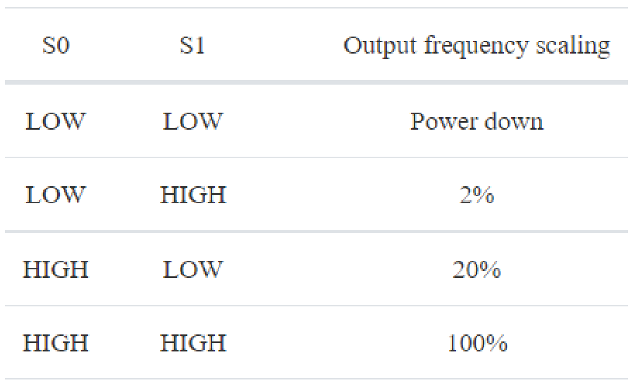
Each 16 photodiodes are connected in parallel. So, we can choose which of the them to read.

If you want to detect only red colour, you can select 16 red filtered photodiodes by setting two pins LOW according to the table. Similarly, you can choose different types of photodiodes by different combinations of S2 and S3.



S2	S3	Photodiode type
LOW	LOW	Red
LOW	HIGH	Blue
HIGH	LOW	Clear (No filter)
HIGH	HIGH	Green

An internal current-to-frequency converter converts readings from photodiodes into a square wave whose frequency is proportional to the intensity of the chosen colour. The range of the typical output frequency is 2HZ~500KHZ.



S0	S1	Output frequency scaling
LOW	LOW	Power down
LOW	HIGH	2%
HIGH	LOW	20%
HIGH	HIGH	100%

The sensor has two more control pins, S0 and S1, which are used for scaling the output frequency. The frequency can be scaled to three different preset values of 2%, 20% or 100%. This frequency-scaling function allows the sensor to be used with a variety of microcontrollers and other devices.

You can get different scaling factor by different combinations of S0 and S1. For the Arduino most applications use the 20% scaling.

3.2.4 Pin Diagram

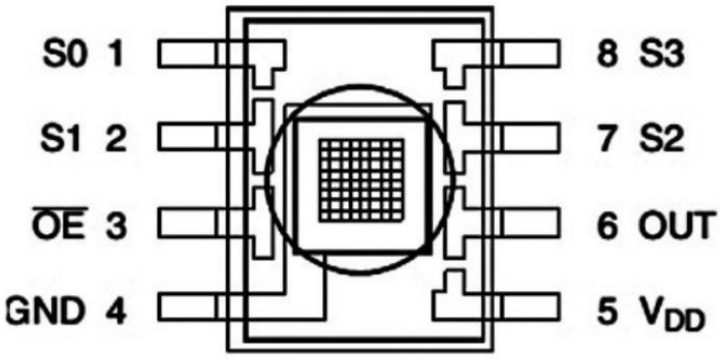


Fig 3.8 TCS3200 Pin Diagram

- Pin-1 and Pin-2 are S0, S1 selection lines respectively. These pins are used for frequency scaling.
- Pin-3, OE, is the output enable pin. This pin is active low.
- Pin-4, GND, is the ground pin. This pin is the power supply ground.
- Pin-5, VDD, is the supply voltage pin.
- Pin-6, OUT, is the output frequency pin. This pin is connected to the microcontroller to read values.
- PIn-7 and Pin-8 are the S2, S3 selection lines respectively. These pins are used as Photodiode type selection inputs.

3.2.5 Specifications

1. The main blocks of this module are TCS3200 RGB sensor chip and 4 white LEDs.
2. The four LED’s are given to provide sufficient lighting conditions for the sensor during colour detection.
3. TCS3200 chip consists of 8×8 array of photodiodes which can detect red, blue, green colours.
4. This module works on the input supply voltage of 2.7V to 5.5V.
5. TCS3200 chip converts light intensity into the frequency with high resolution.
6. This module doesn’t require an ADC to get digital values and can be connected to digital pins of microcontrollers directly.
7. TCS3200 has a programmable colour and full-scale output frequency.
8. Power down feature is also given to this module.
9. The operating temperature range of this module is from -40°C to 85°C.

3.2.5 Wiring TCS230 Color Sensor to Arduino UNO

Wiring up the TCS 230 to an Arduino is very simple. Every pin is used except the Output Enable pin, and the module is powered safely from the 5-volt output of the Arduino. None of the pins used on the Arduino are critical because the module does not require any pin-specific features, so if you want to use different pins you can do so safely. Just be sure to change the pin numbers in the code to reflect any changes to the wiring.

We actually use two sketches to work with the TCS230 colour sensor.

1. The first sketch (calibration sketch) will help us to obtain the raw data from the sensor.
2. The second sketch (main Arduino sketch) will use the raw data previously received to display RGB values for the colour being sensed.

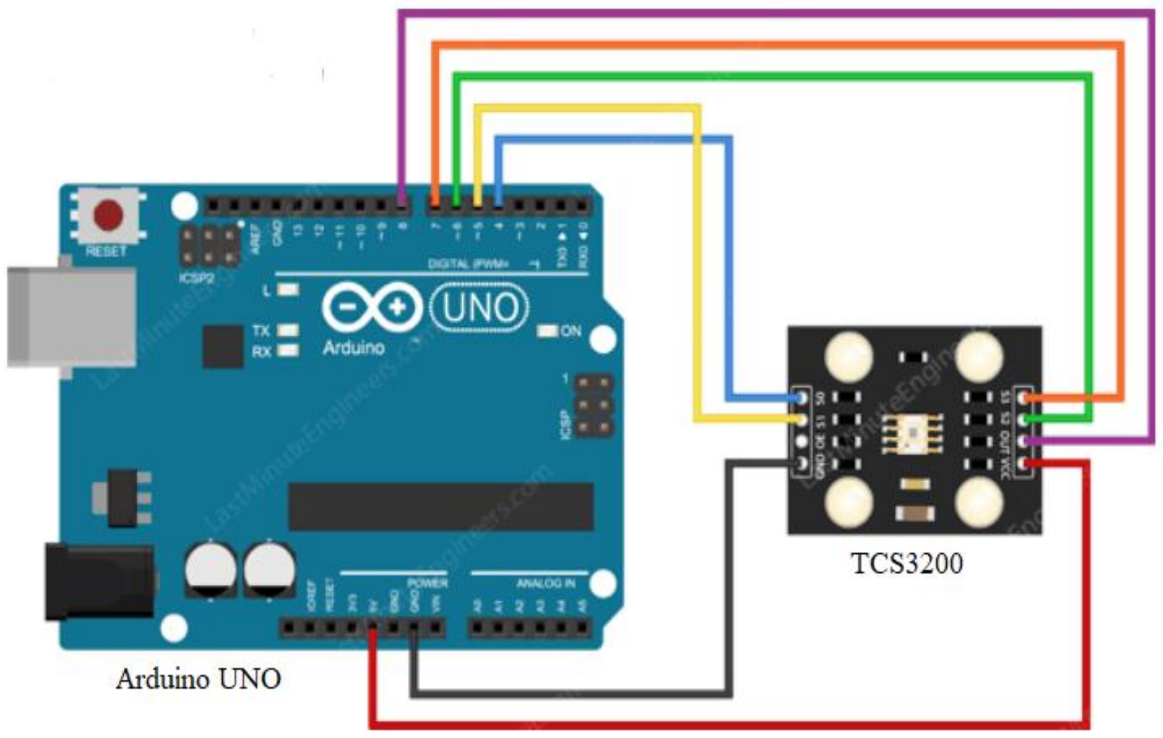


Fig 3.9 Wiring TCS230 Color Sensor to Arduino UNO

3.2.5 Applications of TCS3200

1. TCS3200 is used to detect the colour of the surfaces.
2. This module finds application in Industries, health care, and manufacturing plants.
3. TCS3200 is applied in medical diagnosis systems.
4. For RGB LED consistency control TCS3200 is used.
5. TCS3200 is applied for industrial process control.
6. In laser edge banding machines to detect colour TCS3200 is used.
7. To detect chronic kidney diseases TCS3200 is used for urine analysis.
8. TCS3200 is used in fruit sorting systems.
9. The intensity of blue, red, green radiations can be measured using this sensor module.
10. For classifying different types of metals this module is used.
11. The food industry uses this sensor for sorting of fruits and vegetables.
12. TCS3200 is also used in dental diagnosis and for ammonia detection.
13. For designing of Light-to-Frequency receiver in Multi colour visible light communication, TCS3200 is used.

3.3 SG90 SERVO MOTOR

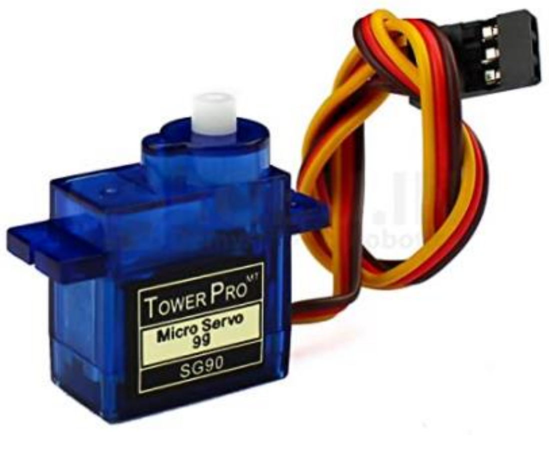


Fig 3.10 SG90 Micro Servo

A **servo motor** is a type of motor that can rotate with great precision. If you want to rotate an object at some specific angles or distance, then you use a servo motor. It is just made up of a simple motor which runs through a **servo mechanism**.

3.3.1 Servo Motor Working Principle

A servo consists of a Motor (DC or AC), a potentiometer, gear assembly, and a controlling circuit. First of all, we use gear assembly to reduce RPM and to increase torque of the motor. Say at initial position of servo motor shaft, the position of the potentiometer knob is such that there is no electrical signal generated at the output port of the potentiometer. Now an electrical signal is given to another input terminal of the error detector amplifier. Now the difference between these two signals, one comes from the potentiometer and another comes from other sources, will be processed in a feedback mechanism and output will be provided in terms of error signal. This error signal acts as the input for motor and motor starts rotating. Now motor shaft is connected with the potentiometer and as the motor rotates so the potentiometer and it will generate a signal. So as the potentiometer's angular position changes, its output feedback signal changes. After sometime the position of potentiometer reaches at a position that the output of potentiometer is same as external signal provided. At this condition, there will be no output signal from the amplifier to the motor input as there is no difference between external applied signal and the signal generated at potentiometer, and in this situation motor stops rotating.

3.3.2 Servo Motor Working Mechanism

It consists of three parts:

1. Controlled device
2. Output sensor
3. Feedback system

It is a closed-loop system where it uses a positive feedback system to control motion and the final position of the shaft. Here the device is controlled by a feedback signal generated by comparing output signal and reference input signal.

Here reference input signal is compared to the reference output signal and the third signal is produced by the feedback system. And this third signal acts as an input signal to the control the device. This signal is present as long as the feedback signal is generated or there is a difference between the reference input signal and reference output signal.

3.3.3 Controlling Servo Motor

All motors have three wires coming out of them. Out of which two will be used for Supply (positive and negative) and one will be used for the signal that is to be sent from the MCU.

Servo motor is controlled by PWM (Pulse with Modulation) which is provided by the control wires. There is a minimum pulse, a maximum pulse and a repetition rate. Servo motor can turn 90 degree from either direction form its neutral position. The servo motor expects to see a pulse every 20 milliseconds (ms) and the length of the pulse will determine how far the motor turns. For example, a 1.5ms pulse will make the motor turn to the 90° position, such as if pulse is shorter than 1.5ms shaft moves to 0° and if it is longer than 1.5ms than it will turn the servo to 180°.

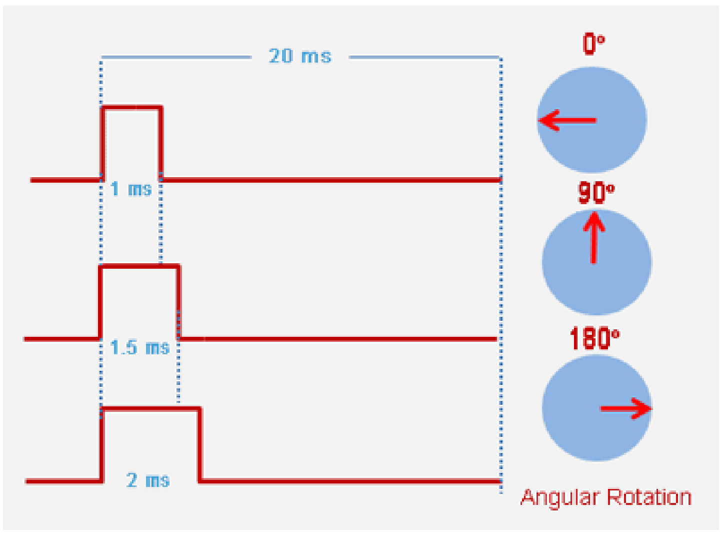


Fig 3.11 Controlling SG90

Servo motor works on **PWM (Pulse width modulation)** principle, means its angle of rotation is controlled by the duration of applied pulse to its Control PIN. Basically servo motor is made up of **DC motor which is controlled by a variable resistor (potentiometer) and some gears**. High speed force of DC motor is converted into torque by Gears. We know that WORK= FORCE X DISTANCE, in DC motor Force is less and distance (speed) is high and in Servo, force is High and distance is less. The potentiometer is connected to the output shaft of the Servo, to calculate the angle and stop the DC motor on the required angle.

Servo motor can be rotated from 0 to 180 degrees, but it can go up to 210 degrees, depending on the manufacturing. This degree of rotation can be controlled by applying the **Electrical Pulse** of proper width, to its Control pin. Servo checks the pulse in every 20 milliseconds. The pulse of 1 ms (1 millisecond) width can rotate the servo to 0 degrees, 1.5 ms can rotate to 90 degrees (neutral position) and 2 ms pulse can rotate it to 180 degree.

3.3.4 Specifications

- Operating Voltage: 3V to 7.2V
- Stall torque at 4.8V: 1.2 kg-cm
- Stall torque at 6.6V: 1.6 kg-c

3.3.5 Motor Wire Connections

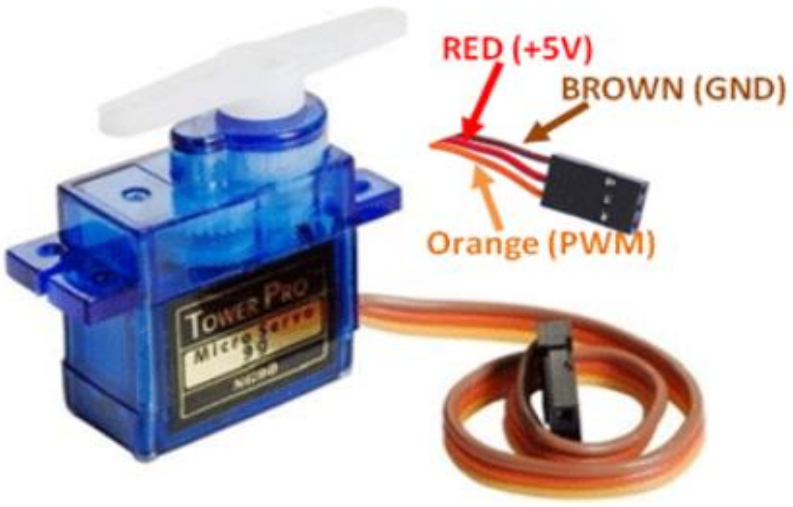


Fig 3.11 Wire Connections

- Orange - PWM
- Red - Supply
- Brown - Ground

CHAPTER 4

CONSTRUCTION

4.1 Materials used

4.1.1 Wooden Hardboard (Thickness: 2.5mm and 4 mm)

Hardboard, also called high-density fiberboard, is a type of fiberboard, which is an engineered wood product. Hardboard is similar to particle board and medium-density fiberboard, but is denser and much stronger and harder because it is made out of exploded wood fibers that have been highly compressed. Consequently, the density of hardboard is 500 kg/m3 (31 lb/cu ft) or more and is usually about 800–1,040 kg/m3 (50–65 lb/cu ft).It differs from particle board in that the bonding of the wood fibers requires no additional adhesive, the original lignin in the wood fibers sufficing to bond the hardboard together, although resin is often added. Unlike solid wood, hardboard is very homogeneous with no grain. A wood veneer can be glued onto it to give the appearance of solid wood.

In this project, wooden hardboards of two different thickness were used: 2.5mm and 4 mm. The 2.5 mm boards were used to make sides, platforms and back of the structure while the 4 mm was used to make the base. The wooden hardboards were suitable for this job because of their rigidity, strength and lightness. They were a bit challenging to work on as it is a hard material to cut. To overcome this problem, we used special cutters made to cut through this material.

Fig 4.1 Wooden Hardboard

4.1.2 Sunboard Sheet (Thicknes: 3mm)

Sunboard or Foam board is a very strong, light, and easily cut sheet material used for the mounting of vinyl prints, as backing in framing, and for painting. It usually has three layers — an inner layer of polystyrene foam and a white claycoated paper on the outside.Apart from printing, Sun board is commonly used to prototype objects, produce architectural models, and to make casting patterns. Backgrounds for scale model displays, dioramas, and games (computer) are often produced by hobbyists from sun board. It’s often useful for photographers as a reflector for light in the studio, and in the design industry to display presentations of new products; Another use is with aero-modellers for building small radio-controlled aircraft. But nowadays sun board is widely used in India and around the world for large format printing and store branding. Photos or creatives are first printed on a self-adhesive vinyl which is then mounted on to the sun board. This sun board can then be taken pasted on a wall, pillar or other flat surfaces using either double-sided adhesive tapes or nails depending on the surface. The more modern method is to print the sun board directly using a UV flatbed printing machine. UV prints on sun board last longer and are free from peeling problems.

Fig 4.2 Sunboard Sheet

In this project, sunboard sheet is the second most used material after wooden hardboard. These are extensively used in making the platforms. Since it is much easier to cut and shape, we placed sunboard sheets inside bigger cuts of the wooden hardboard. Then, the sunboards were cut finely to the exact shape and size. In the first platform, sunboard was used to make the place for hopper and color sensor. In the second platform, it was used to make the support platform for object while they are being transported from hopper to color sensor and finally dropped to the slide. The servos on second and third platforms were also mounted on a sunboard sheet. Small strips of the sheet were placed below sides to level the whole structure.

4.1.3 Non-Corrugated Paperboard (Thickness: 2mm)

Paperboard is a thick paper-based material. While there is no rigid differentiation between paper and paperboard, paperboard is generally thicker (usually over 0.30 mm, 0.012 in, or 12 points) than paper and has certain superior attributes such as foldability and rigidity. According to ISO standards, paperboard is a paper with a grammage above 250 g/m2, but there are exceptions. Paperboard can be single or multi-ply.

Paperboard can be easily cut and formed, is lightweight, and because it is strong, is used in packaging. Another end-use is high quality graphic printing, such as book and magazine covers or postcards. Paperboard is also used in fine arts for creating sculptures.

Fig 4.3 Non-Corrugated Paperboard

Sometimes it is referred to as cardboard, which is a generic, lay term used to refer to any heavy paper pulp–based board, however this usage is deprecated in the paper, printing and packaging industries as it does not adequately describe each product type.
In this project, paperboard was used to make the hopper and containers.

4.1.4 Card Stock (Thickness: 1mm)

Card stock, also called cover stock and pasteboard, is paper that is thicker and more durable than normal writing and printing paper, but thinner and more flexible than other forms of paperboard.

Fig 4.4 Card Stock

Card stock is often used for business cards, postcards, playing cards, catalogue covers, scrapbooking, and other applications requiring more durability than regular paper gives. The surface usually is smooth; it may be textured, metallic, or glossy. When card stock is labeled cover stock, it often has a glossy coating on one or both sides (C1S or C2S, for "coated: one side" or "coated: two sides"); this is used especially in business cards and book covers. This material was used for making the rotating arm in this project.

4.2 Process of construction

4.2.1 Box

The box has been constructed from wooden board of 2.5mm. The dimension of the box is 30cm x 10cm. We started out with a complete board and marked the cuts. Then with help of cutters, all the sides of the box were cut out. This was a tedious and challenging task as the material was very strong and stiff. Cutting through it was very difficult as we could only make a progress of 1/10th of a millimetre with one stroke of blade. On the sides, we sticked doubled strips of wooden boards at the places where platforms would rest.

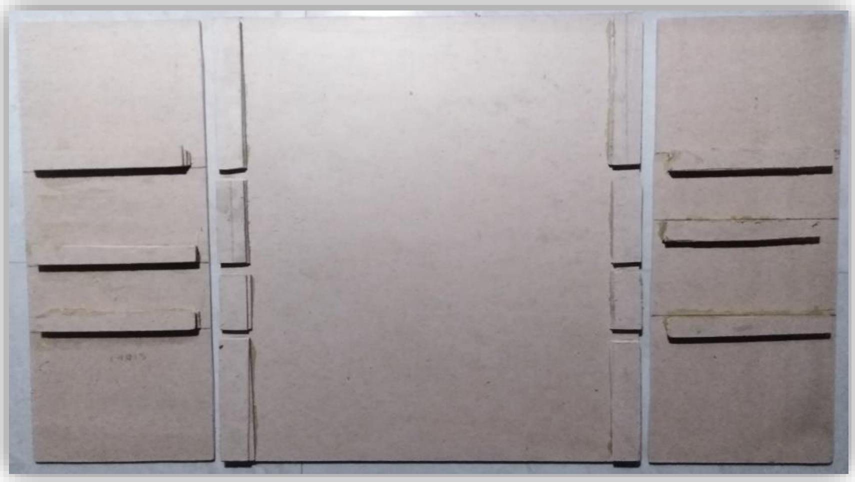


Fig. 4.5 Strips for Platforms

The adhesive used for this was fevi bond. It is a special rubber-based adhesive which is suitable for sticking wood-based materials. The strips would act like support for platforms. This gives a big advantage that the platforms would not need to be sticked permanently and can be taken out any time for repair and maintenance. The base of the box has been constructed from wooden board of 4mm.

Once all the parts were cut, they were put together with the help of hinges and fevibond. L-shaped hinges were used for this. 2 hinges per side were used, using a total of 6 hinges. Each of them was screwed with 2 screws.

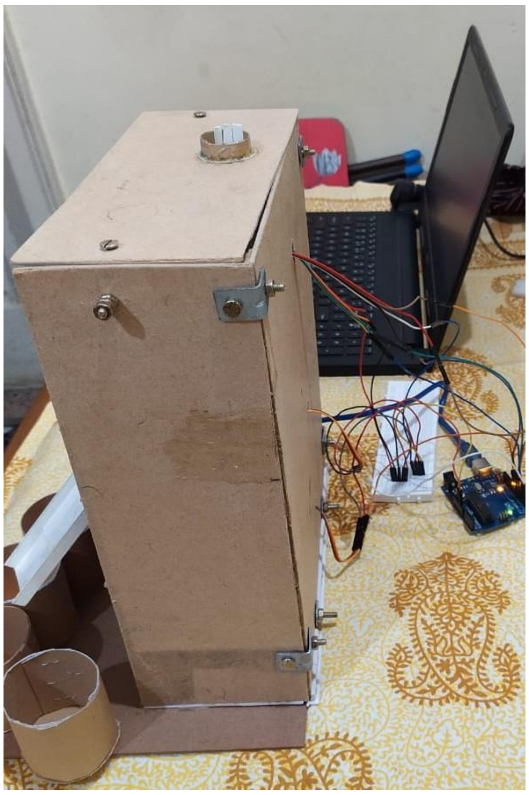


Fig. 4.6 Outer Structure of the box

4.2.2 Platform

Roof:

The roof houses the hopper so a hole of appropriate diameter was made in the roof. The hopper was constructed from millboard. Aligning the hopper at correct position and tilt was a challenging task. It took a lot of effort to place the hopper at 90o to the bottom platform.

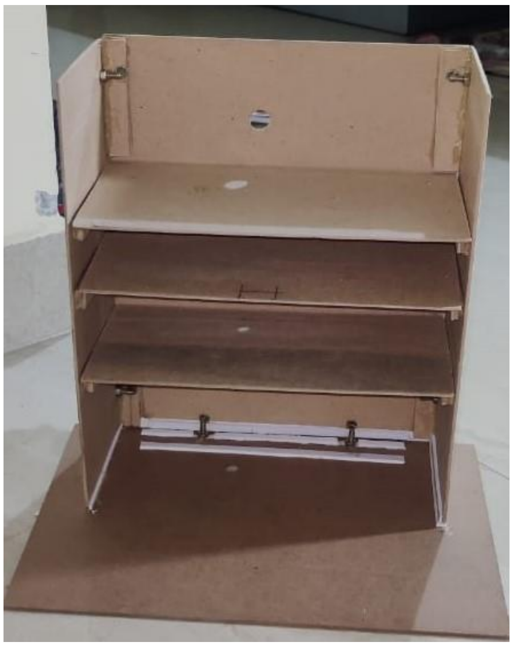


Fig 4.7 Alignment of Platforms

Platform 1:

This housed the color sensor as well as made way for the hopper. It was constructed from wooden board and sunboard. A bigger hole was first made in the platform and the sunboard was inserted into it. Then the holes for color sensor and hopper was made. Working with sunboard was easy as it is an easy material to cut.

Platform 2:

This platform has the top servo motor, the guiding rail and the gap for dropping object onto the slide. For constructing this platform, first the hole for the top servo motor was made. The rotating arm of the servo was made from card stock of a quadrant shape. A hole for carrying object was made at the end of the arm. It was fixed to the servo with fevibond. the servo was first inserted into a sunboard sheet cut to its size and then screwed. This mini structure was then inserted into the platform. It was raised with the help of stacked strips of sunboard.

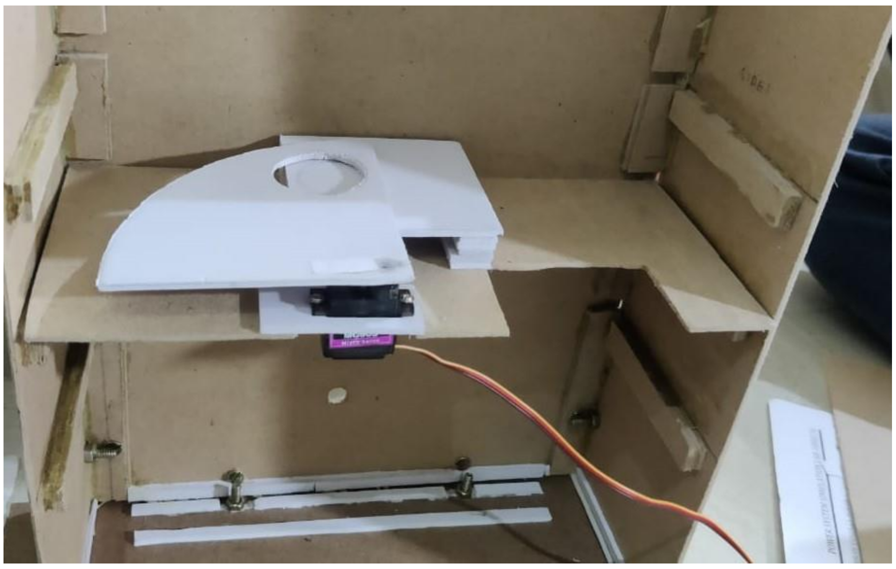


Fig 4.8 Platform 2

The second important part of this platform is the guiding rail. It was made of a square sheet of sunboard. It was also raised to the same height as servo arm with the help of strips of sunboard. A major challenge was to level every structure to the exact height. Numerous trial runs were conducted to ensure that the servo angle was not distorted due to surface unevenness. Accordingly, adjustments were made.

Platform 3:

This platform consists of the bottom servo motor and slide. Like the above platform, a grooved structure of sunboard was first made and the servo was inserted into it. The slide has been constructed from card stock. We observed there were vibrations during rotation of this servo because of not attaching the servo to the slide at its centre of gravity.

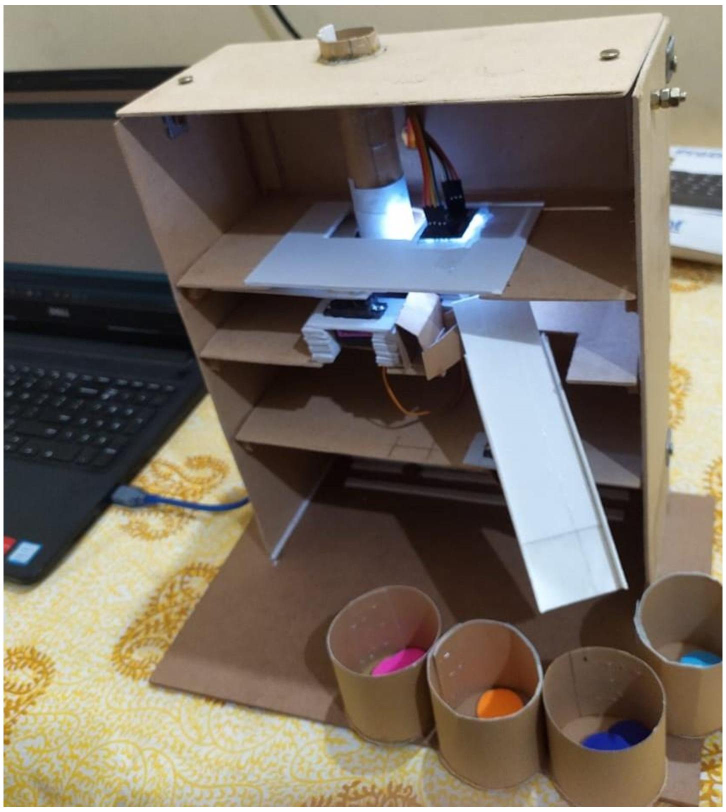


Fig 4.9 Platform 3 with slide

Another challenge was to set the inclination of slide such that it does not touch the sides as well as the objects can fall freely. Many trial runs were conducted and adjustments were made to make the system perfect.

Final Structure:

After assembling all the platforms, the hopper, the slide, the containers, and all the components, which are color sensor, servo motors and Arduino, the final structure was made ready for working.

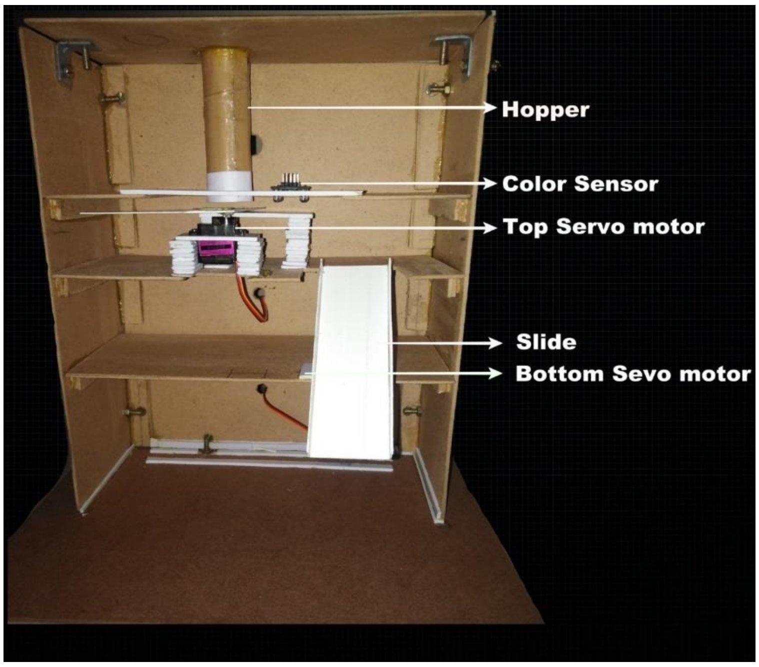


Fig. 4.10 Final Structure

All the components were needed to be placed at the calculated angles for the proper functioning of the model. The holes made on the back panel was used to pass all the wires to keep the electronics and wirings secured at the back side of the model.

CHAPTER 5

PROGRAMMING LOGIC AND FLOWCHART

The program begins at the color values sent by color sensor to the microcontroller. Using conditional statements these values are compared with pre-defined values of 6 pre-defined colors- red, green, blue, yellow, orange and white. If any of the conditions are satisfied, then the object is put into the respective container. If none of the conditions are satisfied, then the object is put in other color container.

The first condition will be checked. If the color of the object is red, then it will move to red container. If not then the second condition will be checked. If the color is blue, then the object will be moved to blue container. If not, then 3rd condition will be checked. If color is green then it will move to green container and this way, object will be checked for yellow, orange and white color. And if all the conditions fails, then since the object is neither of the pre-defined colors, the object will move to the other color container. After the objects gets dropped on its respective color container, a condition will be checked whether there are more objects in the hopper or not. If there will be more objects in the hopper, then the same process will be repeated for another object. This way all the objects will be sorted on basis of color. If there is no object in the hopper then the program will terminate and the machine will shut down. This way, all the objects present in the hopper will be sorted on the basis of color.

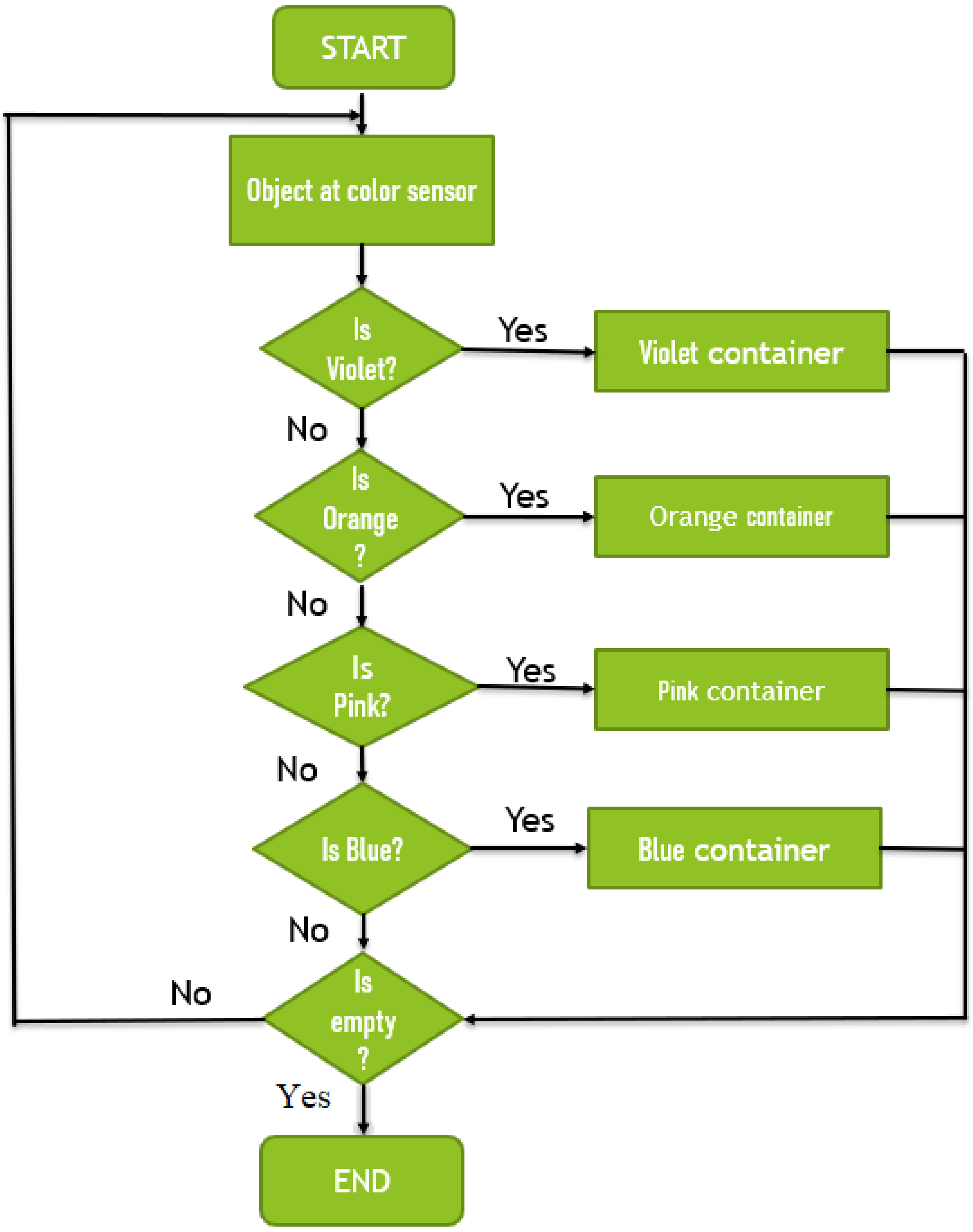


Fig 5.1 Flowchart

5.1 Pseudocode

```
program starts
initialize pins
while hopper is not empty
        if color == ORANGE then
                bottom servo.rotate(orange_angle)
        else if color == PINK then
                bottom servo.rotate(pink_angle)
        else if color == VIOLET then
                bottom servo.rotate(violet_angle)
        else if color == BLUE then
                bottom servo.rotate(blue_angle)
    end while
    program end
```

5.2 Programming Logic

The program starts. All the pins will be initialized. A loop will be used to sort the objects one by one in the respective containers on the basis of colour. If the statement in the loop is true, the loop executes and, inside the loop, using if-else-if block, we sort the objects into its respective colour containers. This way, all the objects will be sorted. Every time the loop executes, the hopper will be checked for the objects. If there are no more objects in the hopper, the loop condition fails and the program terminates.

5.3 Servo.h

5.3.1 Usage

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. To use this library: **<include<Servo.h>**

5.3.2 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.

5.3.3 Knob Circuit

For the **Knob** example, wire the potentiometer so that its two outer pins are connected to power (+5V) and ground, and its middle pin is connected to A0 on the board. Then, connect the servo motor to +5V, GND and pin 9.

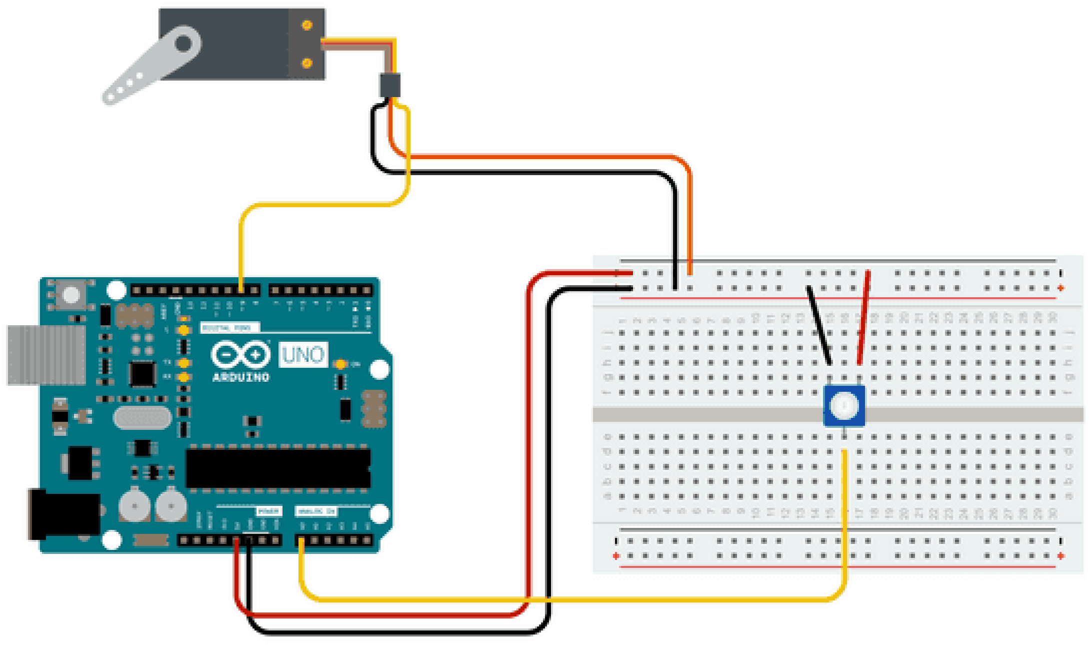


Fig 5.2 Knob Circuit

Code to control a servo position using a potentiometer (variable resistor).

```
#include <Servo.h>

Servo myservo;  // create servo object to control a servo

int potpin = 0;  // analog pin used to connect the potentiometer

int val;    // variable to read the value from the analog pin

void setup() {

    myservo.attach(9);  // attaches the servo on pin 9 to the servo object

}

void loop() {

    val = analogRead(potpin);            // reads the value of the potentiometer

    val = map(val, 0, 1023, 0, 180);     // scale it to use it with the servo

    myservo.write(val);       // sets the servo position according to the scaled value

    delay(15);                           // waits for the servo to get there

}
```

5.3.4 Sweep Circuit

For the **Sweep** example, connect the servo motor to +5V, GND and pin 9.

Code to sweep the shaft of a RC servo motor back and forth across 180 degrees

```
#include <Servo.h>

Servo myservo;  // create servo object to control a servo

int pos = 0;    // variable to store the servo position
```

```
void setup() {

   myservo.attach(9);  // attaches the servo on pin 9 to the servo object

}

void loop() {

   for (pos = 0; pos <= 180; pos += 1) {

            myservo.write(pos);              // tell servo to go to position in variable 'pos'

            delay(15);                       // waits 15ms for the servo to reach the position

   }

   for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees

            myservo.write(pos);              // tell servo to go to position in variable 'pos'

            delay(15);                       // waits 15ms for the servo to reach the position

   }

}
```

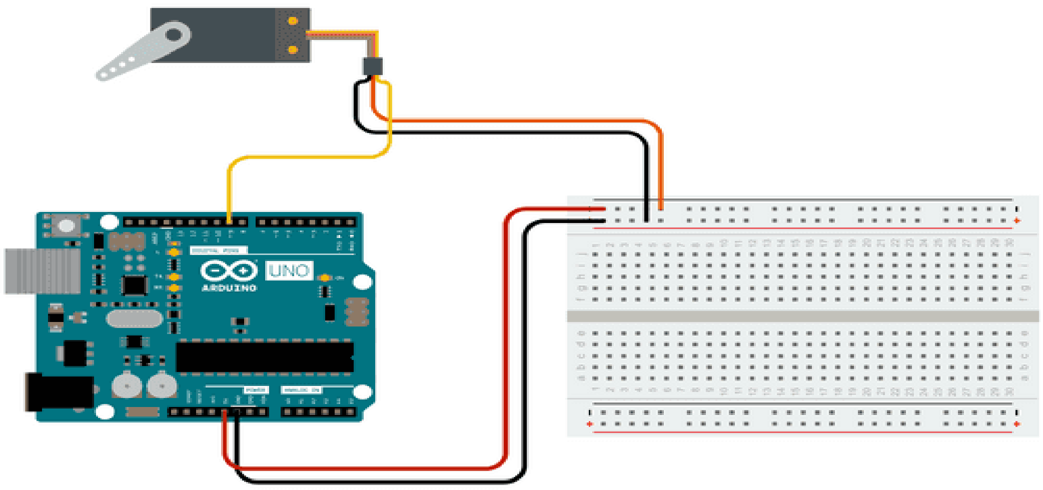


Fig 5.3 Sweep Circuit

5.3.5 Servo – attach()

Attach the Servo variable to a pin. Note that in Arduino 0016 and earlier, the Servo library supports servos on only two pins: 9 and 10.

Syntax

servo.attach (pin)

servo.attach (pin, min, max)

Parameter

- servo: a variable of type Servo
- pin: the number of the pin that the servo is attached to
- min (optional): the pulse width, in microseconds, corresponding to the minimum (0 degree) angle on the servo (defaults to 544)
- max (optional): the pulse width, in microseconds, corresponding to the maximum (180 degree) angle on the servo (defaults to 2400)

Example code

```
#include <Servo.h>

Servo myservo;

void setup(){

    myservo.attach(9);

}

void loop(){}
```

5.3.6 Servo - write()

Writes a value to the servo, controlling the shaft accordingly. On a standard servo, this will set the angle of the shaft (in degrees), moving the shaft to that orientation. On a continuous rotation servo, this will set the speed of the servo (with 0 being full-speed in one direction, 180 being full speed in the other, and a value near 90 being no movement).

Syntax

servo.write(angle)

Parameters

- servo: a variable of type Servo
- angle: the value to write to the servo from 0 to 180

Example Code

```
#include<Servo.h>

Servo myservo;

void Setup(){

    myservo.attach(9);

    myservo.write(90);

}

void loop(){}
```

5.3.7 Servo - writeMicroseconds()

Writes a value in microseconds (us) to the servo, controlling the shaft accordingly. On a standard servo, this will set the angle of the shaft. On standard servos a parameter value of 1000 is fully counter-clockwise, 2000 is fully clockwise, and 1500 is in the middle.

Note that some manufactures do not follow this standard very closely so that servos often respond to values between 700 and 2300. Feel free to increase these endpoints until the servo no longer continues to increase its range. Note however that attempting to drive a servo past its endpoints (often indicated by a growling sound) is a high-current state, and should be avoided.

Continuous-rotation servos will respond to the writeMicrosecond function in an analogous manner to the write function.

Syntax

servo.writeMicroseconds(us)

Parameters

- servo: a variable of type Servo
- us: the value of the parameter in microseconds(int)

Example code

```
#include<Servo.h>

Servo myservo;

void Setup(){

    myservo.attach(9);

    myservo.writeMicroseconds(1500);

}

void loop(){}
```

5.3.8 Servo - read()

Read the current angle of the servo (the value passed to the last call to write()).

Syntax

servo.read()

Returns

The angle of the servo, from 0 to 180 degrees.

5.3.9 Servo - attached()

Check whether the Servo variable is attached to a pin.

Syntax

servo.attached()

Returns

true if the servo is attached to pin; otherwise **false**.

5.3.10 Servo - detach()

Detach the Servo variable from its pin. If all Servo variables are detached, then pins 9 and 10 can be used for PWM output with analogWrite().

Syntax

servo.detach()

Parameters

servo: a variable of type Servo

5.4 Code

```
#include<Servo.h>
#define S0 4
#define S1 5
#define S2 6
#define S3 7
#define sensorOut 8
// Variables for Color Pulse Width Measurements
Servo topServo;
Servo bottomServo;

int R = 0;
int G = 0;
int B = 0;
bool isempty = false;

void setup() {
    // Set S0 - S3 as outputs
    pinMode(S0, OUTPUT);
    pinMode(S1, OUTPUT);
    pinMode(S2, OUTPUT);
    pinMode(S3, OUTPUT);

    // Set Pulse Width scaling to 20%
    digitalWrite(S0, HIGH);
    digitalWrite(S1, LOW);

    topServo.attach(2);
    bottomServo.attach(10);
    // Set Sensor output as input
    pinMode(sensorOut, INPUT);
    // Setup Serial Monitor
    Serial.begin(9600);
}

void loop() {
    topServo.write(90);
    for (int i = 90; i >= 30; i--) {
            topServo.write(i);
            delay(2);
    }
    delay(500);

    for (int i = 0; i < 6; i++)
    {
            // Read Red Pulse Width
            R = getRedPW();
            // Delay to stabilize sensor
            delay(200);

            // Read Green Pulse Width
            G = getGreenPW();
            // Delay to stabilize sensor
            delay(200);

            // Read Blue Pulse Width
            B = getBluePW();
            // Delay to stabilize sensor
            delay(200);



            // Print output to Serial Monitor
            if (i < 5)
            {
                    Serial.print("R = ");
```

```
        Serial.print(R);
        Serial.print(" - G = ");
        Serial.print(G);
        Serial.print(" - B = ");
        Serial.println(B);
    }

    else
    {
        if (R <= 120 && R >= 110 && B >= 300 && B <= 310)
        {
            Serial.println("Orange");
            bottomServo.write(30);
        }
        else if (R <= 140 && R >= 130 && G >= 380 && G <= 400)
        {
            Serial.println("Pink");
            bottomServo.write(0);
        }
        else if (R <= 360 && R >= 340 && B <= 170 && B >= 150)
        {
            Serial.println("Blue");
            bottomServo.write(90);
        }
        else if (R >= 390 && R <= 400 && G <= 645 && G >= 620)
        {
            Serial.println("Violet");
            bottomServo.write(60);
        }
        else if (R < 250 && B < 250 && G < 250)
        {
            Serial.println("Empty");
            isempty = true;
        }
```

```
                else
                {
                        bottomServo.write(90);
                }
        }
  }

  if (!isempty)
  {
        delay(500);
        for (int i = 30; i >= 0; i--) {
                topServo.write(i);
                delay(4);
        }

        delay(200);
        for (int i = 0; i <= 90; i++) {
                topServo.write(i);
                delay(2);
        }
        delay(500);
  }

  else
  {
        for (int i = 30; i <= 90; i++)
        {
                topServo.write(i);
                delay(4);
        }
        exit(0);
  }
}
```

```
// Function to read Red Pulse Widths
int getRedPW() {
    // Set sensor to read Red only
    digitalWrite(S2, LOW);
    digitalWrite(S3, LOW);
    // Define integer to represent Pulse Width
    int PW;
    // Read the output Pulse Width
    PW = pulseIn(sensorOut, LOW);
    // Return the value
    return PW;
}

// Function to read Green Pulse Widths
int getGreenPW() {
    // Set sensor to read Green only
    digitalWrite(S2, HIGH);
    digitalWrite(S3, HIGH);
    // Define integer to represent Pulse Width
    int PW;
    // Read the output Pulse Width
    PW = pulseIn(sensorOut, LOW);
    // Return the value
    return PW;
}

// Function to read Blue Pulse Widths
int getBluePW() {
    // Set sensor to read Blue only
    digitalWrite(S2, LOW);
    digitalWrite(S3, HIGH);
    // Define integer to represent Pulse Width
    int PW;
    // Read the output Pulse Width
    PW = pulseIn(sensorOut, LOW);
    // Return the value
    return PW;
}
```

5.5 Explanation of code

5.5.1 Logic

We will include servo.h library in order to enable all the functions and methods of micro servo inside our code using #include <Servo.h> command.
We will define as S0 as pin number 4, S1 as pin number 5, S2 as pin number 6, S3 as pin number 7 and sensorOut as pin number 8.

We will initialize two variables topServo and bottomServo of type servo.
We will initialize three integers R, G, B and we will assign 0 to all three of them initially and then we will initialize a variable isEmpty of type Boolean. Initially isEmpty will be false.

The first function is of name setup of type void which will determine the pin modes of S0, S1, S2, S3 as output.

We will use digitalWrite command to set S0 and S1 as low. Top servo is attached to pin number 2 and bottom servo is attached to pin number 10.

In the void loop() function first, the top server will be at the initial angle of 90 degree and the for loop will move the top servo from 90 degree to 30 degree with a delay of 2 microseconds, carrying the object along with it towards the color sensor.

After that there will be a delay of 500 microseconds so that the color sensor may read the color and return the RGB values of the color. R value will be fetched using getRedPW() function, G value will be fetched from getGreenPW() function and B value will be returned from getBluePW() function.

The color of the object will be determined using the conditional statements and the color will be printed in the serial monitor and the objects will be sorted on the basis of color. There will be conditional statements to determine the color of the object.

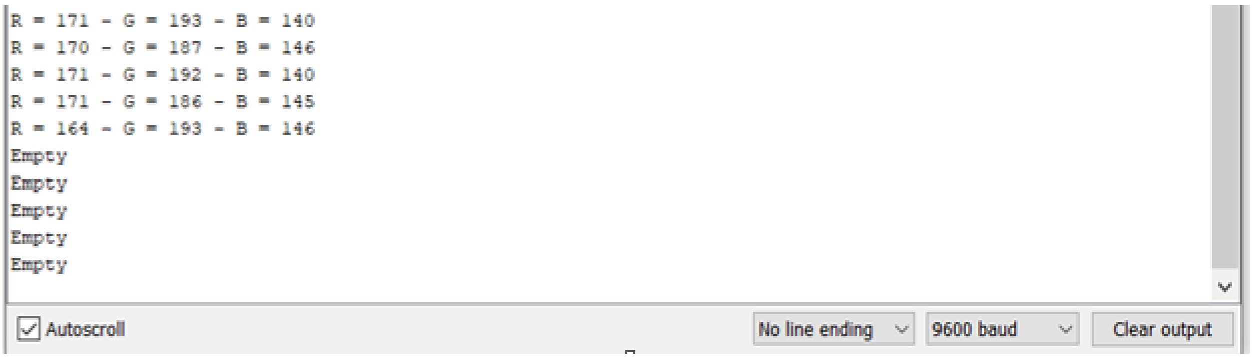
If R value lies between 110 and 120 and B value lies between 300 and 310, the color of the object is orange and the bottom servo will move to the angle of 30 degree.

If value of R is in the range of 130 to 140, G value is in the range of 380 to 400, then the color of the object is pink and the bottom servo will move to the angle of 0 degree.

If the R value lies between 340 to 360 and B value lies between 150 to 170 then the color will be blue and bottom servo will move to the angle of 90 degree.

If the value of R is in the range of 390 to 400 and G value is in range of 620 to 645 then the color of the object is violet and the bottom servo will move to angle of 60 degrees.

If R value is less than 250, G is less than 250and B value is less than 250 then there is no object below the color sensor and isEmpty variable will become true and after a delay of 500 microseconds, the top servo will move back to its initial position and the program will terminate.



```
R = 171 - G = 193 - B = 140
R = 170 - G = 187 - B = 146
R = 171 - G = 192 - B = 140
R = 171 - G = 186 - B = 145
R = 164 - G = 193 - B = 146
Empty
Empty
Empty
Empty
Empty
```

Fig. 5.4 Serial Monitor Displaying Empty

Now, depending on the color that the color sensor read on the basis of RGB value, the bottom servo will move, carrying the slide to a particular angle.

The top servo will move further to drop the object which will be carried by the slide to its respective container. This way, all the objects will get sorted on the basis of their color.

5.5.3 getRedPW():

This function returns the frequency of red color from sensor. The S2 and S3 pins on the sensor are the color selection pins and are used to obtain frequency of RGB using different combinations. For getting the frequency of red color, both S2 and S3 are set to LOW. A variable named PW is defined. The value of the red color is then stored in this variable using pulsein() function and this value is returned. This function reads a pulse (either HIGH or LOW) on a pin. For example, if value is HIGH, pulseIn() waits for the pin to go from LOW to HIGH, starts timing, then waits for the pin to go LOW and stops timing. Returns the length of the pulse in microseconds or gives up and returns 0 if no complete pulse was received within the timeout.

5.5.4 getGreenPW():

This function returns the frequency of green color from sensor. The S2 and S3 pins on the sensor are the color selection pins and are used to obtain frequency of RGB using different combinations. For getting the frequency of green color, both S2 and S3 are set to HIGH. A variable named PW is defined. The value of the green color is then stored in this variable using pulsein() function and this value is returned. This function reads a pulse (either HIGH or LOW) on a pin. For example, if value is HIGH, pulseIn() waits for the pin to go from LOW to HIGH, starts timing, then waits for the pin to go LOW and stops timing. Returns the length of the pulse in microseconds or gives up and returns 0 if no complete pulse was received within the timeout.

5.5.5 getBluePW():

This function returns the frequency of blue color from sensor. The S2 and S3 pins on the sensor are the color selection pins and are used to obtain frequency of RGB using different combinations. For getting the frequency of blue color, S2 is set to LOW and S3 is set to HIGH. A variable named PW is defined. The value of the blue color is then stored in this variable using pulsein() function and this value is returned.

This function reads a pulse (either HIGH or LOW) on a pin. For example, if value is HIGH, pulseIn() waits for the pin to go from LOW to HIGH, starts timing, then waits for the pin to go LOW and stops timing. Returns the length of the pulse in microseconds or gives up and returns 0 if no complete pulse was received within the timeout.

5.5.6 pinMode()

Configures the specified pin to behave either as an input or an output.

Syntax:

pinMode(pin, mode);

- pin: the Arduino pin number to set the mode of.
- mode: INPUT, OUTPUT, INPUT_PULLUP

Example Code:

```
void setup() {
   pinMode(13, OUTPUT);    // sets the digital pin 13 as output
}

void loop() {
   digitalWrite(13, HIGH); // sets the digital pin 13 on
   delay(1000);            // waits for a second
   digitalWrite(13, LOW);  // sets the digital pin 13 off
   delay(1000);            // waits for a second
}
```

5.5.7 digitalWrite()

Write a HIGH or a LOW value to a digital pin.

If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW.
If the pin is configured as an INPUT, digitalWrite() will enable (HIGH) or disable (LOW) the internal pullup on the input pin. It is recommended to set the pinMode() to INPUT_PULLUP to enable the internal pull - up resistor. See the Digital Pins tutorial for more information.

If you do not set the pinMode() to OUTPUT, and connect an LED to a pin, when calling digitalWrite(HIGH), the LED may appear dim. Without explicitly setting pinMode(), digitalWrite() will have enabled the internal pull - up resistor, which acts like a large current - limiting resistor.

Syntax:

digitalWrite(pin, value)

Parameters:

- pin: the Arduino pin number.
- value: HIGH or LOW.

Returns:

Nothing

Example Code:

The code makes the digital pin 13 an OUTPUT and toggles it by alternating between HIGH and LOW at one second pace.

```
void setup() {
    pinMode(13, OUTPUT);    // sets the digital pin 13 as output
}

void loop() {
    digitalWrite(13, HIGH); // sets the digital pin 13 on
    delay(1000);            // waits for a second
    digitalWrite(13, LOW);  // sets the digital pin 13 off
    delay(1000);            // waits for a second
}
```

5.5.8 Serial.print()

Prints data to the serial port as human - readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit.

Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are sent as a single character. Characters and strings are sent as is. For example -

Serial.print(78) gives "78"
Serial.print(1.23456) gives "1.23"
Serial.print('N') gives "N"
Serial.print("Hello world.") gives "Hello world."

An optional second parameter specifies the base (format) to use; permitted values are BIN(binary, or base 2), OCT(octal, or base 8), DEC(decimal, or base 10), HEX(hexadecimal, or base 16). For floating point numbers, this parameter specifies the number of decimal places to use. For example -

Serial.print(78, BIN) gives "1001110"
Serial.print(78, OCT) gives "116"
Serial.print(78, DEC) gives "78"
Serial.print(78, HEX) gives "4E"
Serial.print(1.23456, 0) gives "1"
Serial.print(1.23456, 2) gives "1.23"
Serial.print(1.23456, 4) gives "1.2346"

You can pass flash - memory based strings to Serial.print() by wrapping them with F(). For example:
Serial.print(F("Hello World"))
To send data without conversion to its representation as characters, use Serial.write().

Syntax:
Serial.print(val)
Serial.print(val, format)

Parameters:

- Serial: serial port object. See the list of available serial ports for each board on the Serial main page.
- val : the value to print. Allowed data types : any data type.

Returns:

print() returns the number of bytes written, though reading that number is optional. Data type : size_t.

Example Code:

```
void setup() {
    Serial.begin(9600); // open the serial port at 9600 bps:
}

void loop() {
    // print labels
    Serial.print("NO FORMAT");  // prints a label
    Serial.print("\t");         // prints a tab

    Serial.print("DEC");
    Serial.print("\t");

    Serial.print("HEX");
    Serial.print("\t");

    Serial.print("OCT");
    Serial.print("\t");

    Serial.print("BIN");
    Serial.println();       // carriage return after the last label

    for (int x = 0; x < 64; x++) { // only part of the ASCII chart, change to suit
            // print it out in many formats:
            Serial.print(x);       // print as an ASCII-encoded decimal - same as "DEC"
            Serial.print("\t\t");  // prints two tabs to accomodate the label lenght

            Serial.print(x, DEC);  // print as an ASCII-encoded decimal
            Serial.print("\t");    // prints a tab
            Serial.print(x, HEX);  // print as an ASCII-encoded hexadecimal
            Serial.print("\t");    // prints a tab

            Serial.print(x, OCT);  // print as an ASCII-encoded octal
            Serial.print("\t");    // prints a tab

            Serial.println(x, BIN);  // print as an ASCII-encoded binary
            // then adds the carriage return with "println"
            delay(200);            // delay 200 milliseconds
    }
    Serial.println();       // prints another carriage return
}
```

5.5.9 Serial.print()

Prints data to the serial port as human - readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit. Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are sent as a single character. Characters and strings are sent as is.

For example -
Serial.print(78) gives "78"
Serial.print(1.23456) gives "1.23"
Serial.print('N') gives "N"
Serial.print("Hello world.") gives "Hello world."

An optional second parameter specifies the base (format) to use; permitted values are BIN(binary, or base 2), OCT(octal, or base 8), DEC(decimal, or base 10), HEX(hexadecimal, or base 16). For floating point numbers, this parameter specifies the number of decimal places to use. For example -

Serial.print(78, BIN) gives "1001110"
Serial.print(78, OCT) gives "116"
Serial.print(78, DEC) gives "78"

Serial.print(78, HEX) gives "4E"

Serial.print(1.23456, 0) gives "1"

Serial.print(1.23456, 2) gives "1.23"

Serial.print(1.23456, 4) gives "1.2346"

You can pass flash - memory based strings to Serial.print() by wrapping them with F(). For example:

Serial.print(F("Hello World"))

To send data without conversion to its representation as characters, use Serial.write().

Syntax:

Serial.print(val)

Serial.print(val, format)

Parameters:

- Serial: serial port object. See the list of available serial ports for each board on the Serial main page.
- val : the value to print. Allowed data types : any data type.

Returns:

print() returns the number of bytes written, though reading that number is optional.

Data type : size_t.

Example Code:

```
void setup() {
    Serial.begin(9600); // open the serial port at 9600 bps:
}

void loop() {
    // print labels
    Serial.print("NO FORMAT");  // prints a label
    Serial.print("\t");         // prints a tab
    Serial.print("DEC");
    Serial.print("\t");

    Serial.print("HEX");
    Serial.print("\t");

    Serial.print("OCT");
    Serial.print("\t");

    Serial.print("BIN");
    Serial.println();        // carriage return after the last label

    for (int x = 0; x < 64; x++) { // only part of the ASCII chart, change to suit
            // print it out in many formats:
            Serial.print(x);       // print as an ASCII-encoded decimal - same as "DEC"
            Serial.print("\t\t");  // prints two tabs to accomodate the label lenght

            Serial.print(x, DEC);  // print as an ASCII-encoded decimal
            Serial.print("\t");    // prints a tab

            Serial.print(x, HEX);  // print as an ASCII-encoded hexadecimal
            Serial.print("\t");    // prints a tab

            Serial.print(x, OCT);  // print as an ASCII-encoded octal
            Serial.print("\t");    // prints a tab

            Serial.println(x, BIN);  // print as an ASCII-encoded binary
            // then adds the carriage return with "println"
            delay(200);            // delay 200 milliseconds
    }
    Serial.println();        // prints another carriage return
}
```

5.5.10 setup()

The **setup()** function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup() function will only run once, after each powerup or reset of the Arduino board.

Example Code

```
int buttonPin = 3;

void setup() {
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
}

void loop() {
  // ...
}
```

5.5.11 loop()

After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

Example Code

```
int buttonPin = 3;

// setup initializes serial and the button pin
void setup() {
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
}
```

```
// loop checks the button pin each time,
// and will send serial if it is pressed
void loop() {
  if (digitalRead(buttonPin) == HIGH) {
    Serial.write('H');
  }
  else {
    Serial.write('L');
  }
  delay(1000);
}
```

CHAPTER 6

APPLICATIONS

Application of this project can be seen in multiple sector such as in agricultural sector, food processing sector and in industrial sector.

In agricultural sector, we can consider a case where a farmer produces rice and grains etc. Now, the produced grains will be containing unwanted early materials such as mud, stones etc. These unwanted materials will be sorted based on the colour of the grain. The colour will be fed through the microcontroller and depending on the colour fed, all the unwanted materials will be removed.

In industrial sector, suppose there is a factory manufacturing balls, all of different colours. These colour balls, if sorted based on colour manually using physical labour, will take lot of time and energy and thus the cost of manufacturing will also increase drastically. By using the colour sorting machine, all the different colour balls will be sorted accordingly and the manufacturing process and packaging process will be enhanced drastically.

This project can be used for the industrial purpose in order to reduce the manual labor and improve automation. As we know, sorting of objects on the basis of color is a very frequent task in the industry of almost all sector such as food processing, rice, grains, pulses packaging, etc. This project not only help in reducing manual labor but also reduces production cost by reducing the number or labors. Above all, this machine also improves time-efficiency by automating the process making sorting way faster than when done manually.

CHAPTER 7

CONCLUSION

Sorting of objects based on color is an essential mechanical process in the manufacturing industry. Manually doing this task consumes a lot of time, money and energy. Product consistency and quality is also very poor. To obtain high quality product with low cost and time investment, a machine which can automate this task is the need of the day. So, the objective of this project is to make and demonstrate a color sorting machine that can be used in real world applications. Thus, this project can be used for the industrial purpose in order to reduce the manual labor and improve automation. As we know, sorting of objects on the basis of color is a very frequent task in the industry of almost all sector such as food processing, rice, grains, pulses packaging, etc. This project not only help in reducing manual labor but also reduces production cost by reducing the number or labors. Above all, this machine also improves time-efficiency by automating the process making sorting way faster than when done manually.

APPENDIX

1. ATmega328P Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP[1]	k	Direct Jump	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL[1]	k	Direct Subroutine Call	PC ← k	None	4
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1/2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1/2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0...6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3...0)←Rd(7...4),Rd(7...4)←Rd(3...0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2

2. Electrical Characteristics of TCS3200

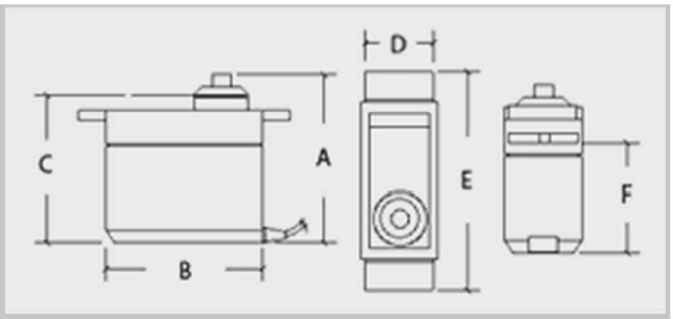
PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
V_{OH}	High-level output voltage	I_{OH} = − 2 mA	4	4.5		V
V_{OL}	Low-level output voltage	I_{OL} = 2 mA		0.25	0.40	V
I_{IH}	High-level input current				5	µA
I_{IL}	Low-level input current				5	µA
I_{DD}	Supply current	Power-on mode		1.4		mA
		Power-down mode			0.1	µA
	Full-scale frequency (See Note 4)	S0 = H, S1 = H	500	600		kHz
		S0 = H, S1 = L	100	120		kHz
		S0 = L, S1 = H	10	12		kHz
	Temperature coefficient of responsivity	λ ≤ 700 nm, −25°C ≤ T_A ≤ 70°C		± 200		ppm/°C
k_{SVS}	Supply voltage sensitivity	V_{DD} = 5 V ±10%		±0.5		%/ V

3. Operating Characteristics at V_{DD} = 5 V, T_A = 25 Degree C, S0 = H, S1 = H values for TCS3200

PARAMETER		TEST CONDITIONS	CLEAR PHOTODIODE S2 = H, S3 = L			BLUE PHOTODIODE S2 = L, S3 = H			GREEN PHOTODIODE S2 = H, S3 = H			RED PHOTODIODE S2 = L, S3 = L			UNIT
			MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX	
f_O	Output frequency (Note 9)	E_e = 47.2 µW/cm^2, λ_p = 470 nm	12.5 (4.7)	15.6 (5.85)	18.7 (7)	61%		84%	22%		43%	0%		6%	kHz
		E_e = 40.4 µW/cm^2, λ_p = 524 nm	12.5 (4.7)	15.6 (5.85)	18.7 (7)	8%		28%	57%		80%	9%		27%	kHz
		E_e = 34.6 µW/cm^2, λ_p = 640 nm	13.1 (4.9)	16.4 (6.15)	19.7 (7.4)	5%		21%	0%		12%	84%		105%	kHz
R_e	Irradiance responsivity (Note 10)	λ_p = 470 nm		331 (124)		61%		84%	22%		43%	0%		6%	Hz/ (µW / cm^2)
		λ_p = 524 nm		386 (145)		8%		28%	57%		80%	9%		27%	Hz/ (µW / cm^2)
		λ_p = 640 nm		474 (178)		5%		21%	0%		12%	84%		105%	Hz/ (µW / cm^2)
	Saturation irradiance (Note 11)	λ_p = 470 nm		1813 (4839)			––			––			––		µW / cm^2
		λ_p = 524 nm		1554 (4138)			––			––			––		µW / cm^2
		λ_p = 640 nm		1266 (3371)			––			––			––		µW / cm^2
f_D	Dark frequency	E_e = 0		2	10		2	10		2	10		2	10	Hz

4. Dimensions of SG90 Servo Motor

Weight (g)	9
Torque (kg)	1.5
Speed(Sec/60deg)	0.09
A(mm)	30
B(mm)	23
C(mm)	27
D(mm)	12
E(mm)	33
F(mm)	16



5. Arduino IDE

The Arduino Integrated Development Environment (IDE) is a cross platform application (for Windows, Linux, MacOS) that is written in functions from C and C++. It is used to write and upload programs to Arduino compatible boards, but also, with the help of third-party cores, other vendor development boards.

The source code for the IDE is released under the GNU, General Public License, version 2. The Arduino IDE supports the languages C and C++ using special rules of code structuring. The Arduino IDE supplies a software library from the wiring 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 program with the 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. By default, *avrdude* is used as the uploading tool to flash the user code onto official Arduino board.

Programs written using Arduino Software (IDE) are called sketches. These sketches are written in the text editor and are saved with the file extension .ino. The editor has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino Software (IDE), including complete error messages and other information. The bottom righthand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

commands are found within the five menus: File, Edit, Sketch, Tools, Help. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.

File

- New Creates a new instance of the editor, with the bare minimum structure of a sketch already in place.
- Open Allows to load a sketch file browsing through the computer drives and folders.
- Open Recent Provides a short list of the most recent sketches, ready to be opened.
- Sketchbook Shows the current sketches within the sketchbook folder structure; clicking on any name opens the corresponding sketch in a new editor instance.
- Examples Any example provided by the Arduino Software (IDE) or library shows up in this menu item. All the examples are structured in a tree that allows easy access by topic or library.
- Close Closes the instance of the Arduino Software from which it is clicked.
- Save Saves the sketch with the current name. If the file hasn't been named before, a name will be provided in a "Save as.." window.
- Save as... Allows to save the current sketch with a different name.
- Page Setup It shows the Page Setup window for printing.
- Print Sends the current sketch to the printer according to the settings defined in Page Setup.

- Preferences Opens the Preferences window where some settings of the IDE may be customized, as the language of the IDE interface.
- Quit Closes all IDE windows. The same sketches open when Quit was chosen will be automatically reopened the next time you start the IDE.

Edit

- Undo/Redo Goes back of one or more steps you did while editing; when you go back, you may go forward with Redo.
- Cut Removes the selected text from the editor and places it into the clipboard.
- Copy Duplicates the selected text in the editor and places it into the clipboard.
- Copy for Forum Copies the code of your sketch to the clipboard in a form suitable for posting to the forum, complete with syntax coloring.
- Copy as HTML Copies the code of your sketch to the clipboard as HTML, suitable for embedding in web pages.
- Paste Puts the contents of the clipboard at the cursor position, in the editor.
- Select All Selects and highlights the whole content of the editor.
- Comment/Uncomment Puts or removes the // comment marker at the beginning of each selected line.
- Increase/Decrease Indent Adds or subtracts a space at the beginning of each selected line, moving the text one space on the right or eliminating a space at the beginning.
- Find Opens the Find and Replace window where you can specify text to search inside the current sketch according to several options.
- Find Next Highlights the next occurrence - if any - of the string specified as the search item in the Find window, relative to the cursor position.
- Find Previous Highlights the previous occurrence - if any - of the string specified as the search item in the Find window relative to the cursor position.

Sketch

- Verify/Compile Checks your sketch for errors compiling it; it will report memory usage for code and variables in the console area.
- Upload Compiles and loads the binary file onto the configured board through the configured Port.
- Upload Using Programmer This will overwrite the bootloader on the board; you will need to use Tools > Burn Bootloader to restore it and be able to Upload to USB serial port again. However, it allows you to use the full capacity of the Flash memory for your sketch. Please note that this command will NOT burn the fuses. To do so a Tools -> Burn Bootloader command must be executed.
- Export Compiled Binary Saves a .hex file that may be kept as archive or sent to the board using other tools.
- Show Sketch Folder Opens the current sketch folder.
- Include Library Adds a library to your sketch by inserting #include statements at the start of your code.

Tools

- Auto Format This formats your code nicely: i.e. indents it so that opening and closing curly braces line up, and that the statements inside curly braces are indented more.
- Archive Sketch Archives a copy of the current sketch in .zip format. The archive is placed in the same directory as the sketch.
- Fix Encoding & Reload Fixes possible discrepancies between the editor char map encoding and other operating systems char maps.
- Serial Monitor Opens the serial monitor window and initiates the exchange of data with any connected board on the currently selected Port. This usually resets the board, if the board supports Reset over serial port opening.

- Port This menu contains all the serial devices (real or virtual) on your machine. It should automatically refresh every time you open the top-level tools menu.
- Programmer For selecting a hardware programmer when programming a board or chip and not using the onboard USB-serial connection.
- Burn Bootloader - The items in this menu allow you to burn bootloader onto the microcontroller on an Arduino board. This is not required for normal use of an Arduino board but is useful if you purchase a new ATmega microcontroller (which normally come without a bootloader). Ensure that you've selected the correct board from the Boards menu before burning the bootloader on the target board. This command also set the right fuses.

The Arduino Software (IDE) uses the concept of a sketchbook: a standard place to store your programs (or sketches). The sketches in your sketchbook can be opened from the File > Sketchbook menu or from the Open button on the toolbar. The first time you run the Arduino software, it will automatically create a directory for your sketchbook. You can view or change the location of the sketchbook location from with the Preferences dialog.

Beginning with version 1.0, files are saved with a .ino file extension. Previous versions use the .pde extension. You may still open .pde named files in version 1.0 and later, the software will automatically rename the extension to .ino.

Tabs, Multiple Files, and Compilation

Allows you to manage sketches with more than one file (each of which appears in its own tab). These can be normal Arduino code files (no visible extension), C files (.c extension), C++ files (.cpp), or header files (.h).

Before compiling the sketch, all the normal Arduino code files of the sketch (.ino, .pde) are concatenated into a single file following the order the tabs are shown in. The other file types are left as is.

Uploading

Before uploading your sketch, you need to select the correct items from the Tools > Board and Tools > Port menus. Boards are described below. On the Mac, the serial port is probably something like /dev/tty.usbmodem241 (for an UNO or Mega2560 or Leonardo) or /dev/tty.usbserial-1B1 (for a Duemilanove or earlier USB board), or /dev/tty.USA19QW1b1P1.1 (for a serial board connected with a Keyspan USB-to-Serial adapter). On Windows, it's probably COM1 or COM2 (for a serial board) or COM4, COM5, COM7, or higher (for a USB board) - to find out, you look for USB serial device in the ports section of the Windows Device Manager. On Linux, it should be /dev/ttyACMx , /dev/ttyUSBx or similar. Once you've selected the correct serial port and board, press the upload button in the toolbar or select the Upload item from the Sketch menu. Current Arduino boards will reset automatically and begin the upload. With older boards (pre-Diecimila) that lack auto-reset, you'll need to press the reset button on the board just before starting the upload. On most boards, you'll see the RX and TX LEDs blink as the sketch is uploaded. The Arduino Software (IDE) will display a message when the upload is complete, or show an error.

When you upload a sketch, you're using the Arduino bootloader, a small program that has been loaded on to the microcontroller on your board. It allows you to upload code without using any additional hardware. The bootloader is active for a few seconds when the board resets; then it starts whichever sketch was most recently uploaded to the microcontroller. The bootloader will blink the on-board (pin 13) LED when it starts (i.e. when the board resets).

Libraries

Libraries provide extra functionality for use in sketches, e.g. working with hardware or manipulating data. To use a library in a sketch, select it from the Sketch > Import Library menu. This will insert one or more #include statements at the top of the sketch and compile the library with your sketch. Because libraries are uploaded to the board with your sketch, they increase the amount of space it takes up. If a sketch no longer needs a library, simply delete its #include statements from the top of your code.

Some libraries are included with the Arduino software. Others can be downloaded from a variety of sources or through the Library Manager. Starting with version 1.0.5 of the IDE, you do can import a library from a zip file and use it in an open sketch.

Third-Party Hardware

Support for third-party hardware can be added to the hardware directory of your sketchbook directory. Platforms installed there may include board definitions (which appear in the board menu), core libraries, bootloaders, and programmer definitions. To install, create the hardware directory, then unzip the third-party platform into its own sub-directory. (Don't use "arduino" as the sub-directory name or you'll override the built-in Arduino platform.) To uninstall, simply delete its directory.

Serial Monitor

This displays serial sent from the Arduino board over USB or serial connector. To send data to the board, enter text and click on the "send" button or press enter. Choose the baud rate from the drop-down menu that matches the rate passed to Serial.begin in your sketch. Note that on Windows, Mac or Linux the board will reset (it will rerun your sketch) when you connect with the serial monitor. Please note that the Serial Monitor does not process control characters; if your sketch needs a complete management of the serial communication with control characters, you can use an external terminal program and connect it to the COM port assigned to your Arduino board.

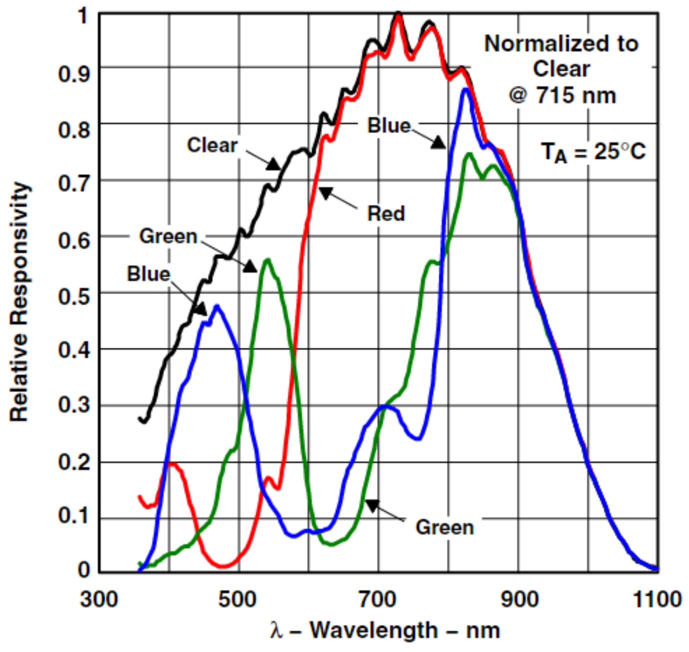
Preferences

Some preferences can be set in the preferences dialog (found under the Arduino menu on the Mac, or File on Windows and Linux). The rest can be found in the preferences file, whose location is shown in the preference dialog.

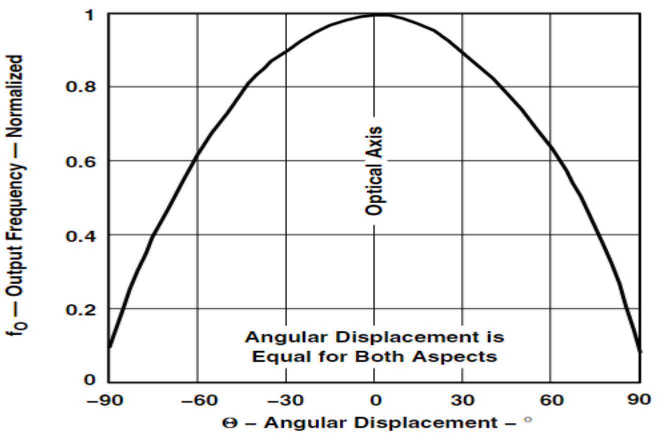
Boards

The board selection has two effects: it sets the parameters (e.g. CPU speed and baud rate) used when compiling and uploading sketches; and sets and the file and fuse settings used by the burn bootloader command.

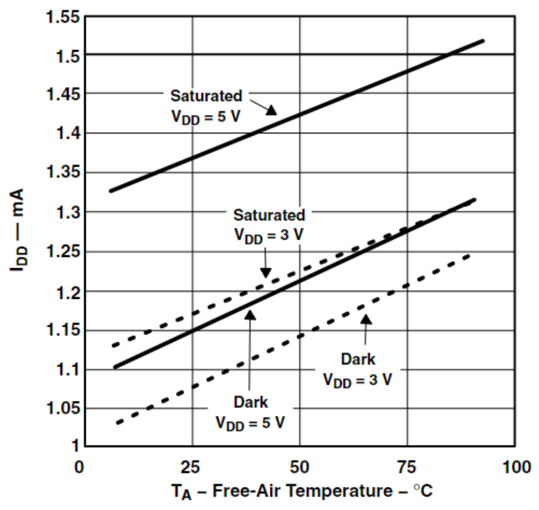
6. Photodiode Spectral Responsivity of TCS3200



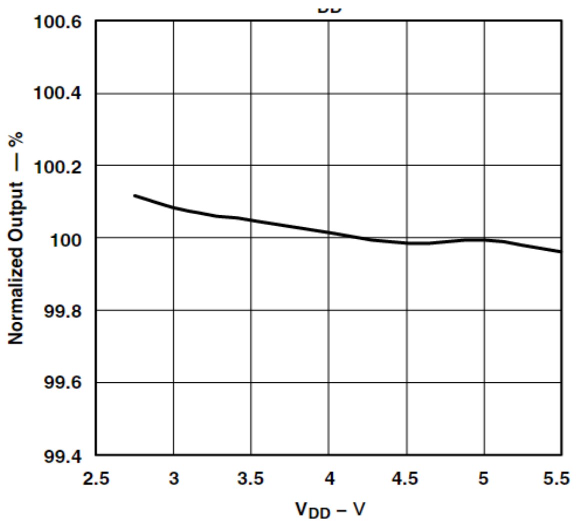
7. Normalized Output Frequency vs Angular Displacement of TCS3200



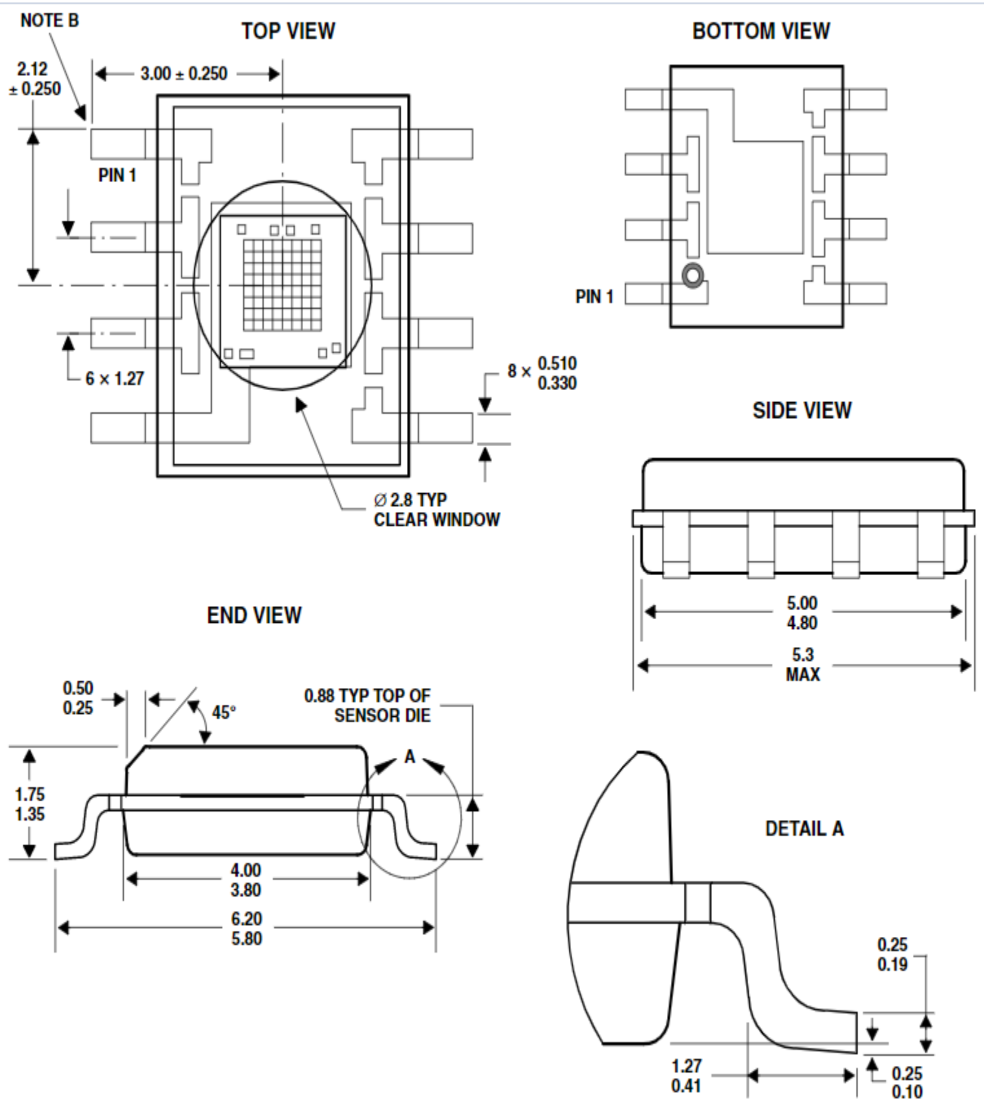
8. IDD VS VDD VS TEMPERATURE OF TCS3200



9. Normalized Output vs VDD



10. Dimensions of TCS3200



REFERENCES

1. "Automatic Colour Sorting Machine Using Arduino Mega Microcontroller", Aye Myat Myat Myo1, Zar Chi Soe2, International Journal of Latest Technology in Engineering, Management and Applied Science Volume VIII, Issue VIII, August 2019 - An implementation of Colour Sorting Machine using Arduino Mega. The information gained are block diagram, circuit diagram and working of the project.

2. "Design and Development of an Automatic Colour Sorting Machine on Belt Conveyor", Aung Thike, Zin Zin Moe San, Dr. Zaw Min Of, International Journal of Science and Engineering Applications Volume 8 – Issue 07, 176-179, 2019, ISSN:-2319-7560 - implementation using conveyer belt system. Information gained include working of colour sensor, servo motors and applications of this project.

3. "Programming Arduino Getting Started with Sketches", Book by Simon Monk - A book on Arduino programming in C using Arduino IDE. An essential resource while writing the code for project

4. Data Sheets of different sources and ICs