

A PROJECT REPORT ON

**Low Cost Self-Activating Touchless Hand Sanitizer Dispensing with
Battery Imposed System**

Submitted to

GONDWANA UNIVERSITY, GADCHIROLI

by

**SHIVANI N. KODURWAR
SHUBHAM P. KAJALE
PRANAV B. KHATALE**



**DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION
ENGINEERING**

GOVERNMENT COLLEGE OF ENGINEERING.

CHANDRAPUR – 442403

2020-21

A PROJECT REPORT ON

**Low Cost Self-Activating Touchless Hand Sanitizer Dispensing with
Battery Imposed System**

Submitted to

GONDWANA UNIVERSITY, GADCHIROLI

by

**SHIVANI N. KODURWAR
SHUBHAM P. KAJALE
PRANAV B. KHATALE**

under the guidance of

Prof. H. M. RAZA



**DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION
ENGINEERING**

GOVERNMENT COLLEGE OF ENGINEERING.

CHANDRAPUR - 442403

2020-21

CERTIFICATE

This is to certify that the project entitled “**Low Cost Self-Activating Touchless Hand Sanitizer Dispensing with Battery Imposed System**” has been carried out by the team under my guidance of the Third-Year Bachelor of Engineering in Electronics and Telecommunication Engineering of Gondwana University, Gadchiroli during academic year 2020-21.

Team:

- 1. Shivani N. Kodurwar**
- 2. Shubham P. Kajale**
- 3. Pranav B. Khatale**

Date:

Place: Chandrapur

Prof. H. M. RAZA

**Guide
&
Head of the Department**

Dr. S. G. AKOJWAR

**Principal
(Government College of Engineering. Chandrapur)**

**GOVERNMENT COLLEGE OF ENGINEERING.
CHANDRAPUR - 442403**

Contents

Certificate	ii
Contents	iii
List of Figures	v

1 Introduction	1
2 REVIEW OF LITERATURE	2
2.1 Our Finding.....	2
3 Material & Methodology	3
3.1 Ultrasonic Sensor.....	4
Description.....	5
Technical Specifications.....	6
Sensor's Working.....	7
Pinouts.....	8
Working Principle.....	8
Sequence Chart.....	9
3.2 Servo Motor SG90.....	10
Description.....	10
Technical Specifications.....	10
Electrical Connections.....	12
Construction & Working.....	13
3.3 Arduino UNO R3 Board.....	15
Description.....	16
Technical Specifications.....	17
Arduino UNO - ATmega328p Pin Mapping.....	18

Main Chip.....	18
Chip Communication.....	26
Reset Button.....	26
Voltage Regulator.....	27
Memory.....	27
TX & RX LED's.....	27
Power LED Indicator.....	27
3.4 USB Cable.....	27
3.5 9V Battery, Connector & Power Jack.....	28
Software IDE Description	30
4.1 Writing Sketches.....	30
File.....	31
Edit.....	32
Sketch.....	33
Tools.....	33
Help.....	34
4.2 Sketchbook.....	34
4.3 Tabs, Multiple Files, and Compilation.....	35
4.4 Uploading.....	35
4.5 Libraries.....	36
4.6 Third Party Hardware.....	36
4.7 Serial Monitor.....	36
4.8 Preferences.....	37
4.9 Language Support.....	37
4.10 Boards.....	37
Advantages & Applications	38
5.1 Advantages.....	38
5.2 Applications.....	38
Result & Conclusion	40
6.1 Result.....	40
6.2 Conclusion.....	40
References	41
Appendix-A	43

List of Figure

Fig. No.	Figure name
2.1	Automatic Soap Dispensers: Our Finding.....2
3.0	Block diagram of Low Cost Self-Activating Touchless Hand Sanitizer....3
	Dispensing with Battery Imposed System
3.1	Ultrasonic Sensor HCSR04.....4
3.2	Servo Motor SG90.....10
3.3	Arduino UNO R3.....15
3.4	USB Cable.....27
3.5	Power Supply.....28
4.0	Arduino IDE.....30

CHAPTER 1

1. INTRODUCTION

The corona virus disease is a major problem in the future world. Presently there is no medicine or vaccine made available in the India for general public. As there is a severe attack in this world, the people are suffering from the corona disease. The COVID-19 disease is not a simple virus attack, it makes severe to the human by infecting the respiratory system. The virus disease is heavily spreading in the world, as the nations are trying to monitor and maintain the spread of corona in the nation and other nations. The world is suffering a lot due to this corona virus.

There is a strict evaluation everywhere to control the corona disease and spread to the nation. The hospital and the nurse people are suffering to cure the affected people and stop spreading the virus to the neighboring people.

The mask and the sanitizer are provided everywhere to protect the people from spreading the virus and to kill the virus from the human hands. The virus is spreading from the human hands and mouth saliva. The mouth spread is controlled with the mask cloth and the human hand is controlled by the hand sanitizer. The hand touch bottle while pressing the dispenser usage also spreads from human to human. There should be an automatic hand wash sanitizer dispenser, to control and maintain the spread from human to human. As there is an impact in using the hand wash sanitation by foot or by pressing the sanitizer bottle used to have a spread of the virus disease from one human to another.

A long press is made with the footer, such that the mechanical stress is made on the instrument. The mechanical stress made, is forced to spray out the Sanitizer liquid. The human at aged people is unable to use this system as there is mechanical stress and there is a sudden liquid force coming from the sanitizer bottle.

The Easy Non-Contact Automatic Hand Sanitizer Dispenser or Automatic Soap Dispenser with Arduino UNO, it has the Atmega328p microcontroller to control the sanitizer liquid with the help of a Servo motor. This is used to power up the system by the external power supply of 9V battery or through computer.

CHAPTER 2

2. REVIEW OF LITERATURE

The Easy Non-Contact Automatic Hand Sanitizer Dispenser or Automatic Soap Dispenser with Arduino, is one of the right solutions to this problem to solve as it has the Arduino microcontroller to control the sanitizer liquid amount flow with the help of a Servo motor. This is used to power up the system by the external power supply of 9V battery or through computer USB cable. This method is good to use and the drawback is the battery replacement for the usage of the system.

2.1 Our Finding:

Several foot-operated hand sanitization stations are installed at different government offices, hospitals, banks, railway stations & other public places but maximum of them are foot operated. A long press is made with the footer, such that the mechanical stress is made on the instrument. The mechanical stress made, is forced to spray out the Hand Sanitizer Liquid. The human at aged people is unable to use this system as there is mechanical stress and there is a sudden liquid force coming from the sanitizer bottle. So, a touchless solution is required to stop the virus spread due to the foot-operated hand sanitization station.

CHAPTER 3

3. MATERIAL & METHODOLOGY

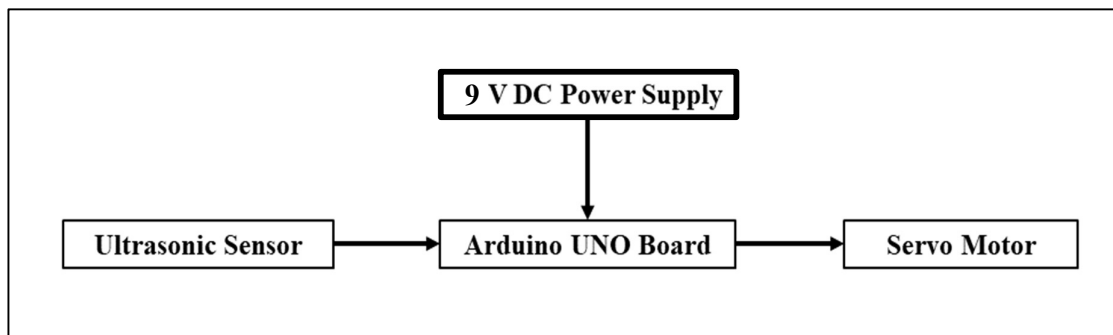


Figure 3.1: Block Diagram of Low Cost Self-Activating Touchless Hand Sanitizer Dispensing with Battery Imposed System

Working:

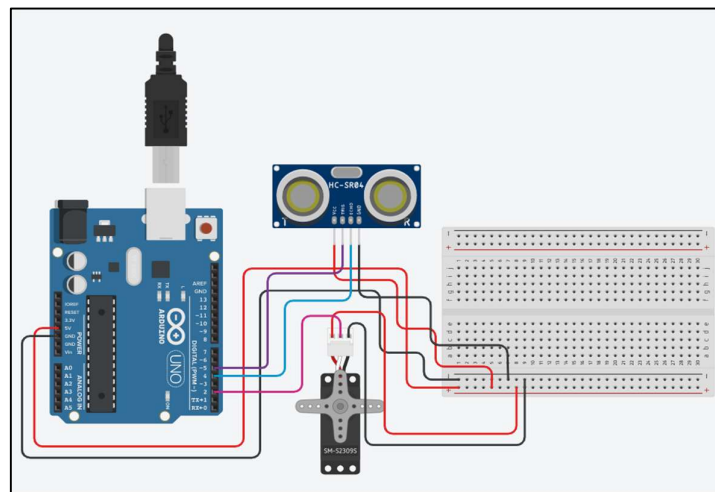


Figure 3.2: Circuit Diagram of Low Cost Self-Activating Touchless Hand Sanitizer Dispensing with Battery Imposed System

Low Cost Self-Activating Touchless Hand Sanitizing Dispensing with Battery Imposed System

The Ultrasonic Sensor sends the distance information to Arduino board & the board sends the control signal to servo motor to get activated for rotation. So, the required amount of hand sanitizer is dispensed through the sanitizer bottle. Such that the consumer can have only 2 to 3 mL of sanitizer liquid for sanitation. The Arduino code which is meeting the expected need is uploaded to the board to complete the project. So, it works like the normal contactless automatic machine. The human gets the limited sanitizer liquid for sanitation in hand, to sanitize the hands.

Components Used:

- 1) Ultrasonic Sensor HCSR04
- 2) Servo Motor SG90
- 3) Arduino UNO R3 Board
- 4) USB Cable
- 5) 9V Battery
- 6) 9V Battery Connector with 2.1 mm Center positive Power Jack
- 7) Sanitizer Bottle having Dispenser
- 8) Breadboard
- 9) Computer with Arduino IDE
- 10) Nylon String
- 11) Hot Glue
- 12) Plastic Insulation Tape
- 13) Thick Cardboard (10 cm * 10 cm)

3.1 Ultrasonic Sensor HCSR04:

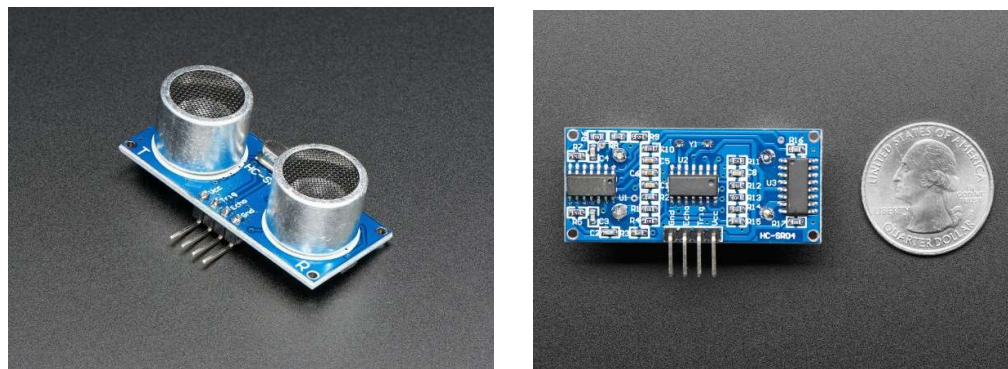


Figure 3.3 Ultrasonic Sensor HCSR04

3.1.1 Description:

HC-SR04 ultrasonic distance sensor. This economical sensor provides 2cm to 400cm of non-contact measurement functionality with a ranging accuracy that can reach up to 3mm. Each HC-SR04 module includes an ultrasonic transmitter, a receiver and a control circuit.

There are only four pins that you need to worry about on the HC-SR04: VCC (Power), Trig (Trigger), Echo (Receive), and GND (Ground). You will find this sensor very easy to set up and use for your next range-finding project! This sensor has additional control circuitry that can prevent inconsistent "bouncy" data depending on the application. The HC-SR04 ultrasonic sensor uses sonar to determine distance to an object like bats do. It offers excellent non-contact range detection with high accuracy and stable readings in an easy-to-use package. It comes complete with ultrasonic transmitter and receiver modules. Ultrasonic ranging module HC - SR04 provides 2cm - 700cm non-contact measurement function, the ranging accuracy can reach to 3mm. Ensured stable signal within 5m, gradually faded signal outside 5m till disappearing at 7m position. The module includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:

1. Using IO trigger for at least 10us high level signal
2. The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
3. If the signal back, through high level, time of high output IO duration is the time from sending ultrasonic to returning.

Test distance = (high level time \times velocity of sound (340M/S) / 2.

These ubiquitous sensors are really common in robotics projects, but they can also be used for automation, interactive art and motion sensing. They work at about 2cm to 400cm away, but we think 10cm-250cm will get you the best results. The HC-SR04 sensors are fast, fairly easy to use, and low cost. They do require powering from 5V for best results: connect GND (ground) and VCC to 5V power. While the Trig signal can be 3V or 5V, the 'return' Echo signal is 5V logic.

3.1.2 Technical Specifications:

- Power & Logic Voltage: DC 5V
- Current during measurement: 15mA
- Ultrasonic Frequency: 40 kHz
- Effectual Angle: 15°
- Trigger Input Signal: 10uS high TTL pulse
- Sensor dimensions (excluding header): 45.5 x 20 x 15.5mm
- Weight: 8.7g
- Ranging Distance: 2cm - 4m//1" – 13ft
- Quiescent Current: <2mA
- Working Current: 15mA
- Resolution: 0.3 cm
- Measuring Angle: 30 degree
- Trigger Input Pulse width: 10uS TTL Pulse
- Echo Output Signal: Input TTL lever signal and the range in proportion

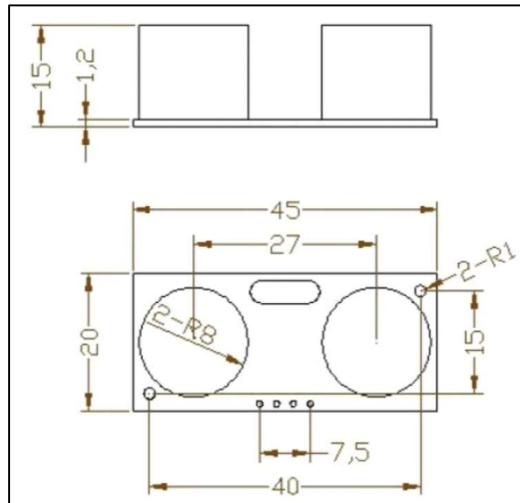


Figure 3.4 Dimensions of HCSR04

3.1.3 Sensor's Working:

The ultrasonic sensor uses sonar to determine the distance to an object. Here's what happens:

1. The transmitter (trig pin) sends a signal: a high-frequency sound.
2. When the signal finds an object, it is reflected and...
3. ... the transmitter (echo pin) receives it.

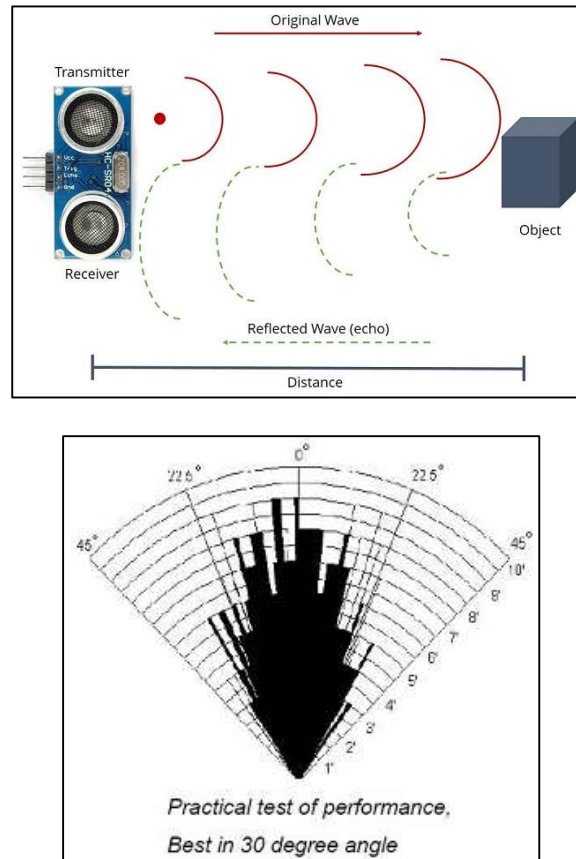


Figure 3.5 HCSR04 Sensor working range

The time between the transmission and reception of the signal allows us to calculate the distance to an object. This is possible because we know the sound's velocity in the air.

3.1.4 Pinouts:



Figure 3.6 Pinout of HCSR04 Sensor

The HC-SR04 sensors are simple to use. Let's take a look!

- **Vcc** - Power pin - this sensor requires +5V power for best results.
- **Trig** - Signal can be 3V or 5V [Trigger (INPUT)]
- **Echo** - "Return" signal is 5V logic - the sensor comes with two 10k resistors to use as a divider to convert the 5V logic level to a safe 2.5V that you can read with your 3V device. [Echo (OUTPUT)]
- **Gnd** - Ground pin.

3.1.5 Working Principle:

The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8-cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion. You can calculate the range through the time interval between sending trigger signal and receiving echo signal.

Formula: $\mu\text{S} / 58 = \text{centimetres}$ or $\mu\text{S} / 148 = \text{inch}$; or: the range = high level time * velocity $(340\text{M/S}) / 2$;

we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.

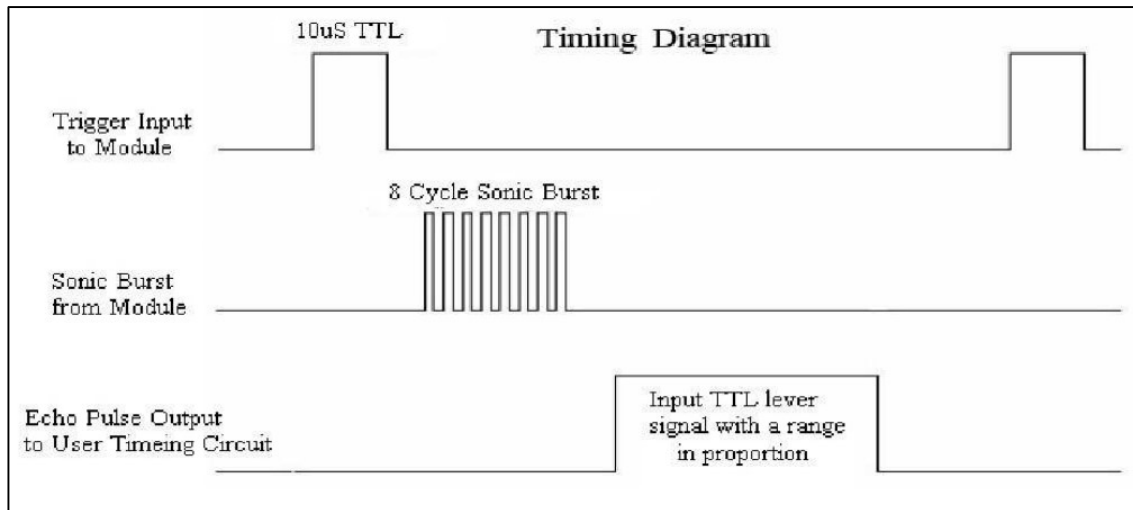


Figure 3.7 Timing Diagram (a)

3.1.6 Sequence Chart:

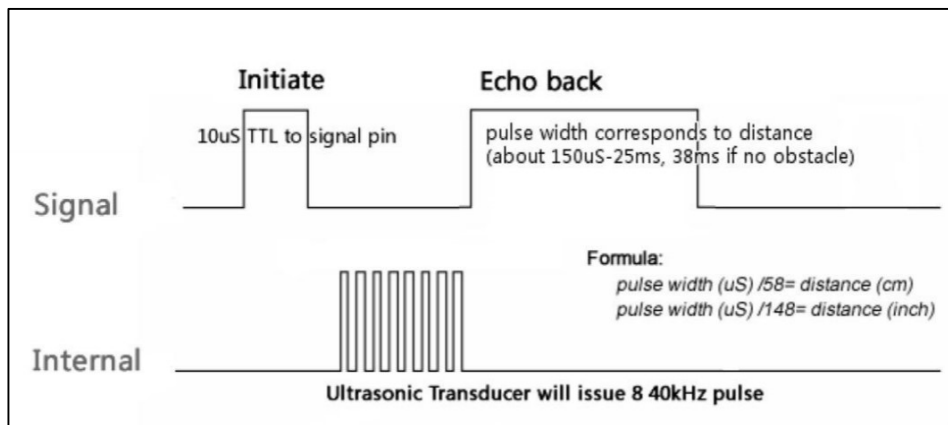


Figure 3.8 Timing Diagram (b)

A short ultrasonic pulse is transmitted at the time 0, reflected by an object. The sensor receives this signal and converts it to an electric signal. The next pulse can be transmitted when the echo is faded away. This time period is called cycle period. The recommended cycle period should be no less than 50 ms. If a 10 μs width trigger pulse is sent to the signal pin, the Ultrasonic module will output eight 40 kHz ultrasonic signal and detect the echo back. The measured distance is proportional to the echo pulse width and can be calculated by the formula above. If no obstacle is detected, the output pin will give a 38ms high level signal.

3.2 Servo Motor SG90:



Figure 3.9 Servo Motor

3.2.1 Description:

A servomechanism (servo) can refer to quite a few different machines that have been around longer than most may realize. Essentially, a servo is any motor-driven system with a feedback element built in. Servos are found everywhere from heavy machinery, to power steering in vehicles, to robotics and a wide variety of electronics.

Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but smaller. Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.

3.2.2 Technical Specifications:

- Weight: 9 g
- Stall torque: 1.8 kgf·cm
- Operating speed: 0.1 s/60 degree
- Operating voltage: 4.8 V (~5V)
- Dead band width: 10 μ s
- Temperature range: 0 °C – 55 °C

Low Cost Self-Activating Touchless Hand Sanitizing Dispensing with Battery Imposed System

- Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is all the way to the left. ms pulse) is all the way to the right, ""-90" (~1ms pulse) is all the way to the left.
- Modulation: Analog
- Torque: 4.8V: 25.0 oz-in (1.80 kg-cm)
- Speed: 4.8V: 0.10 sec/60°
- Weight: 0.32 oz (9.0 g)
- Motor Type: 3-pole
- Gear Type: Plastic
- Rotation/Support: Bushing
- Rotational Range: 180°
- Pulse Cycle: ca. 20 ms
- Pulse Width: 500-2400 μ s
- Dimensions:
Length: 0.91 in (23.1 mm)
Width: 0.48 in (12.2 mm)
Height: 1.14 in (29.0 mm)

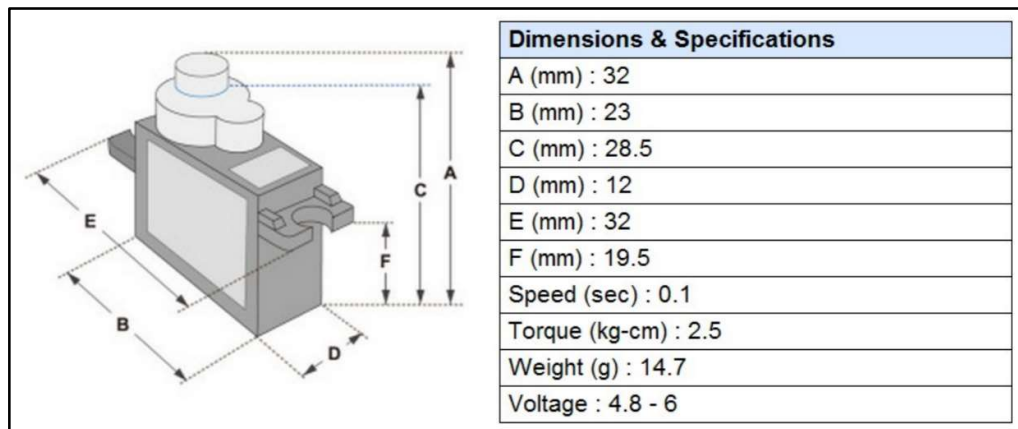


Figure 3.10 Dimensions of Servo Motor

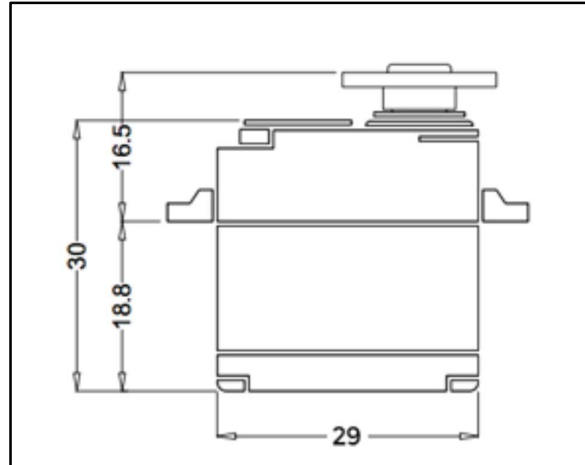


Figure 3.11 Dimensions & Specifications

3.2.3 Electrical Connections

Most hobby servos use a standard type of 3-pin plug, with the same control signalling, which makes RC servos reasonably interchangeable. The connector is a female, 3-pin, 0.1" pitch header. One thing that can be confusing is that the wiring colour code isn't always consistent there are several colour codes at play. The good news is that the pins are usually in the same order, just that the colours are different. The table below summarizes common colour schemes. A useful mnemonic is that the drabest colour (black or brown) is usually ground, and red is usually the power supply.

Pin Number	Signal Name	Colour Scheme 1 (Futaba)	Colour Scheme 2 (JR)	Colour Scheme 3 (Hitec)
1	Ground	Black	Brown	Black
2	Power Supply	Red	Red	Red or Brown
3	Control Signal	White	Orange	Yellow or White

Table 3.1: Servo Connection colour coding

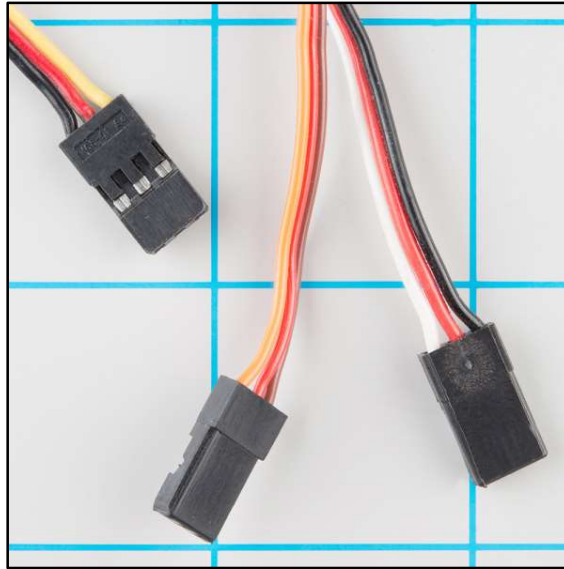


Figure 3.12 Servo Cables

3.2.4 Construction & Working:

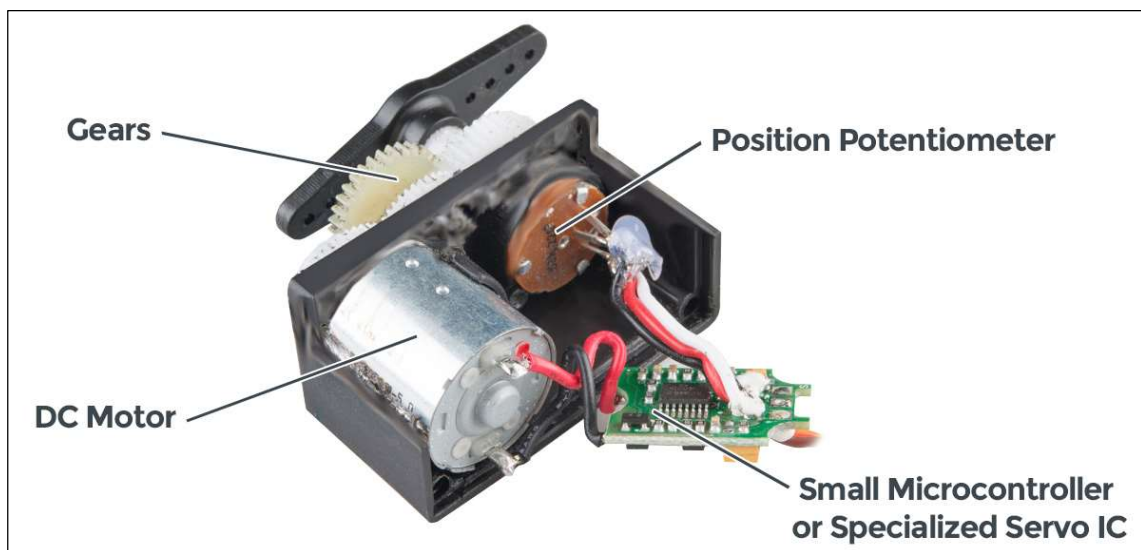


Figure 3.13 Servo Motor internal structure



Figure 3.14 Servo Motor Horns

There are three core components in servo SG90: a DC motor, a controller circuit, and a potentiometer or similar feedback mechanism. The DC motor is attached to a gearbox and output/drive shaft to increase the speed and torque of the motor. The DC motor drives the output shaft. The controller circuit interprets signals sent by the controller, and the potentiometer acts as the feedback for the controller circuit to monitor the position of the output shaft. Nearly all hobby servos have a standard three-pin, 0.1"-spaced connector to power and control the servo. The color coding can vary between brands, but the pins are almost universally in the same order. When combined together, you can power and control the direction, speed and position of the output shaft with just three wires.

In order to move a servo to a position along its movement arc, or, in the case of continuous rotation servos the speed and direction of the motor, the controller needs to send a precisely timed signal for the servo to interpret. Typical hobby servos expect to see a pulse every 20ms, and the width of this signal determines the position. This width is usually between one and two milliseconds. This type of signal control is frequently referred to as Pulse Width Modulation, abbreviated as PWM. A servo controller will normally be a dedicated piece of hardware that can take inputs from other components like a joystick, potentiometer or sensor feedback to set the control signal for the servo. Other control options include using the PWM-capable pins on a microcontroller to send that signal directly to the servo.

Depending on the size and torque output of your servo the input voltage will vary, but most hobby servos will work fine with 5V from your preferred microcontroller or battery

circuit. More important than voltage is the current draw a servo can pull while moving and with a load attached. When unloaded, a common hobby servo can pull as little as 10mA, but larger servos under load can pull in excess of an Ampere or more. It is important to check the specifications of the servo you intend to use to make sure your power supply has the proper voltage range and can deliver enough current to move the servo with your load attached.

The third pin of the servo connector carries the control signal, used to tell the motor where to go. This control signal is a specific type of pulse train. The pulses occur at a 20 mSec (50 Hz) interval, and vary between 1 and 2 ms in width. The Pulse Width Modulation hardware available on a microcontroller is a great way to generate servo control signals. Common servos rotate over a range of 90° as the pulses vary between 1 and 2 mSec they should be at the center of their mechanical range when the pulse is 1.5 ms.

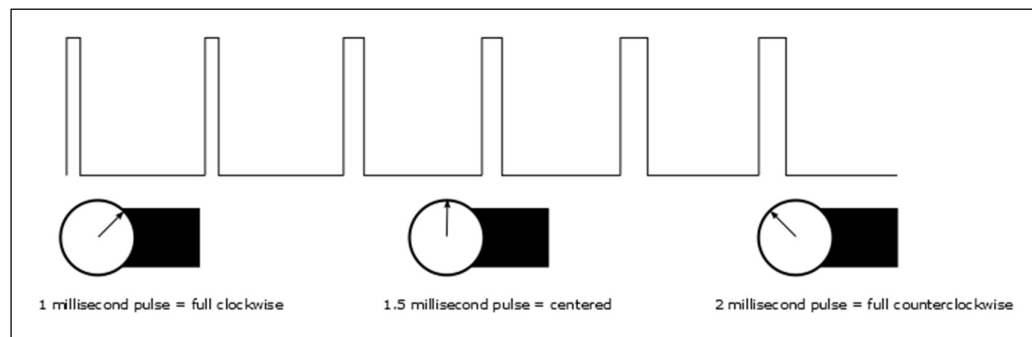


Figure 3.15 Servo Motor Pulse to Position relation

3.3 Arduino UNO R3 Board



Figure 3.16 Arduino UNO R3

3.3.1 Description:

Arduino UNO is an open-source board from Arduino platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator (CSTCE16M0V53-R0), a USB connection, a power jack, an ICSP header and a reset button. The Uno R3 also adds SDA and SCL pins next to the AREF. In addition, there are two new pins placed near the RESET pin. One is the IOREF that allows the shields to adapt to the voltage provided from the board. The other is not connected and is reserved for future purposes. The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter. Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program. Finally, Arduino provides a standard form factor that breaks out the functions of the micro-controller into a more accessible package. "Uno" means one in Italian and was chosen to mark the release of Arduino Software (IDE) 1.0. The Uno board and version 1.0 of Arduino Software (IDE) were the reference versions of Arduino, now evolved to newer releases. The Uno board is the first in a series of USB Arduino boards, and the reference model for the Arduino platform; for an extensive list of current, past or outdated boards. The Arduino Uno has a resettable polyfuse that protects user's computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed. It contains everything needed to support the microcontroller; user can simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started. User can tinker with Arduino Uno without worrying too much about doing something wrong, worst case scenario user can replace the chip for a few Indian rupees and start over again.

3.3.2 Technical Specifications:



Figure 3.17 Arduino Front & Back image

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25

Table 3.2 Technical Details

3.3.3 Arduino - ATmega328p Pin Mapping:

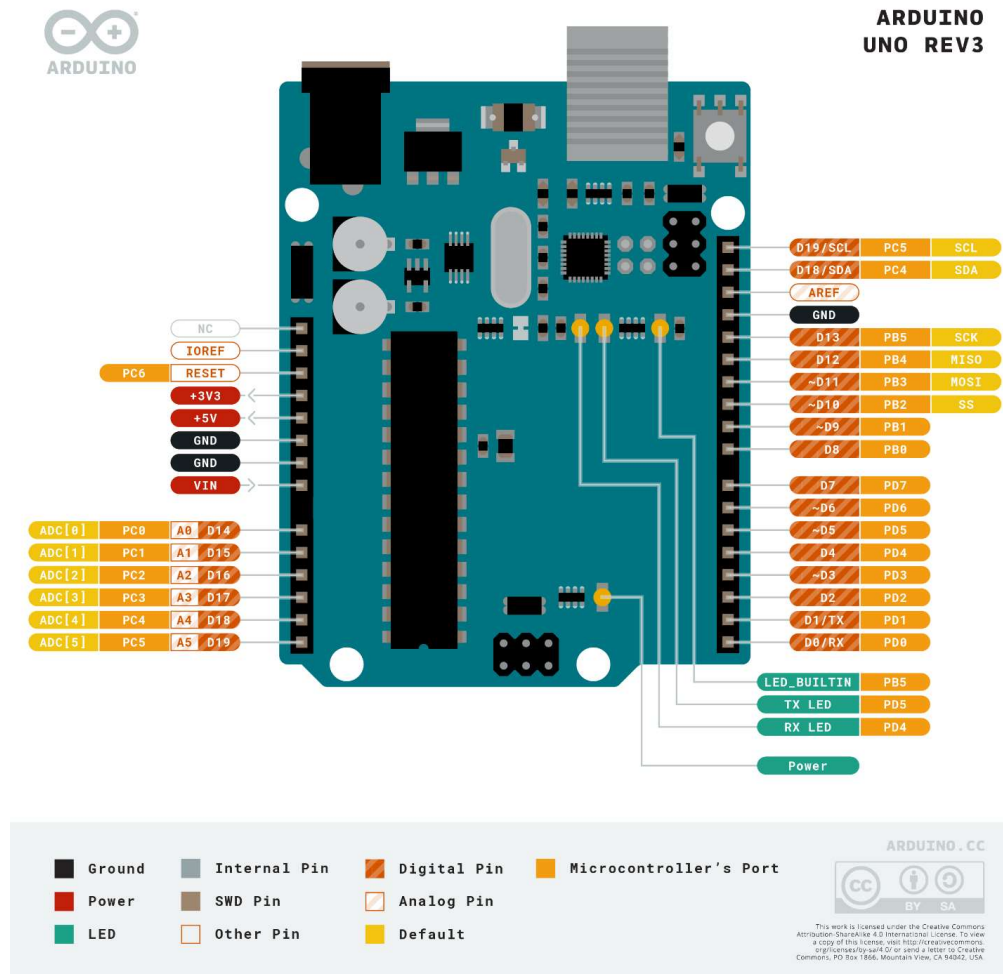


Figure 3.18 Arduino UNO Board Pin Mapping

3.3.3.1 Main Chip:

The black thing with all the metal legs is an IC, or Integrated Circuit. Think of it as the brains of our Arduino. The ATmega328 is a single-chip microcontroller created by Atmel in the MegaAVR family. It has a modified Harvard architecture 8-bit RISC processor core.

Each of the 14 digital 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

Low Cost Self-Activating Touchless Hand Sanitizing Dispensing with Battery Imposed System

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 Atmel 8-bit AVR RISC-based microcontroller combines 32 KB ISP flash memory with read-while-write capabilities, 1 KB EEPROM, 2 KB SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible timer / counters with compare modes, internal and external interrupts, serial programmable USART, a byte-oriented 2-wire serial interface, SPI serial port, 6-channel 10-bit A/D converter (8-channels in TQFP and QFN/MLF packages), programmable watchdog timer with internal oscillator, and five software selectable power saving modes. The device operates between 1.8-5.5 volts. The device achieves throughput approaching 1 MIPS per MHz

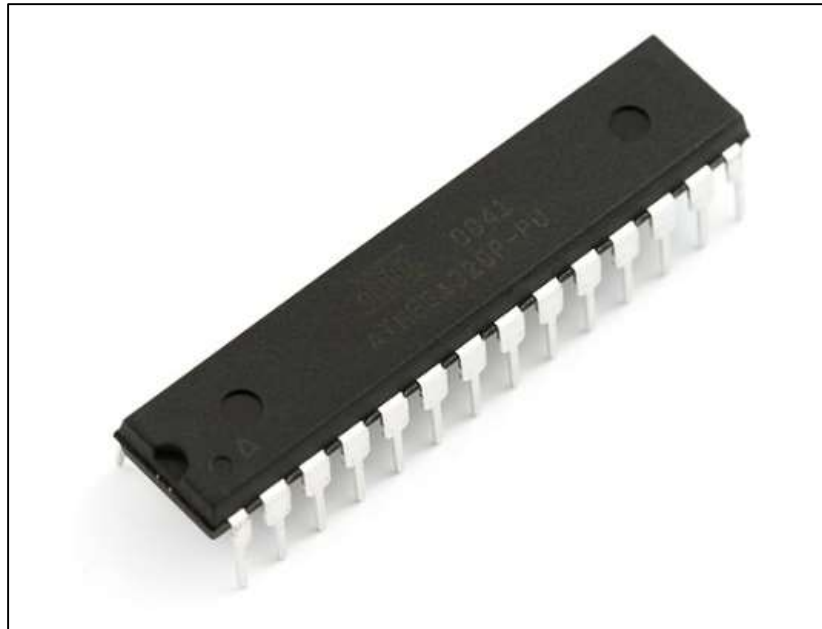


Figure 3.19 ATMega328p DIP chip

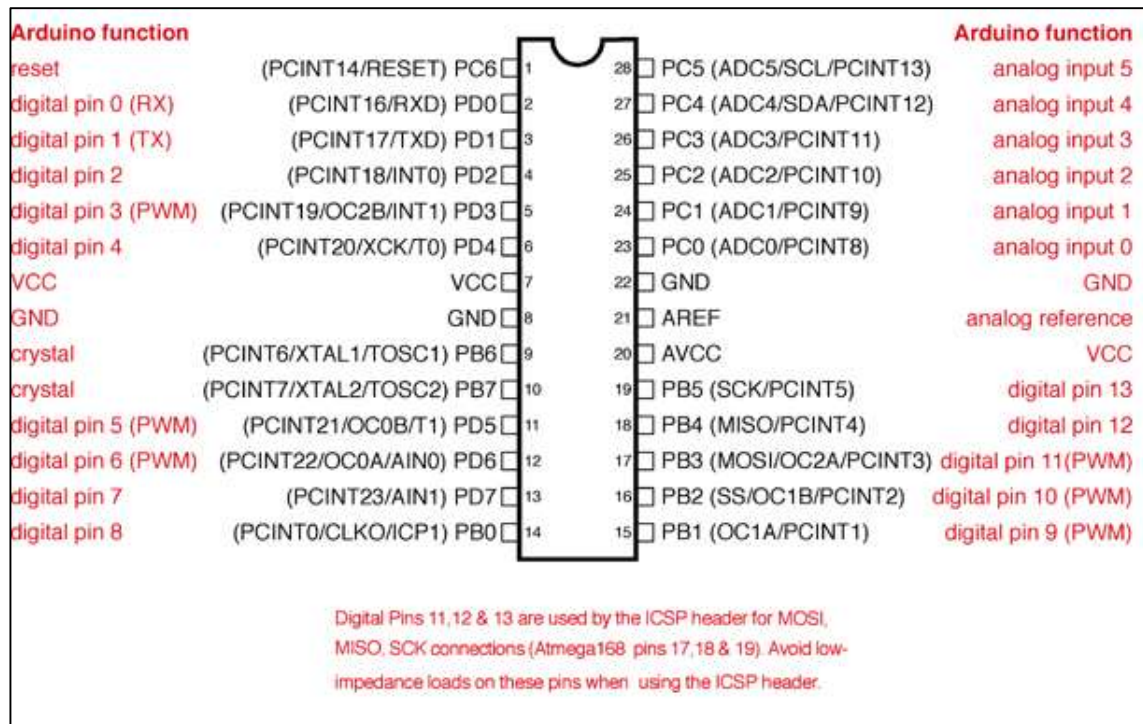


Figure 3.20 Pin Diagram of ATmega328p DIP chip

The ATmega328P is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega328P achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega328P provides the following features: 4K/8K/16K/32K bytes of In System Programmable Flash with Read-While-Write capabilities, 256/512/512/1K bytes EEPROM, 512/1K/1K/2K bytes SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, a byte-oriented 2-wire Serial Interface, an SPI serial port, a 6-channel 10-bit ADC (8 channels in TQFP and QFN/MLF packages), a programmable Watchdog Timer with internal Oscillator, and five software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, USART, 2-wire

Low Cost Self-Activating Touchless Hand Sanitizing Dispensing with Battery Imposed System

Serial Interface, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or hardware reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption.

The device is manufactured using Atmel's high-density non-volatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional non-volatile memory programmer, or by an On-chip Boot program running on the AVR core. The Boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega328P is a powerful microcontroller that provides a highly flexible and cost-effective solution to many embedded control applications.

The ATmega328P AVR is supported with a full suite of program and system development tools including: C Compilers, Macro Assemblers, Program Debugger/Simulators, In-Circuit Emulators, and Evaluation kits.

Parameter	Value
CPU type	8-bit AVR
Performance	20 MIPS at 20 MHz
Flash memory	32 KB
SRAM	2 KB
EEPROM	1 KB
Pin count	28 or 32 pin
Maximum operating frequency	20 MHz
Number of touch channels	16
Hardware QTouch Acquisition	No
Maximum I/O pins	23
External interrupts	2
USB Interface	No
USB Speed	—

Table 3.3 ATmega328p technical details**Port B (PB7:0) XTAL1/XTAL2/TOSC1/TOSC2:**

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running. Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit. Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier. If the Internal Calibrated RC Oscillator is used as chip clock source, PB7..6 is used as TOSC2..1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

Port C (PC5:0):

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC5..0 output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

PC6/RESET:

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C. If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a Reset.

Port D (PD7:0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

ALU:

Arithmetic Logic Unit The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format.

Status Register:

The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. Note that the Status Register is updated after all ALU

Low Cost Self-Activating Touchless Hand Sanitizing Dispensing with Battery Imposed System

operations, as specified in the Instruction Set Reference. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code. The Status Register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

Stack Pointer:

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. Note that the Stack is implemented as growing from higher to lower memory locations. The Stack Pointer Register always points to the top of the Stack. The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. A Stack PUSH command will decrease the Stack Pointer.

Serial: Pin 0 (RX) and Pin 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.

External Interrupts: Pin 2 and Pin 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.

PWM: Pins 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the `analogWrite()` function. User may have noticed the tilde (~) next to some of the digital pins (3, 5, 6, 9, 10, and 11 on the UNO). These pins act as normal digital pins, but can also be used for something called Pulse-Width Modulation (PWM).

SPI: Pin 10 (SS), Pin 11 (MOSI), Pin 12 (MISO), Pin 13 (SCK). These pins support SPI communication using the SPI library.

LED: Pin 13. 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.

TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

The Uno has 6 analog inputs, labelled 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. There are a couple of other pins on the board:

Low Cost Self-Activating Touchless Hand Sanitizing Dispensing with Battery Imposed System

AREF: Reference voltage for the analog inputs. Used with `analogReference()`. Stands for Analog Reference. Most of the time user can leave this pin alone. It is sometimes used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.

Reset: Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Analog Input: The area of pins under the 'Analog In' label (A0 through A5 on the UNO) are Analog In pins. Analog pins can read the signal from an analog sensor (like a temperature sensor) and convert it into a digital value that human can read.

Digital Pins: Across from the analog pins are the digital pins (0 through 13 on the UNO). These pins can be used for both digital input (like telling if a button is pushed) and digital output (like powering an LED).

Power Pins:

The Arduino Uno board can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm centre-positive plug into the board's power jack. Leads from a battery can be inserted in the GND and Vin pin headers of the POWER connector.

The board can operate on an external supply from 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may become unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

Vin: The input voltage to the Arduino 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.

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 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage the board. The 5V pin supplies 5 volts of power, and the 3.3V pin supplies 3.3 volts of power. Most of the simple components used with the Arduino run happily off of 5 or 3.3 volts.

3V3: A 3.3 V supply generated by the on-board regulator. Maximum current draw is 50 mA.

GND: Ground pins. Short for 'Ground'. There are several GND pins on the Arduino, any of which can be used to ground your circuit.

IOREF: This pin on the Arduino 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.

3.3.3.2 Chip 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. The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino Software (IDE) includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPI communication, use the SPI library.

3.3.3.3 Reset Button

Arduino has a reset button. Pushing it will temporarily connect the reset pin to ground and restart any code that is loaded on the Arduino. This can be very useful if the code doesn't repeat, but if user want to test it multiple times.

3.3.3.4 Voltage Regulator

The voltage regulator is not actually something user can (or should) interact with on the Arduino UNO. But it is potentially useful to know that it is there and what it's for. The voltage regulator does exactly what it says it controls the amount of voltage that is let into the Arduino board.

3.3.3.5 Memory

The ATmega328 has 32 KB (with 0.5 KB occupied by the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the [EEPROM library](#)).

3.3.3.6 TX & RX LED's

TX is short for transmit; RX is short for receive. These markings appear quite a bit in electronics to indicate the pins responsible for serial communication. In this case, there are two places on the Arduino UNO where TX and RX appear once by digital pins 0 and 1, and a second time next to the TX and RX indicator LEDs. These LEDs will give user some nice visual indications whenever his Arduino is receiving or transmitting data (like when he's loading a new program onto the board).

3.3.3.7 Power LED Indicator

Just beneath and to the right of the word "UNO" on the circuit board, there's a tiny LED next to the word 'ON'. This LED should light up whenever user plug his Arduino into a power source. If this light doesn't turn on, there's a good chance something is wrong.

3.4 USB Cable

Arduino Uno board can be connected with the USB female A port of computer. USB cable length is approximately 1m, USB cable type is A/B Standard USB 2.0 cable.



Table 3.21 USB Cable

3.5 9V Battery, Connector with Power Jack

The nine-volt battery in its most common form was introduced for the early transistor radios. It has a rectangular prism shape with rounded edges and a polarized snap connector at the top. This type is commonly used in pocket radios, paintball guns, and small electronic devices. They are also used as backup power to keep the time in certain electronic clocks. This format is commonly available in primary carbon-zinc and alkaline chemistry, in primary lithium iron disulfide, and in rechargeable form in nickel-cadmium, nickel-metal hydride and lithium-ion. Mercury oxide batteries in this form have not been manufactured in many years due to their mercury content. This type is designated NEDA 1604, IEC 6F22 and "Ever Ready" type PP3 (zinc-carbon) or MN1604 6LR61 (alkaline).



Figure 3.22 9V Battery



Figure 3.23 Battery Connector with 2.1mm Center Positive Jack

CHAPTER 4

4. SOFTWARE IDE DESCRIPTION

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino and Genuino hardware to upload programs and communicate with them.

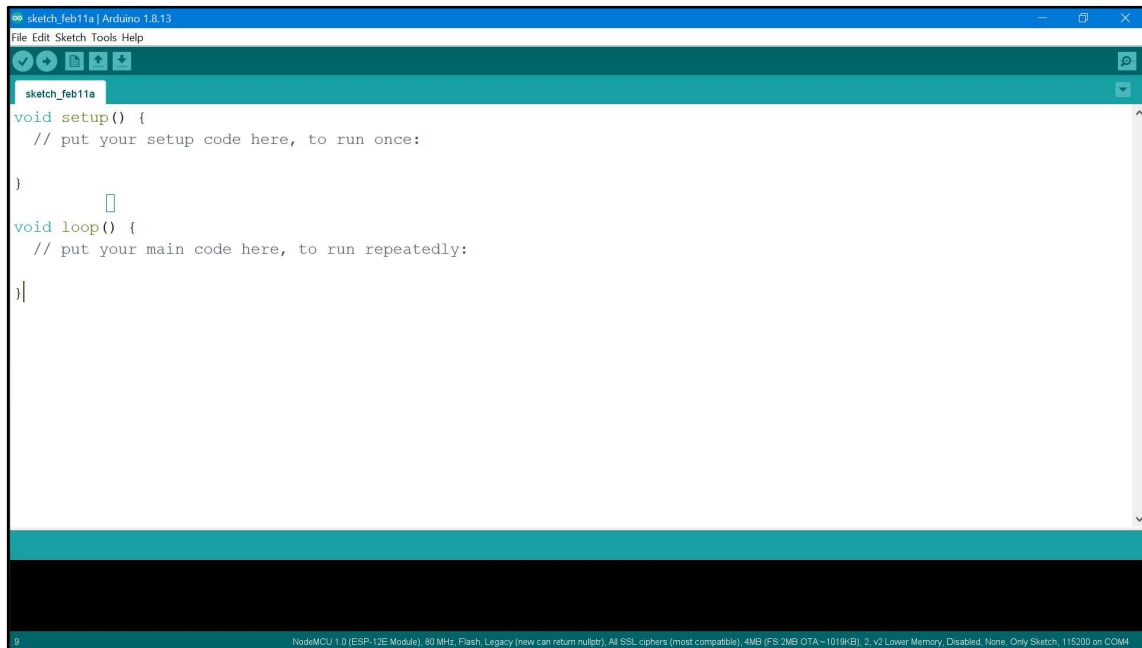








Figure 4.1 Arduino IDE

4.1 Writing Sketches

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.

Low Cost Self-Activating Touchless Hand Sanitizing Dispensing with Battery Imposed System

NB: Versions of the Arduino Software (IDE) prior to 1.0 saved sketches with the extension .pde. It is possible to open these files with version 1.0, you will be prompted to save the sketch with the .ino extension on save.

	Verify: Checks your code for errors compiling it.
	<p>Upload: Compiles your code and uploads it to the configured board. See uploading below for details.</p> <p>Note: If you are using an external programmer with your board, you can hold down the "shift" key on your computer when using this icon. The text will change to "Upload using Programmer"</p>
	New: Creates a new sketch.
	<p>Open: Presents a menu of all the sketches in your sketchbook. Clicking one will open it within the current window overwriting its content.</p> <p>Note: due to a bug in Java, this menu doesn't scroll; if you need to open a sketch late in the list, use the File Sketchbook menu instead.</p>
	Save: Saves your sketch.
	Serial Monitor: Opens the serial monitor.

Additional 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.

4.1.1 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.

4.1.2 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.

4.1.3 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. For more details, see libraries below. Additionally, from this menu item you can access the Library Manager and import new libraries from .zip files.
- Add File...: Adds a source file to the sketch (it will be copied from its current location). The new file appears in a new tab in the sketch window. Files can be removed from the sketch using the tab menu accessible clicking on the small triangle icon below the serial monitor one on the right-side of the toolbar.

4.1.4 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.

Low Cost Self-Activating Touchless Hand Sanitizing Dispensing with Battery Imposed System

- **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.
- **Board:** Select the board that you're using. See below for descriptions of the various boards.
- **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. Normally you won't need this, but if you're burning a bootloader to a new microcontroller, you will use this.
- **Burn Bootloader:** The items in this menu allow you to burn a bootloader onto the microcontroller on an Arduino board. This is not required for normal use of an Arduino or Genuino 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.

4.1.5 Help

Here you find easy access to a number of documents that come with the Arduino Software (IDE). You have access to Getting Started, Reference, this guide to the IDE and other documents locally, without an internet connection. The documents are a local copy of the online ones and may link back to our online website.

- **Find in Reference:** This is the only interactive function of the Help menu: it directly selects the relevant page in the local copy of the Reference for the function or command under the cursor.

4.2 Sketchbook

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.

4.3 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).

4.4 Uploading

Before uploading your sketch, you need to select the correct items from the Tools > Board and Tools > Port menus. The 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 File 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).

4.5 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.

There is a list of libraries in the reference. 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. See these instructions for installing a third-party library.

4.6 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.

For details on creating packages for third-party hardware, see the Arduino IDE 1.5 3rd party Hardware specification.

4.7 Serial Monitor

Displays serial data being sent from the Arduino or Genuino board (USB or serial board). 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 that matches the rate passed to `Serial.begin` in your sketch. Note that on Windows, Mac or Linux, the Arduino or Genuino board will reset (rerun your sketch execution to the beginning) when you connect with the serial monitor.

You can also talk to the board from Processing, Flash, MaxMSP, etc

4.8 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.

4.9 Language Support

Since version 1.0.1, the Arduino Software (IDE) has been translated into 30+ different languages. By default, the IDE loads in the language selected by your operating system. (Note: on Windows and possibly Linux, this is determined by the locale setting which controls currency and date formats, not by the language the operating system is displayed in.)

If you would like to change the language manually, start the Arduino Software (IDE) and open the Preferences window. Next to the Editor Language there is a dropdown menu of currently supported languages. Select your preferred language from the menu, and restart the software to use the selected language. If your operating system language is not supported, the Arduino Software (IDE) will default to English.

You can return the software to its default setting of selecting its language based on your operating system by selecting System Default from the Editor Language drop-down. This setting will take effect when you restart the Arduino Software (IDE). Similarly, after changing your operating system's settings, you must restart the Arduino Software (IDE) to update it to the new default language.

4.10 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. Some of the board definitions differ only in the latter, so even if you've been uploading successfully with a particular selection, you'll want to check it before burning the bootloader. You can find a comparison table between the various boards [here](#).

Arduino Software (IDE) includes the built-in support for the boards in the following list, all based on the AVR Core. The Boards Manager included in the standard installation allows to add support for the growing number of new boards based on different cores like Arduino Due, Arduino Zero, Edison, Galileo and so on.

CHAPTER 5

5. ADVANTAGES & APPLICATION

5.1 Advantage

1. The most significant advantage of this sanitizer dispenser is the hands-free usage. The spreading of germs / viruses by means of touching the pump of sanitizer bottle by a large amount of people is almost reduced to zero by means of this dispensing machine.
2. This machine is user-friendly. This machine does not require any high-end skills nor does it require massive amount of strength for operation purpose.
3. Almost all parts required for the manufacturing of this dispenser can be obtained from the industrial waste discarded by fabrication, construction, production, manufacturing factories, electronics shop.
4. This machine is portable and does not require much space. It can easily be placed in cramped-up places.
5. This machine entirely works on electronics mechanism. It requires only power source or battery cells for operation.
6. This battery-operated sanitizer dispenser is cost efficient.

5.2 Applications

The applications of the sanitizer dispensers are so vast that they can be placed upon any place with a heavy traffic of people. Shopping malls, supermarkets, gas stations, parks, etc. are some places where the machines could be placed. Some of the places which require immediate installation of these dispensers are:

Hospitals – These dispensers should be placed at regular intervals in the hospitals so as to prevent the spread of infections. Patients, Doctors, Nurses, Staff members, Visitors, etc. would be tremendously benefitted by this foot-operated sanitizer dispenser.

Restaurants - These machines can be placed at the entrance of all the restaurants, food chains, etc. People entering the restaurant will have their hands sanitized and clean for food consumption by the use of this machine. A dispenser can also be placed in the kitchen of all restaurants for the chefs and waiters.

Low Cost Self-Activating Touchless Hand Sanitizing Dispensing with Battery Imposed System

Offices – The desks of the workers can be equipped with this sanitizer dispensers. Also, the meeting rooms, conference rooms, can have these dispensers to avoid spreading of viruses.

Schools / Colleges – All Educational institutions must have these dispensers outside classrooms, in the cafeterias, near the playgrounds, etc. as a large number of students visiting the campus can easily contract viruses, infections from some infected students.

Religious places- Religious shrines such as temples, church, mosque, etc. consists of a large number of people visiting every day. So, these sands can be placed at the entrance of these places. Also, the modified stand can be used in this case which can even be used in disinfecting the belongings of the people visiting these places.

CHAPTER 6

6. RESULT & CONCLUSION

6.1 Result

By implementing this sanitizer dispensing machine users can avoid the spread of Covid-19 due to infected hands. This sanitizer dispenser activates when it detects an human in front of it using Ultrasonic Sensor and dispenses the sanitizer from sanitizer bottle by pressing its dispensing nozzle using servo motor.

6.2 Conclusion

Implementing of Contactless Automatic Hand Wash Dispenser for Sanitation is efficient and the cost price is minimized. It works like the normal contactless automatic machine. The human gets the limited sanitizer liquid for sanitation in hand, to wash the hands and to protect themselves from the corona disease. This system can be utilized in malls, high populated areas. The economic cost of the project, it will be better quality when considering the life of the system and the project.

CHAPTER 7

7. REFERENCES

- [1] <https://www.arduino.cc/en/Guide/Introduction/>
- [2] <https://learn.adafruit.com/arduino-tips-tricks-and-techniques>
- [3] <https://en.wikipedia.org/wiki/Arduino>
- [4] <https://learn.sparkfun.com/tutorials/what-is-an-arduino/all>
- [5] <https://github.com/arduino/Arduino>
- [6] <https://opensource.com/resources/what-arduino>
- [7] <https://www.tutorialspoint.com/arduino/index.htm>
- [8] <https://www.youtube.com/arduino>
- [9] <https://www.youtube.com/watch?v=jRqLFn9rDJs&t=8202s>
- [10] <https://www.youtube.com/watch?v=hjRSwBcLcSU>
- [11] <https://www.youtube.com/watch?v=7uHfjpU3OH0>
- [12] https://www.youtube.com/watch?v=CbJHL_P5RJ8
- [13] https://www.youtube.com/watch?v=672_Y8PxFJc
- [14] <https://www.makerspaces.com/arduino-uno-tutorial-beginners/>
- [15] <https://www.sparkfun.com/products/15569>
- [16] <https://www.sparkfun.com/HCSR04>
- [17] <https://www.sparkfun.com/products/10218>
- [18] <https://www.sparkfun.com/products/10512>
- [19] https://en.wikipedia.org/wiki/Nine-volt_battery

Low Cost Self-Activating Touchless Hand Sanitizing Dispensing with Battery Imposed System

- [20] <https://www.duracell.in/product/ultra-alkaline-9v-batteries/>
- [21] <http://www.evereadyindia.com/battery/battery-filter.aspx?size=v9>
- [22] <https://learn.sparkfun.com/tutorials/installing-an-arduino-library>
- [23] <https://learn.sparkfun.com/tutorials/installing-arduino-ide>
- [24] <https://learn.sparkfun.com/tutorials/installing-board-definitions-in-the-arduino-ide>
- [25] <https://learn.sparkfun.com/tutorials/data-types-in-arduino>
- [26] <https://learn.sparkfun.com/tutorials/analog-vs-digital>
- [27] <https://learn.sparkfun.com/tutorials/logic-levels>
- [28] <https://www.sparkfun.com/servos#types-of-servos>
- [29] <https://youtu.be/vbAZdQGTA0s>
- [30] <https://learn.sparkfun.com/tutorials/pulse-width-modulation/all>
- [31] <https://learn.sparkfun.com/tutorials/basic-servo-control-for-beginners>
- [32] <https://learn.sparkfun.com/tutorials/hobby-servo-tutorial>
- [33] <https://www.sparkfun.com/servos>

Appendix-A

```
#include<Servo.h> // Including Servo Motor header file

#define pinServo 2 // Declaring Pin 2 as Servo Motor signal pin

Servo S1;

int degree = 0;

long cm = 0;

long readUltrasonicDistance (int triggerPin, int echoPin)
{
    pinMode (triggerPin, OUTPUT);
    digitalWrite (triggerPin, LOW);           //sets the trigger pin off
    delayMicroseconds (2);
    digitalWrite (triggerPin, HIGH);
    delayMicroseconds (10);
    digitalWrite (triggerPin, LOW);
    pinMode (echoPin, INPUT);                 //sets the echo pin as input
    return (pulseIn(echoPin, HIGH)*0.01723);
}

void setup ()
{
    S1.attach(pinServo);
    Serial.begin (9600);                      //initialize serial communication
    S1.write (0);                             //set servo to 0 degree
    delay (2000);
    pinMode (2, OUTPUT);
}
```

Low Cost Self-Activating Touchless Hand Sanitizing Dispensing with Battery Imposed System

```
void loop ()
{
    cm = readUltrasonicDistance (5, 4);
    Serial.print(cm);
    Serial.println ("cm");
    if (cm > 40) {
        digitalWrite (2, LOW);
        S1.write(0);
    }
    if (cm <= 40)
    {
        digitalWrite (2, HIGH);
        S1.write(60);
        delay (500);
        S1.write(0);
        delay (3000);
    }
    delay (100);
}
```