PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100 Ft. Road, BSK III Stage, Bengaluru – 560 085

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Course Title: Problem Solving with C Laboratory		
Course code: UE19CS152		
Semester : II sem	**Section: I**	**Team Id: I_10**
SRN: PES2UG19CS396	**Name:** Snehil Jain	
SRN: PES2UG19CS282	**Name:** Pihoo Verma	
SRN: PES2UG19CS326	**Name:** Ria Singh	

PROJECT REPORT

Problem Statement: Position Sensor Handwash Dispenser

Description: After giving arduino power, when we place our hand in front of the ultrasonic sensor at distance less than 5cm, the arduino commands one servo to rotate 100 degrees clockwise, and other servo to rotate 100 degrees anti-clockwise, thus putting a vertical force on the nozzle of the hand-wash liquid dispenser. Hence, without even touching the dispenser, the liquid gets dispensed. As we remove our hand, the distance increases from the sensor, the arduino commands as per the program to return back to their normal position.

C-concepts used:

Macros

1) #define

#define is a useful C++ component that allows the programmer to give a name to a constant value before the program is compiled. Defined constants in arduino don't take up any program memory space on the chip. The compiler will replace references to these constants with the defined value at compile time. This can have some unwanted side effects though, if for example, a constant name that had been #defined is included in some other constant or variable name. In that case the text would be replaced by the #defined number (or

text). In general, the const keyword is preferred for defining constants and should be used instead of #define.
Syntax: #define constantName value
Parameters: constantName: the name of the macro to define, value: the value to assign to the macro.

Arduino Library

1) servo

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.

Syntax: #include <Servo.h>
Circuits: 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.

Functions: attach(), write(), read(), attached(), detach(), writeMicroseconds()

Arduino Constants

Defining Pin Levels: HIGH and LOW: When reading or writing to a digital pin there are only two possible values a pin can take/be-set-to: and
HIGH
The meaning of HIGH (in reference to a pin) is somewhat different depending on whether a pin is set to an INPUT or OUTPUT. When a pin is configured as an INPUT with pinMode(), and read with digitalRead(), the Arduino (ATmega) will report HIGH if:

- a voltage greater than 3.0V is present at the pin (5V boards)
- a voltage greater than 2.0V volts is present at the pin (3.3V boards)

A pin may also be configured as an INPUT with pinMode(), and subsequently made HIGH with digitalWrite(). This will enable the internal 20K pullup resistors, which will pull up the input pin to a HIGH reading unless it is pulled LOW by external circuitry. This can done alternatively by passing INPUT_PULLUP as argument to the pinMode() funtion, as explained in more detail in the section "Defining Digital Pins modes: INPUT, INPUT_PULLUP, and OUTPUT" further below. When a pin is configured to OUTPUT with pinMode(), and set to HIGH with digitalWrite(), the pin is at:

- 5 volts (5V boards)
- 3.3 volts (3.3V boards)

In this state it can source current, e.g. light an LED that is connected through a series resistor to ground.

LOW
The meaning of LOW also has a different meaning depending on whether a pin is set to INPUT or OUTPUT. When a pin is configured as an INPUT with pinMode(), and read with digitalRead(), the Arduino (ATmega) will report LOW if:

- a voltage less than 1.5V is present at the pin (5V boards)

- a voltage less than 1.0V (Approx) is present at the pin (3.3V boards)

When a pin is configured to OUTPUT with pinMode(), and set to LOW with digitalWrite(), the pin is at 0 volts (both 5V and 3.3V boards). In this state it can sink current, e.g. light an LED that is connected through a series resistor to +5 volts (or +3.3 volts).

Defining Digital Pins modes: INPUT, INPUT_PULLUP, and OUTPUT: Changing a pin with pinMode() changes the electrical behavior of the pin.

Pins Configured as INPUT

Arduino (ATmega) pins configured as INPUT with pinMode() are said to be in a high-impedance state. Pins configured as INPUT make extremely small demands on the circuit that they are sampling, equivalent to a series resistor of 100 Megohms in front of the pin. This makes them useful for reading a sensor. If you have your pin configured as an INPUT, and are reading a switch, when the switch is in the open state the input pin will be "floating", resulting in unpredictable results. In order to assure a proper reading when the switch is open, a pull-up or pull-down resistor must be used. The purpose of this resistor is to pull the pin to a known state when the switch is open. A 10 K ohm resistor is usually chosen, as it is a low enough value to reliably prevent a floating input, and at the same time a high enough value to not draw too much current when the switch is closed. See the Digital Read Serial tutorial for more information. If a pull-down resistor is used, the input pin will be LOW when the switch is open and HIGH when the switch is closed. If a pull-up resistor is used, the input pin will be HIGH when the switch is open and LOW when the switch is closed.

Pins Configured as OUTPUT

Pins configured as OUTPUT with pinMode() are said to be in a low-impedance state. This means that they can provide a substantial amount of current to other circuits. ATmega pins can source (provide current) or sink (absorb current) up to 40 mA (milliamps) of current to other devices/circuits. This makes them useful for powering LEDs because LEDs typically use less than 40 mA. Loads greater than 40 mA (e.g. motors) will require a transistor or other interface circuitry. Pins configured as outputs can be damaged or destroyed if they are connected to either the ground or positive power rails.

Arduino Functions

1) digitalWrite()

[Digital I/O]
Description
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

2) delayMicroseconds()

[Time]
Description
Pauses the program for the amount of time (in microseconds) specified by the parameter. There are a thousand microseconds in a millisecond and a million microseconds in a second.Currently, the largest value that will produce an accurate delay is 16383. This could change in future Arduino releases. For delays longer than a few thousand microseconds, you should use delay() instead.
Syntax: delayMicroseconds(us)
Parameters: us- the number of microseconds to pause.
Allowed data types: unsigned int.
Returns: Nothing

3)pulseIn()

[Advanced I/O]

Description

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. The timing of this function has been determined empirically and will probably show errors in longer pulses. Works on pulses from 10 microseconds to 3 minutes in length.

Syntax:

```
pulseIn(pin, value)
pulseIn(pin, value, timeout)
```

Parameters:

pin: the number of the Arduino pin on which you want to read the pulse. Allowed data types: int.
value: type of pulse to read: either HIGH or LOW. Allowed data types: int.
timeout (optional): the number of microseconds to wait for the pulse to start; default is one second. Allowed data types: unsigned long.
Returns: The length of the pulse (in microseconds) or 0 if no pulse started before the timeout.

4)Serial.println()

Description

Prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n'). This command takes the same forms as Serial.print().

Syntax:

```
Serial.println(val)
Serial.println(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.
format: specifies the number base (for integral data types) or number of decimal places (for floating point types).

Returns: println() returns the number of bytes written, though reading that number is optional.

6) pinMode()

[Digital I/O]

Description

Configures the specified pin to behave either as an input or an output. See the Digital Pins page for details on the functionality of the pins.As of Arduino 1.0.1, it is possible to enable the internal pullup resistors with the mode INPUT_PULLUP. Additionally, the INPUT mode explicitly disables the internal pullups.

Syntax: pinMode(pin, mode)

Parameters:

pin: the Arduino pin number to set the mode of.
mode: INPUT, OUTPUT, or INPUT_PULLUP. See the Digital Pins page for a more complete description of the functionality.
Returns: Nothing

Arduino Structure

1) loop()

Description

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);
}
```

2) setup()

[Sketch]

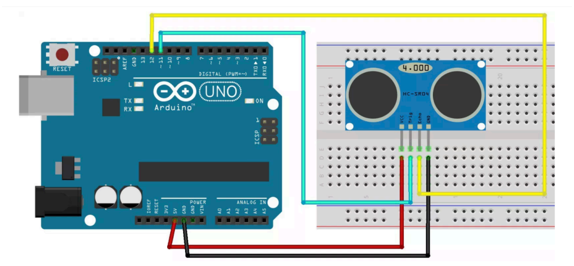
Description

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() {
  // ...
}
```

Arduino Circuit with the Position Sensor



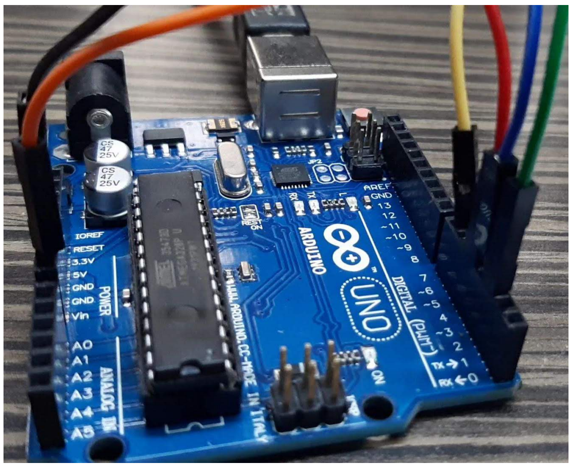
The following table shows the connections you need to make:

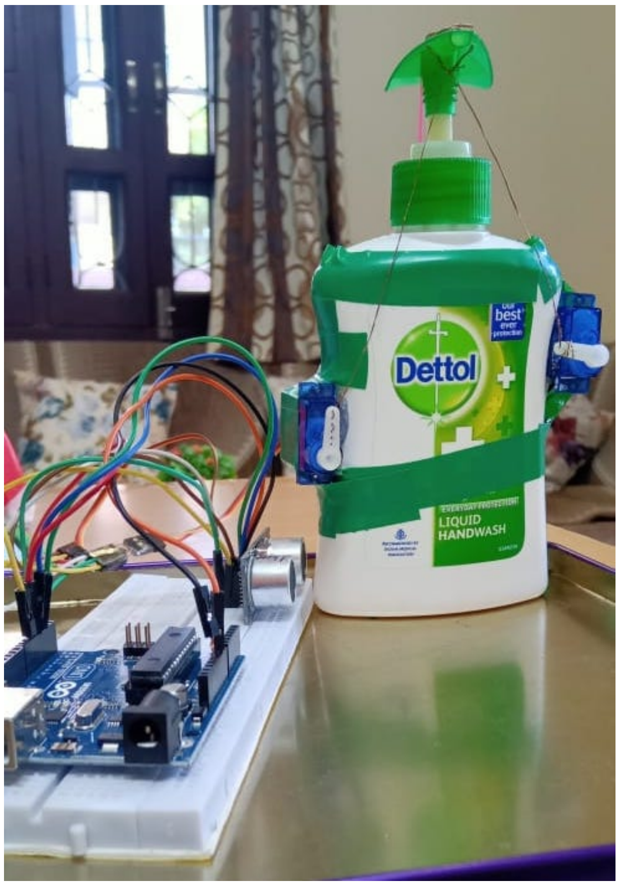
Ultrasonic Sensor HC-SR04	Arduino
VCC	5V
Trig	Pin 11
Echo	Pin 12
GND	GND

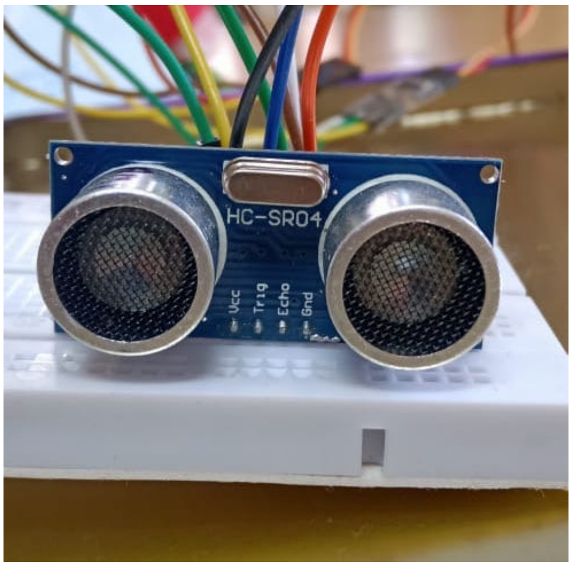
Learning Outcome:

1) Understood that in position sensor, ultrasonic element is used to calculate the distance and respond appropriately.
2) Studied how to code Arduino functions, loops, structures in C.
3) Learned how to use mechanical rotation of servo for our purposes.

Output Screenshots:







Output Video: https://www.youtube.com/watch?v=58pv1Sw96Lc&feature=youtu.be

Name and Signature of the Faculty: