# Final Project Report

John Clauson, Henryk Sosnowski, Jordin Carlon

CPE 301

Dec 15, 2024

# Description of Physical Circuit Design and Code:

**Circuit Design:**

We connected all of the components across 3 bread boards shown in figure 1.There are 4 LEDs colored blue, red, green, and yellow which light up to show the current state of the system. The red LED lights up when in the error state, blue when running, yellow when stopped, and green when idle. Each of the LEDs is connected to a pull up resistor. Three buttons are used to start the system, reset during the error state, or change the vent position. These also have a pull down resistor connected to each of them. The DHT module measures the humidity and temperature which are used to determine state changes. They are also displayed on the LCD screen. The DS1307 RTC is used to track the time that state transitions occur. The state transitions will print to the serial monitor with a time stamp from the RTC. The stepper motor changes the vent position and can be changed by pressing the vent button, which will rotate it a tenth of a rotation for each button press. The LCD displays the temperature and humidity during the idle and running states. It also displays an error message when the system is in error state. The LCD is connected to the potentiometer to control contrast. The fan motor and fan blade move air through the cooler. It runs during the running state and stops during any other state. The power supply module keeps the motors from burning out the Arduino board. The water level detection sensor module is used to tell the system when to enter the error state when the water level gets low.

This system as demonstrated is powered by an AC/DC adapter plugged into a wall outlet and the Arduino powered via USB for serial logging. It can also operate remotely using two 9 V batteries. The system will not overheat under normal operating conditions outside of several hours of continuous fan operation. The stock water sensor from the Arduino kit functioned for testing and prototyping but is not adequate for continuous use. After an hour of operation with the sensor submerged in water the leads began to corrode and flake off, so an alternative component is required for practical use.

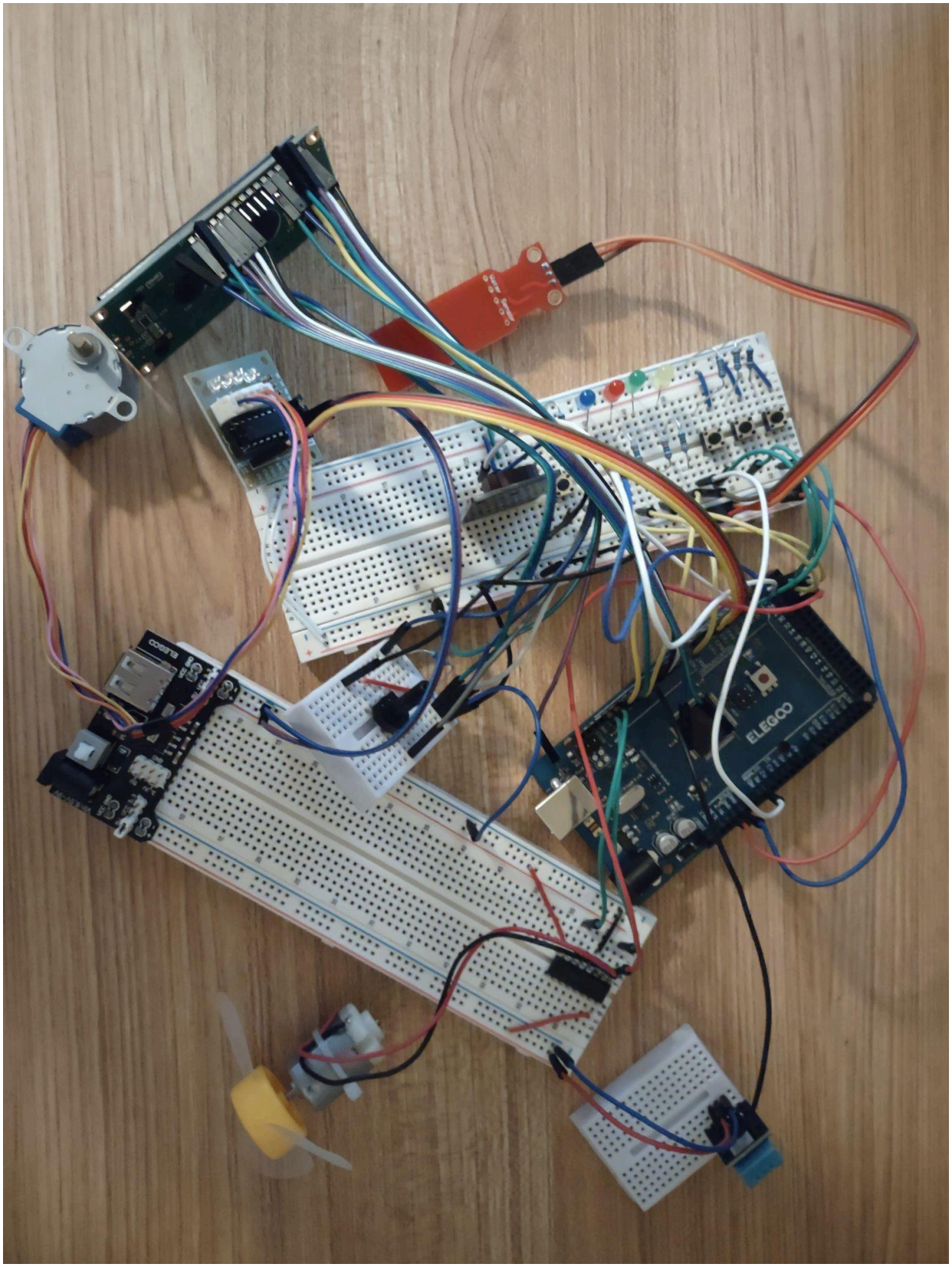


Figure 1: Complete Circuit

**Code Breakdown:**

The total code is 420 lines long, with 12 functions. The constraints of the project allowed us to use the libraries shown below, but none of the basic library functions. Delay(), digitalwrite and read, analogwrite and read, any of the Serial monitor library functions, and pinmode. Figure 2, which can be seen below, is the first few lines of code, initializing all the permitted libraries.

```
//Clauson, John; Sosnowski, Henryk; Carlon, Jordin
// CPE301 Final Project

#include <dht.h>
#include <Stepper.h>
#include <RTClib.h>
#include <Wire.h>
#include <LiquidCrystal.h>

dht DHT; //Humidity/Temp Sensor
#define DHT11_PIN 8

RTC_DS3231 rtc; //Real Time Clock Module

//define RDA and TBE to check for received data and if buffer is empty
#define RDA 0x80
#define TBE 0x20
```

Figure 2: Library Initialization

After that, we used GPIO to set pins and prepare the serial monitor.

```
//Serialprint
volatile unsigned char *myUCSR0A = (unsigned char *)0x00C0;
volatile unsigned char *myUCSR0B = (unsigned char *)0x00C1;
volatile unsigned char *myUCSR0C = (unsigned char *)0x00C2;
volatile unsigned int  *myUBRR0  = (unsigned int *) 0x00C4;
volatile unsigned char *myUDR0   = (unsigned char *)0x00C6;

//pin modes
volatile unsigned char* port_a = (unsigned char*) 0x22;
volatile unsigned char* ddr_a = (unsigned char*) 0x21;
volatile unsigned char* pin_a = (unsigned char*) 0x20;
volatile unsigned char* port_b = (unsigned char*) 0x25;
volatile unsigned char* ddr_b = (unsigned char*) 0x24;
volatile unsigned char* port_d = (unsigned char*) 0x2B;
volatile unsigned char* ddr_d = (unsigned char*) 0x2A;
volatile unsigned char* pin_d = (unsigned char*) 0x29;
volatile unsigned char* port_e = (unsigned char*) 0x2E;
volatile unsigned char* ddr_e = (unsigned char*) 0x2D;
volatile unsigned char* pin_e = (unsigned char*) 0x2C;
volatile unsigned char* port_h = (unsigned char*) 0x102;
volatile unsigned char* ddr_h = (unsigned char*) 0x101;

//unsigned variables for ADC
volatile unsigned char* my_ADMUX = (unsigned char*) 0x7C;
volatile unsigned char* my_ADCSRB = (unsigned char*) 0x7B;
volatile unsigned char* my_ADCSRA = (unsigned char*) 0x7A;
volatile unsigned int* my_ADC_DATA = (unsigned int*) 0x78;
```

Figure 3: GPIO Ports, DDR, and Pins

Next, lines 47-73 were used to store different global variables not related to the GPIO, set the interrupt, and prepare the LCD library for operation. Void setup() begins at line 75, and starts by starting the serial monitor, and setting pins to output or input using GPIO.

```
void setup() {
  U0init(9600);
  Serialstring("Serial Monitor Ready");
  U0putchar('\n');

  attachInterrupt(digitalPinToInterrupt(Start), pin_ISR, RISING); //Setting inturrupt for Start button

  //Pin modes for all components
  *ddr_a &= 0xFE; //Sets 22 PA0 to input for Stop Button
  *port_a |= 0x01;
  *ddr_a &= 0xFD; //Sets 23 PA1 to input for Reset Button
  *port_a |= 0x02;
  *ddr_d &= 0xF7;//Sets 18 PD3 to input for Vent Button
  *port_d |= 0x08;
  *ddr_e |= 0x10;  //Sets 2 PE4 to output for Red LED
  *ddr_h |= 0x40; //Sets 9 PH6 to output for Yellow LED
  *ddr_a |= 0x04;  //Sets 24 PA2 to output for Green LED
  *ddr_a |= 0x08;  //Sets 25 PA3 to output for Blue LED
  *port_e |= 0x20;
  *ddr_a |= 0x80;  //Sets 29 PA7 to output for DC motor enable
  *ddr_b |= 0x40;  //Sets 12 PB6 to output for DC motor input 1
  *ddr_b |= 0x80;  //Sets 13 PB7 to output for DC motor input 2

  count = millis(); //inital count value
```

Figure 4: Starting Serial Monitor, Setting Pins to Output or Input.

The void setup() function concludes by starting the rtc library, and running a test script to output the startup time to the serial monitor. Line 118 and 119 are used to initialize the LCD and ADC.

```
Wire.begin();
rtc.begin();

DateTime now = rtc.now(); //Initial Clock Output mm/dd/yyyy "day of the week" hh:mm:ss

Serialint(now.month());
Serialstring("/");
Serialint(now.day());
Serialstring("/");
Serialint(now.year());
Serialstring(" ");
Serialint(now.hour());
Serialstring(":");
Serialint(now.minute());
Serialstring(":");
Serialint(now.second());
Serialstring("\n");

lcd.begin(16,2); //LCD columns and rows
adc_init();  //Start the ADC
```

Figure 5: RTC Test and LCD Initialization

The void setup() was followed by the void loop(), which provides the main operation for the circuit. If it is composed of 4 main “if” statements. The first times intervals between clock updates and writes clock outputs to the LCD. The second if statement allows for control over the

stepper motor position in certain states. The third is simply designed to reset the LCD in case a previous state has not been cleared. FInally, the fourth is the main state-control system, which takes inputs from sensors and buttons, and changes the state accordingly, as explained in the circuit operation explanation above.

```
if(((millis() - count) > 1000) && (state != 0)) { //output clock, temperature, and humidity updates every second
  int chk = DHT.read11(DHT11_PIN);
  DateTime now = rtc.now();
  sprintf(t, "%02d:%02d:%02d", now.hour(), now.minute(), now.second());
  lcd.setCursor(0,0);
  lcd.write(t); //prints time to top left corner of lcd screen

  int temperature = DHT.temperature;
  int humidity = DHT.humidity;

  lcd.setCursor(0,1);
  lcd.print(temperature); //prints temp and humidity to lcd screen
  lcd.setCursor(2,1);
  lcd.print("C");
  lcd.setCursor(13,1);
  lcd.print(humidity);
  lcd.setCursor(15,1);
  lcd.print("%");

  count = millis(); //updates "count" value for next iteration
}
```

Figure 6: Clock Update Checker

```
if((state != 0) && (*pin_d & 0x08)) { //Vent control loop
  myStepper.setSpeed(10); //speed set to 10 rpm
  myStepper.step(stepsPerRevolution/10); //rotate a tenth of a rotation every time
  lcd.setCursor(5,1);
  lcd.print("VENT");
  if(tracker == 0) {
    int chk = DHT.read11(DHT11_PIN);
    DateTime now = rtc.now();
    Serialstring("Date/Time: "); //prints "Date/Time: " to serial monitor
    Serialint(now.month());
    Serialstring("/");
    Serialint(now.day());
    Serialstring("/");
    Serialint(now.year());
    Serialstring(" ");
    Serialint(now.hour());
    Serialstring(":");
    Serialint(now.minute());
    Serialstring(":");
    Serialint(now.second());
    Serialstring(" ");
    Serialstring("VENT CHANGE\n");
    tracker++;
  }
} else {
  lcd.setCursor(5,1);
  lcd.print("    ");
  tracker = 0;
}
```

Figure 7: Stepper Angle Adjustment Statement

```
if(begin == 1) {
  lcd.setCursor(4,0); //reset lcd
  lcd.print("        ");
  begin = 0;
}
```

Figure 8: LCD Reset Statement

The fourth main operation statement is too long and complex to show, but can be seen in the github files from line 181 to 288. These lines are split into the four operating modes (idle, running, error, and disabled) depending on the current state variable and primarily control the indicator LEDs and DC motor function. This concludes the loop and leads into our functions, starting with state_trans (Fig. 9) that prints state transitions with a timestamp to the serial monitor for logging.

```
int state_trans() {
  int chk = DHT.read11(DHT11_PIN);
  DateTime now = rtc.now();
  while(now.day()<1 || now.day()>31){
    now = rtc.now();
  }
  Serialstring("Date/Time: "); //prints "Date/Time: " to serial monitor
  Serialint(now.month());
  Serialstring("/");
  Serialint(now.day());
  Serialstring("/");
  Serialint(now.year());
  Serialstring(" ");
  Serialint(now.hour());
  Serialstring(":");
  Serialint(now.minute());
  Serialstring(":");
  Serialint(now.second());
  Serialstring(" ");
  if(state == 0) { //Prints correct state to serial monitor
    Serialstring("STATE: DISABLED\n");
  } else if(state == 1) {
    Serialstring("STATE: IDLE\n");
  } else if(state == 2) {
    Serialstring("STATE: ERROR\n");
  } else if(state == 3) {
    Serialstring("STATE: RUNNING\n");
    }
}
```

Figure 9: Serial Monitor Logging Function

The next function of note is Serialint (Fig. 10), which prints an integer input of one to four digits to the serial monitor. It works by determining the length of the input and stripping each digit into an array before printing via the UDR0 one by one. Longer inputs result in bugs, but as this is only intended for printing dates it will work until the year 10000.

```
//prints an int between 1 to 4 digits long to the serial monitor
void Serialint(int input){
  int length = 1;
  int inputl = input;
  for (int i = 0; inputl/10 !=0; i++){
    length += 1;
    inputl = inputl/10;
  }
  int print[length] = {0};
  for (int n=1; n <= length; n++){
    print[length-n] = input%10;
    input = input/10;
  }
  for (int n=0; n < length; n++){
    while((*myUCSR0A & TBE)==0);
    *myUDR0 = print[n] + '0';
  }
}
```

Figure 10: Serial Monitor Integer Printing Function

The next function, Serialstring (Fig. 11), prints a character array input to the serial monitor one at a time via the UDR0.

```
//prints a char array to the serial monitor
void Serialstring(unsigned char input[]) {
  for (int i=0; input[i]; i++){
    while((*myUCSR0A & TBE)==0);
    *myUDR0 = input[i];
  }
}
```

Figure 11: Serial Monitor String Printing Function

The other functions were based on previously used lab functions.

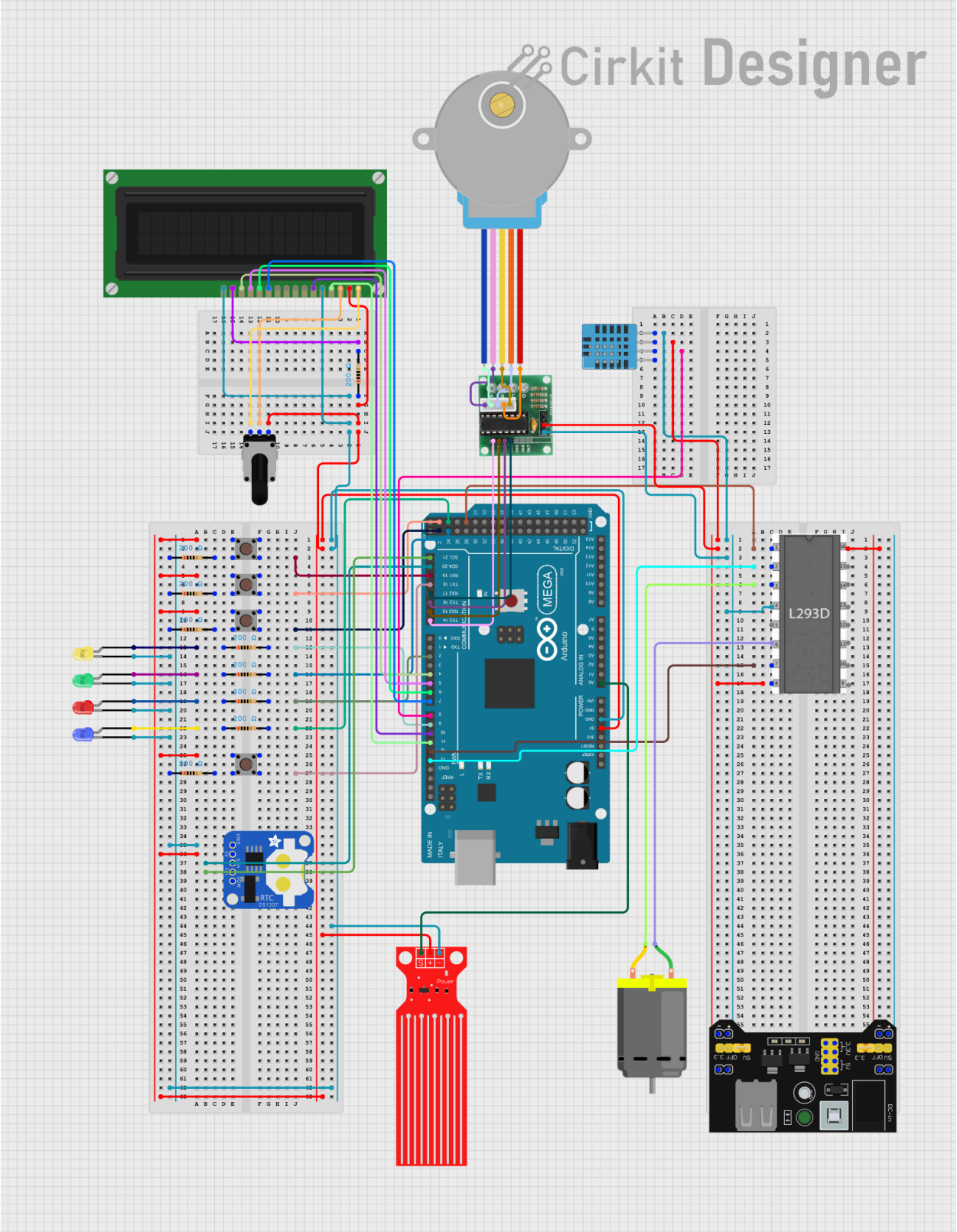


Figure 12: Schematic Diagram

Specification Sheets:
Arduino Atmega 1250 2560
https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

DHT11 Temperature and Humidity Sensor
https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

Stepper Motor 28 BYJ-48
https://components101.com/motors/28byj-48-stepper-motor

ULN2003 Stepper Motor Driver Board
https://www.electronicoscaldas.com/datasheet/ULN2003A-PCB.pdf

DS1307 RTC
https://www.analog.com/en/products/ds1307.html

L293D Motor Driver for DC Motor
https://www.ti.com/lit/ds/symlink/l293.pdf

LCD1602a
https://www.openhacks.com/uploadsproductos/eone-1602a1.pdf

MB102 Breadboard Power Supply Module 3.3V/5V
https://components101.com/modules/5v-mb102-breadboard-power-supply-module

GitHub Repository:
https://github.com/sosnowskih/CPE301_Group7

Video of Operation:
https://www.youtube.com/watch?v=TIdDzs4NjS4