2019

Automatic Room Temperature Controller

CONTROLS, OUTPUT FROM INPUT WITH A FEEDBACK THAT OUTPUT IS OUT OF RANGE

UMAIR MUGHAL

Abstract

The design of automatic room temperature is simple, it uses a temperature sensor to detect the temperature of room and turns on appliances in accordance with temperature. However we add different stuff and program it in microcontroller to make it better in real world use. Like these appliances are not only dependent on the temperature of room, but on a motion sensor that gives signal to 'AND' gate to confirm that there is need of turning these appliances.

DESCRIPTION

General room temperature sensors, we are using are mostly used to control temperature only. But in our case we used other sensors and parameters to use them in real life (like time dependence), however there are some bugs in programming need to be fixed. As, we are building a project within the limits of control theory, so this report explains the idea of the working principle of project in accordance with the control theory. This is a close loop system as one can understand by considering the whole system in big frame. Because in general, one can imagine our system as open loop by taking input and apply a function on it, and turns on heater or air conditioner. The details for the close system are discussed in next section.

Acknowledgments

The project is basically designed in less time as required, this is because of some misunderstanding and issues in semester work. However, after designing this project in a hard time, I'm thankful to all of members who help me to perform this task completely. Especially, I'm thankful to Sajjad-ul-Haq by creating an environment of work to take it completion. I'm also thankful to Ahmad Nawaz for taking work serious, and designing a structure for project.

UMAIR

INTENIONALLY BLANK

TABLE OF CONTENTS

CHAPTER NO. 1

BASIC PRINCIPLE AND IDEA

As, we briefly discussed in introduction, that this system is close loop system. One can imagine it as a system that takes temperature through sensor from output (room), check it with reference value and gives the input in the form of error to function for actions. The controller in this case to control the signal conditions is PID. Before going into details first we need to see the brief details of controller, because they play an important role for applying conditions (in general sense) to a system one want to get control on it.

HOW IT'S A CONTROL SYSTEM

Automatic room temperature controller consist of a sensor that measures temperature of room, we used LM335 because it's more accurate than its last models, and however it increases the cost of system. This data about component is given in document appendix. The figure 1 below shows the basic working principle of our system. Simply it takes input from temperature sensor and gives output to heater and AC by controlling them. There's a LCD that shows temperature and time and thus these appliances and room control is dependent on time. We can turn of the brightness of LCD by turning of its LED+ (16^{th}) pin.

CONTROLLERS

As the name implies, a temperature controller is an instrument used to control temperatures, mainly without extensive operator involvement. A controller in a temperature control system will accept a temperature sensor such as a thermocouple or RTD as input and compare the actual temperature to the desired control temperature, or set point. It will then provide an output to a control element.

A good example would be an application where the controller takes an input from a temperature sensor and has an output that is connected to a control element such as a heater or fan. The controller is usually just one part of a temperature control system, and the whole system should be analyzed and considered in selecting the proper controller.

ON/OFF CONTROL

An on-off controller is the simplest form of temperature control device. The output from the device is either on or off, with no middle state. An on-off controller will switch the output only when the temperature crosses the set point. For heating control, the output is on when the temperature is below the set point, and off above set point. Since the temperature crosses the set point to change the output state, the process temperature will be cycling continually, going from below set point to above, and back below. In cases where this cycling occurs rapidly, and to prevent damage to contactors and valves, an on-off differential, or "hysteresis," is added to the controller operations. This differential requires that the temperature exceed set point by a certain amount before the output will turn off or on again. On-off differential prevents the output from "chattering" or making fast, continual switches if the cycling above and below the set point occurs very rapidly.

On-off control is usually used where a precise control is not necessary, in systems which cannot handle having the energy turned on and off frequently, where the mass of the system is so great that temperatures change extremely slowly, or for a temperature alarm. One special type of on-off control used for alarm is a limit controller. This controller uses a latching relay, which must be manually reset, and is used to shut down a process when a certain temperature is reached.

PROPORTIONAL CONTROL

Proportional controls are designed to eliminate the cycling associated with on-off control. A proportional controller decreases the average power supplied to the heater as the temperature approaches set point. This has the effect of slowing down the heater so that it will not overshoot the set point, but will approach the set point and maintain a stable temperature. This proportioning action can be accomplished by turning the output on and off for short time intervals. This "time proportioning" varies the ratio of "on" time to "off" time to control the temperature. The proportioning action occurs within a "proportional band" around the set point temperature. Outside this band, the controller functions as an on-off unit, with the output either fully on (below the band) or fully off (above the band). However, within the band, the output is turned on and off in the ratio of the measurement difference from the set point. At the set point (the midpoint of the proportional band), the output on: off ratio is 1:1; that is, the on-time and off-time are equal. if the temperature is further from the set point, the on- and off-times vary in proportion to the temperature difference. If the temperature is below set point, the output will be on longer; if the temperature is too high, the output will be off longer.

PID CONTROL

The third controller type provides proportional with integral and derivative control, or PID. This controller combines proportional control with two additional adjustments, which helps the unit automatically compensate for changes in the system. These adjustments, integral and derivative, are expressed in time-based units; they are also referred to by their reciprocals, RESET and RATE, respectively. The proportional, integral and derivative terms must be individually adjusted or "tuned" to a particular system using trial and error. It provides the most accurate and stable control of the three controller types, and is best used in systems which have a relatively small mass, those which react quickly to changes in the energy added to the process. It is recommended in systems where the load changes often and the controller is expected to compensate automatically due to frequent changes in set point, the amount of energy available, or the mass to be controlled. OMEGA offers a number of controllers that automatically tune themselves. These are known as auto tune controllers.

OVERVIEW OF SYSTEM

Figure below shows the project diagram in simplest way. In hardware form we not use much circuit elements for designing logics. These logics are created in Arduino whose detail is shows in Programming section of this document. As, one can see we used Arduino UNO for simplicity. Specifications of this Arduino is given in appendix, it fits all on all the required parameters.

Generally temperature controller logic is so simple with the equipment in this modern rein, even with the use of simple diodes and 'AND' gates we can built it. But this design which is given in figure 2, can only be used for temperature control, conditions and other parameters are difficult to design for the development procedures. So, we want a system with a platform that gives us a better way of designing, so that we can develop it for later use. These future development ideas are given in Development chapter.

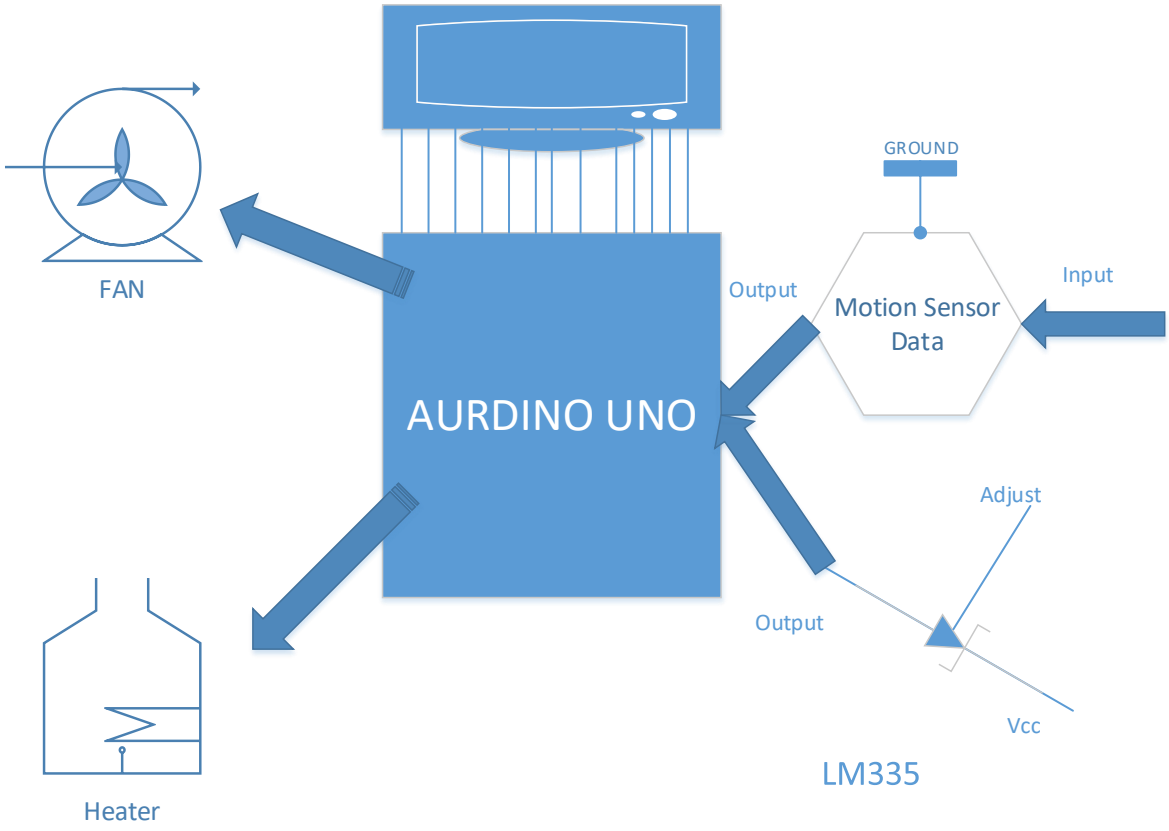

Figure 1: Visio Drawing

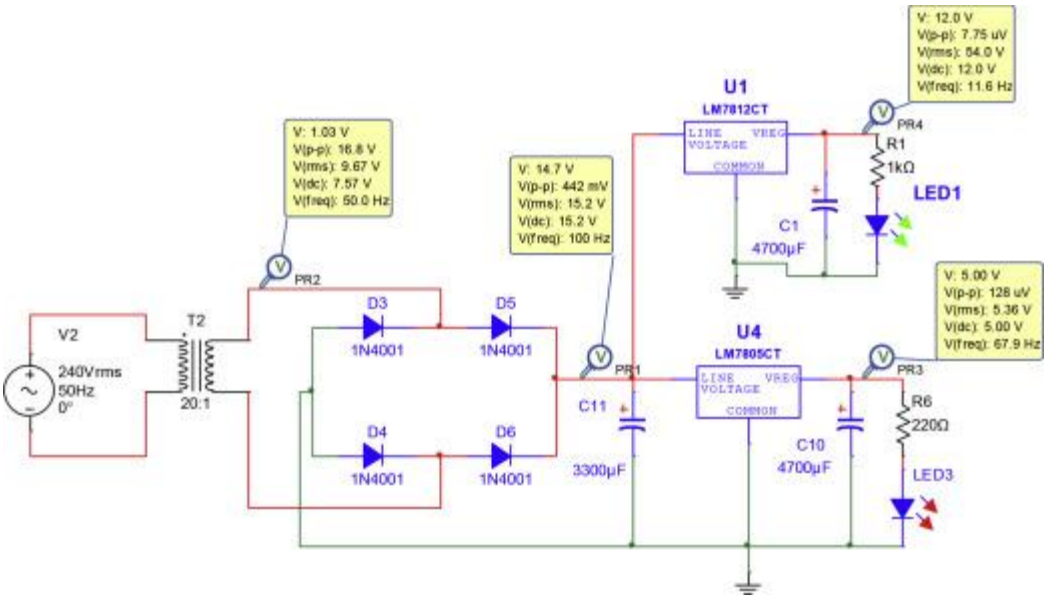

Figure 2: Using Simple Circuit analysis

BLOCK DIAGRAM FORMATION

To design a general block diagram before defining their parameters and defining the transfer function, we simply need to observe how our system is behaving as control system. What could be the input and how output causes change in input? After the above discussion about the basic principle behind our system, now it's not a difficult task to design a block diagram without defining the parameters. The parameters are defined in chapter 2, where we will observe how to design a transfer function by using mathematical modelling techniques using physics laws.

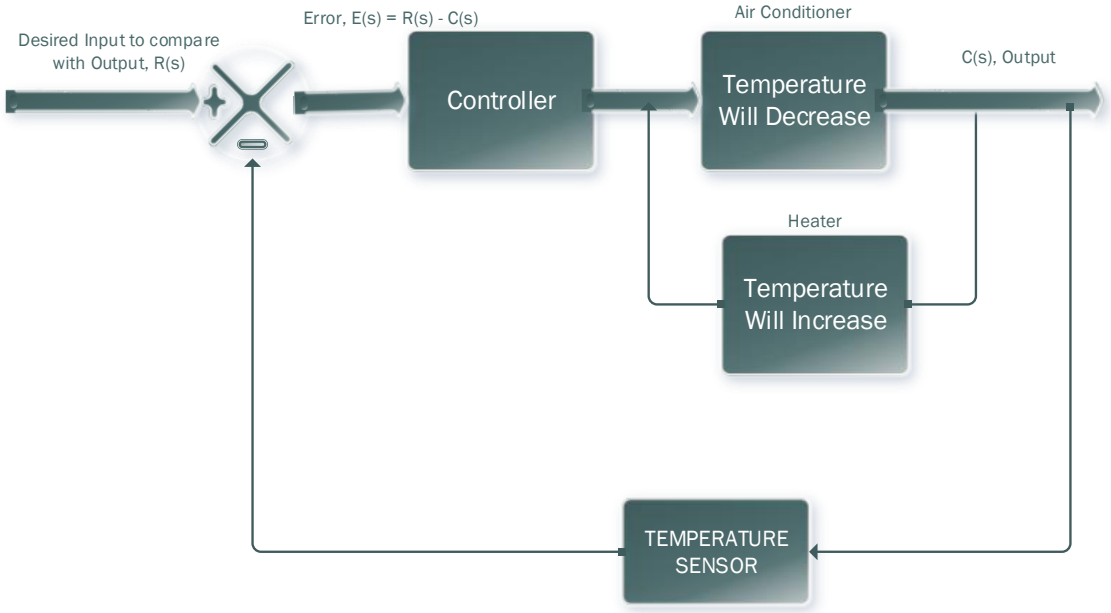

Figure 3: General Block Diagram

So, in here we have two blocks and a direct line. These blocks causes change in temperature which is decided by controller. Generally if room temperature is less then, error is high means we need to increase our temperature.

But this diagram not works because in general with the increase in temperature, we only want to perform decrease in temperature. This is happening because we have different desired inputs. Like to run AC limit in program is less than 20, and for heater it's greater than 30. So, in the case we use MIMO system. Because modern control system is out of scope of this course, so in this case we'll develop two block diagrams; one for increasing temperature and one for decreasing.

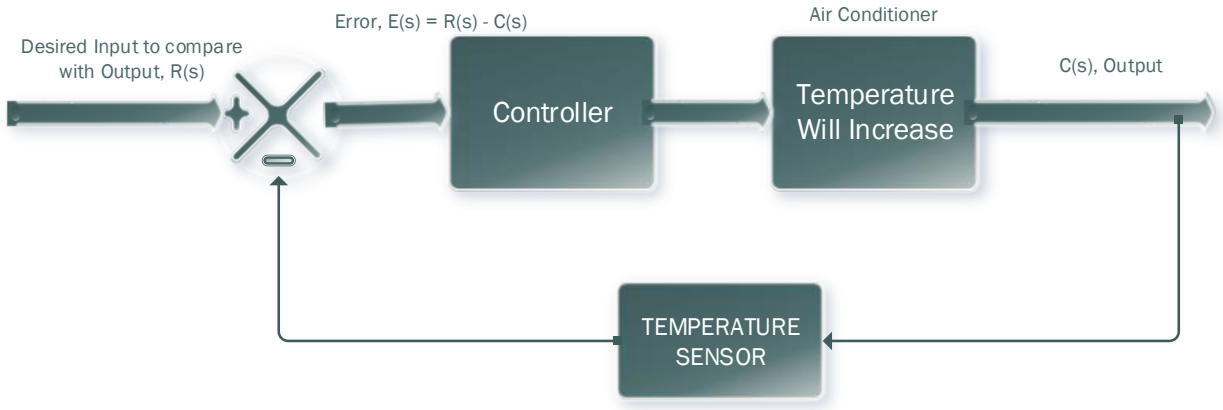

Figure 4: Block Diagram (a)

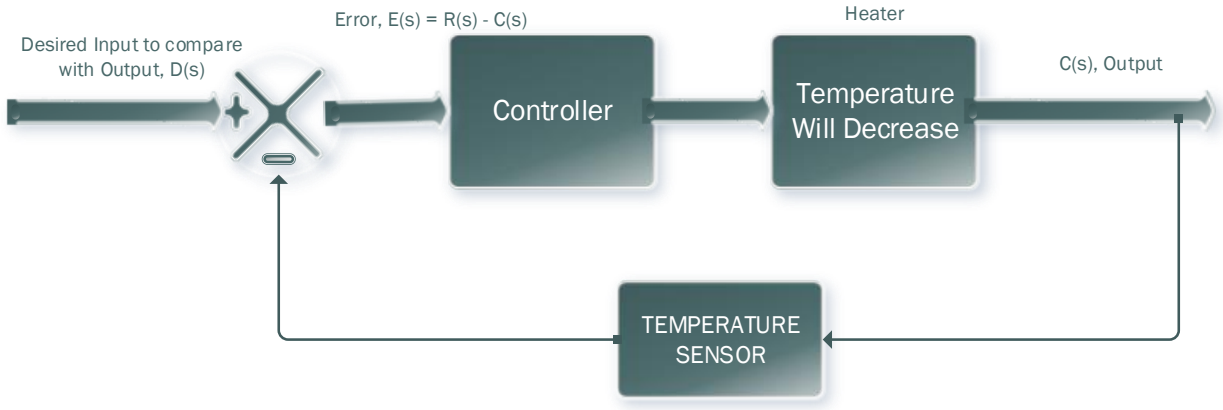

Figure 5: Block Diagram (b)

In the case of (a) we want to compare the output (which is less) with the desired input value R(s). In this case we have the less temperature of room, and we want to compare it with the limit of 30, in such a way that if temperature will exceeds above this limit (>30), then it causes function (air conditioner) to turn on to increase the temperature. As, this value of temperature should be higher to perform function, so in this case we have negative error. So, now we can understand the reason of using two block diagrams, because one can imagine the opposite behavior of other diagram with respect to this.

In the same way we have the second case in which we have high value of temperature measured by the sensor. This value is then compared with the desired value which we had set 20. So, when temperature of room moved towards the value less than limit, we will have the positive error because of negative feedback path. So, we can conclude following points from this:

- When E(s) < 0: Temperature will increase
- When E(s) > 0: Temperature will decrease

One could have misconception from its working behavior in a sense that why we are taking two blocks for a single system. But the basic reason behind this is to avoid the complexity of MIMO systems.

We can join both block diagrams into signal one which is shown below. This representation is choose for simplified view.

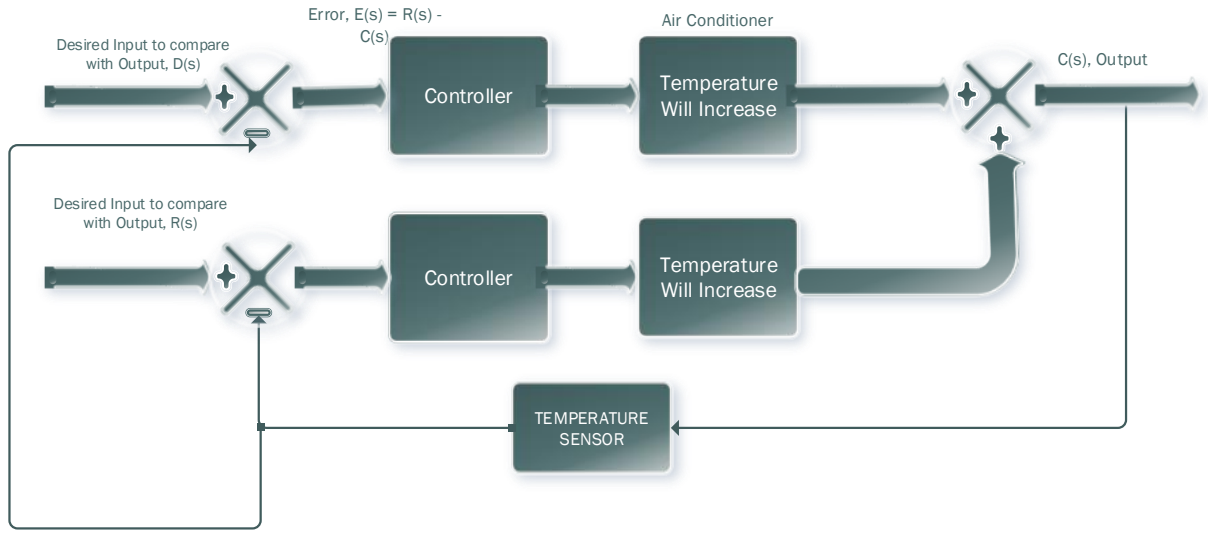

Figure 6: combined block diagram

RANGE TEMPERATURE (ASSUMPTION)

A general system is designed on the basis of basic structure. In this structure we have two blocks; one is for increasing temperature and other is for decreasing (in general two systems). For first system when temperature exceeds 30◦C, then we have negative error, and it perform function to make system stable, and vice versa for other.

Now, as we also have the temperature in between these two limits (20 or 30), then what we need to do for that limit. As, we have two different systems instead of one, so to maintain this we have two ways:

- Another model
- Neglecting

In this case we are neglecting this. Nothing worse happens by doing this because increment of another model causes complexity for this basic design. If we neglect this factor then our transfer

function limits for only two systems (increasing and decreasing temperature systems), which will gives the values as required for these two systems.

CHAPTER 2:

ELECTRICAL AND MECHANICAL MODEL DESIGNING

In this chapter we will generalize the idea of designing electrical model of system. Before designing the model we will first see the some parameters requires to design the best model. Then in this chapter, we'll see the modelling simulation on different softwares, and observe the system behavior.

FACTORS REQUIRE TO MODEL A SYSTEM

If one knows the skills to do something, he could complete it, but probably not in the same manner like the one who knows the skill and do after planning. Engineering things require creativity and technical stuff, so unlike patterned works, it's not possible to proceed a task to completion before planning. In the ancient age, we had an amazing tool to plan the engineering work, mathematics. One after developing the equations of a system which one observed from nature, one could predict that what will happen next to system (like what will happen next after changing the parameters of system). Now, we have more advanced form of mathematical tools in which we can design models to predict their behavior and prevent things from practical before proceeding, which are called as simulators. So, we have to design the simulations of our system to predict its behavior.

In practical view of a system modeling, it's not simple to predict the behavior and act accordingly, so there's always the need of managing the practical stuff.

ELECTRICAL MODEL DESIGNING

In the model we are going to operate two tasks using programming which will be discussed later.

- To show time on LCD
- Measure temperature and keep updating it

USED COMPONENTS

We used the following components to design the circuit model:

1. Two way switch
2. Arduino UNO
3. Breadboard
4. Potentiometer (0-100kΩ)
5. Resistors (2kΩ & 220Ω)
6. LCD (16,2) → JHD162A
7. Temperature sensor → LM335
8. Amplifier → LM741

9. Motion Sensor
10. AND Gate IC → 7408
11. Fan (In figure it's represented by LED attached to pin 8 of Arduino)
12. Heater (In figure it's represented by LED attached to pin 9 of Arduino)
13. Wires (M-M & M-F)
14. Breadboard

DETAILS

The model is represented in figure 6, which is drawn on Fritzing. It shows an LCD which is connected with Arduino and share data via 4 data pins. Also, we have 5V supply at the terminal line of breadboard. All the operations are performing on these 5V. The temperature sensor LM335 is used to measure atmospheric temperature. We have the potentiometer to vary the contrast of LCD, the resistance of potentiometer ranges to 100Ω, and is connected to the third pin (V_0) of LCD. In this model the two LEDs are the representation of heater and fan, which turns on when error occurs. The way switch is used to switch between time and temperature. Here, schematics and breadboard model are given below:

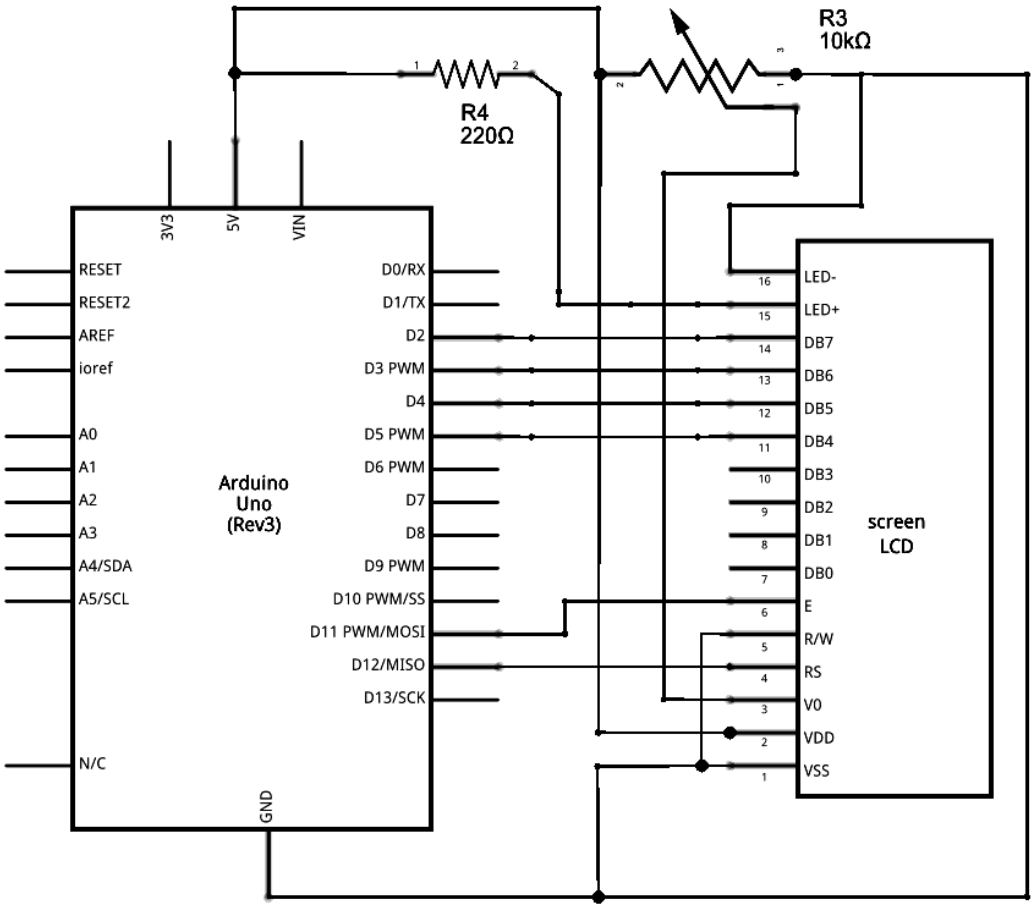

Figure 7: Schematics

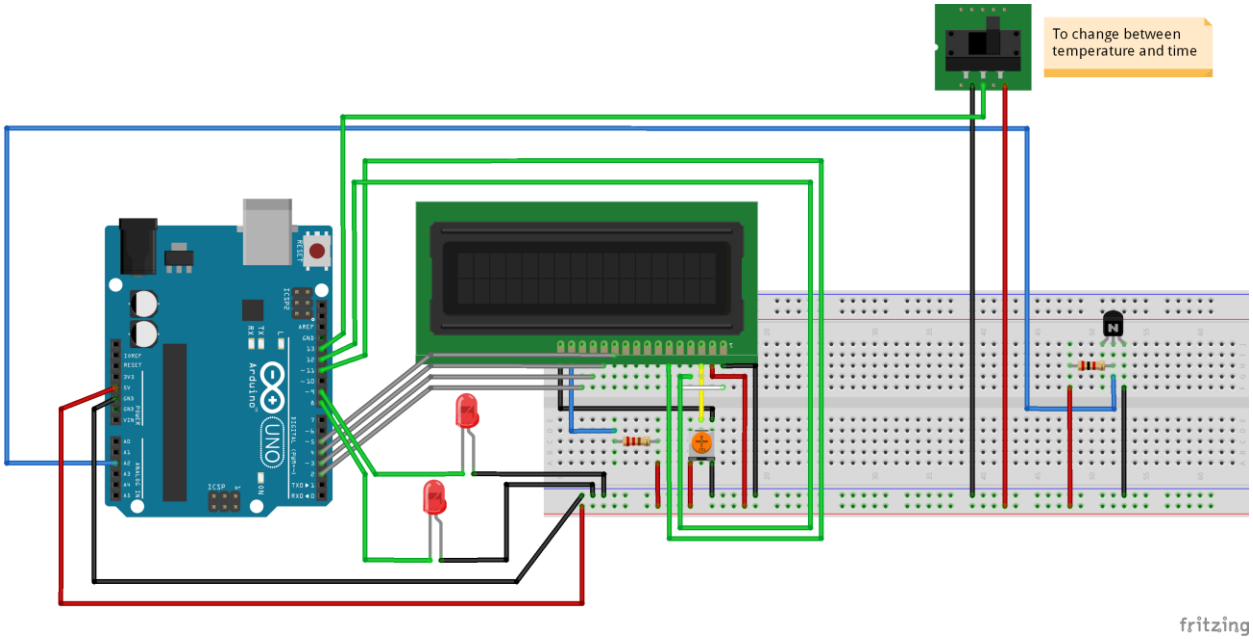

Figure 8: Circuit model

MODEL DESIGNING OF EXTRENAL CIRCUIT

For the external circuit we need to take a motion sensor, and a 'AND' gate and an amplifier. By simply connection of amplifier with 12V relay and then to the heater we can easily perform work. Here, we have the proteus diagram.

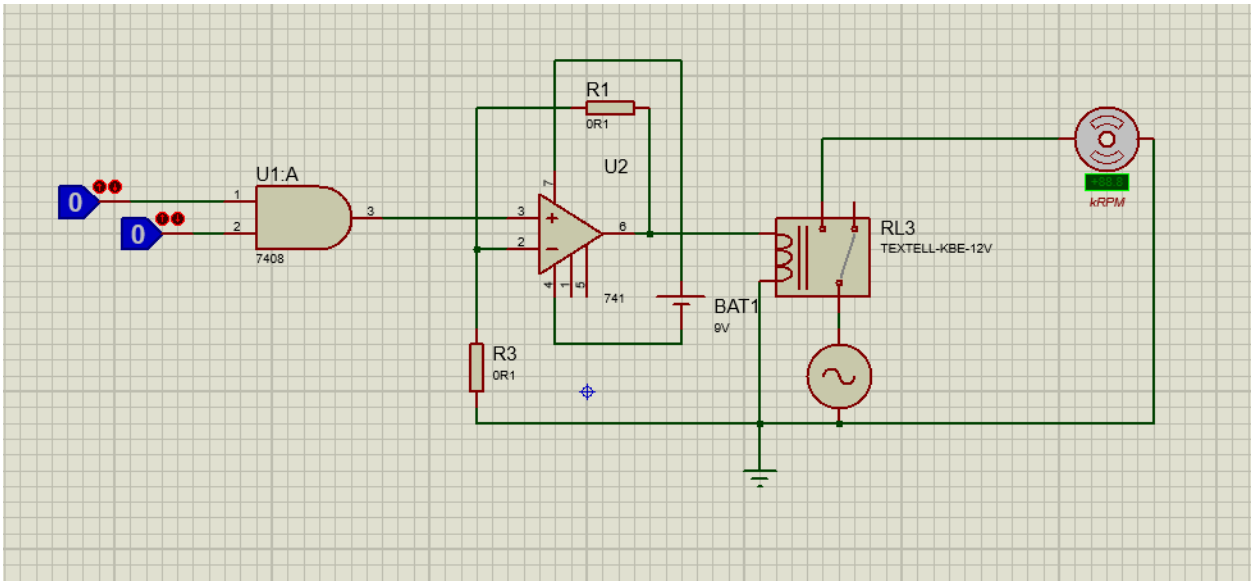

Figure 9: Proteus Amplifier circuit

We used this circuit to use components at high voltage, like practical heater and compressor. However we not attached this in project, but one can easily attach compressor or heater or both with terminals leaving. In above diagram we attached a fan. In our practical circuit these terminals are opened.

The PCD Design of that circuit is given below:

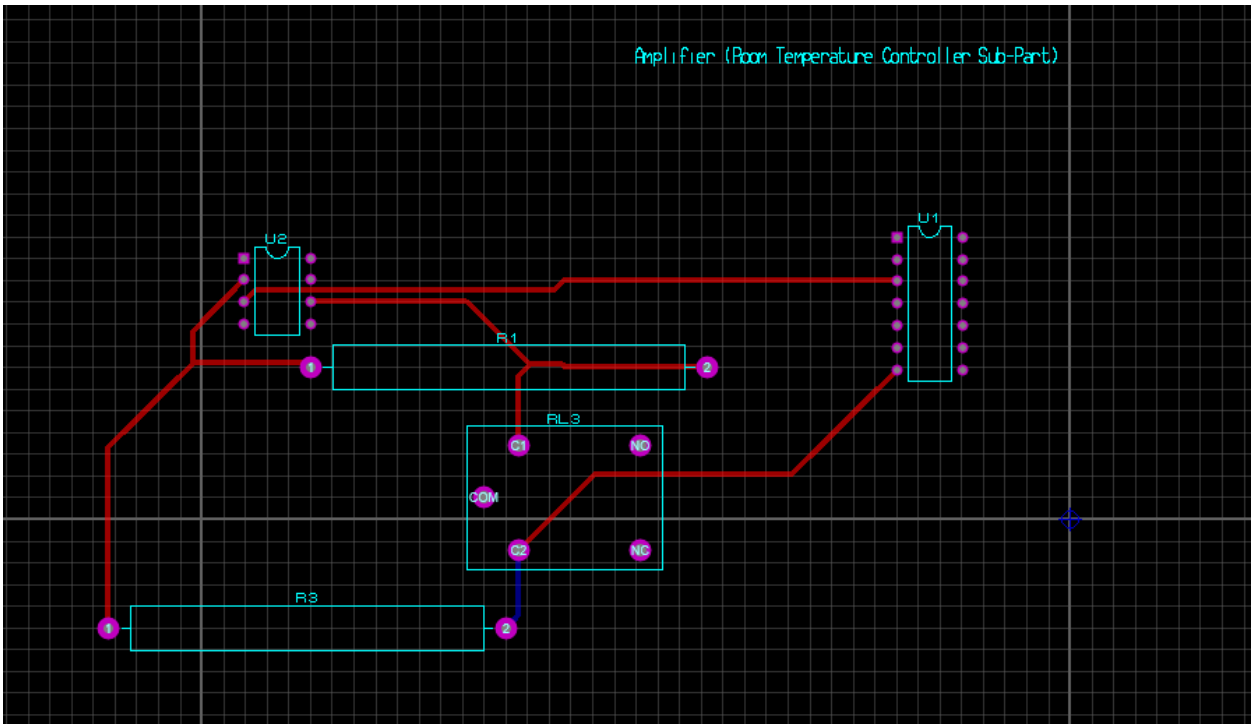

Figure 10: PCB of amplifier circuit

SIMULATIONS

Here, we are going to draw our system on simulation tool proteus. The figure 7, shows the system in simulated environment. In this case we have general stuff, like LCD and Arduino. Libraries of these components are not available in Arduino by default so we can download it from any related website and insert them in proteus.

The logic state on the left side is used as a switch, to switch between temperature and time. The simulation video is given in link (https://www.youtube.com/watch?v=L7zUO-b30hU); we also added the simulation results in results heading below. The procedure that how this system works is already explained above. So, after simulating the model designed above we conclude that the model is right and we can get the required results. So, prediction before practical gives us a quick way to show if we are following the right path or not.

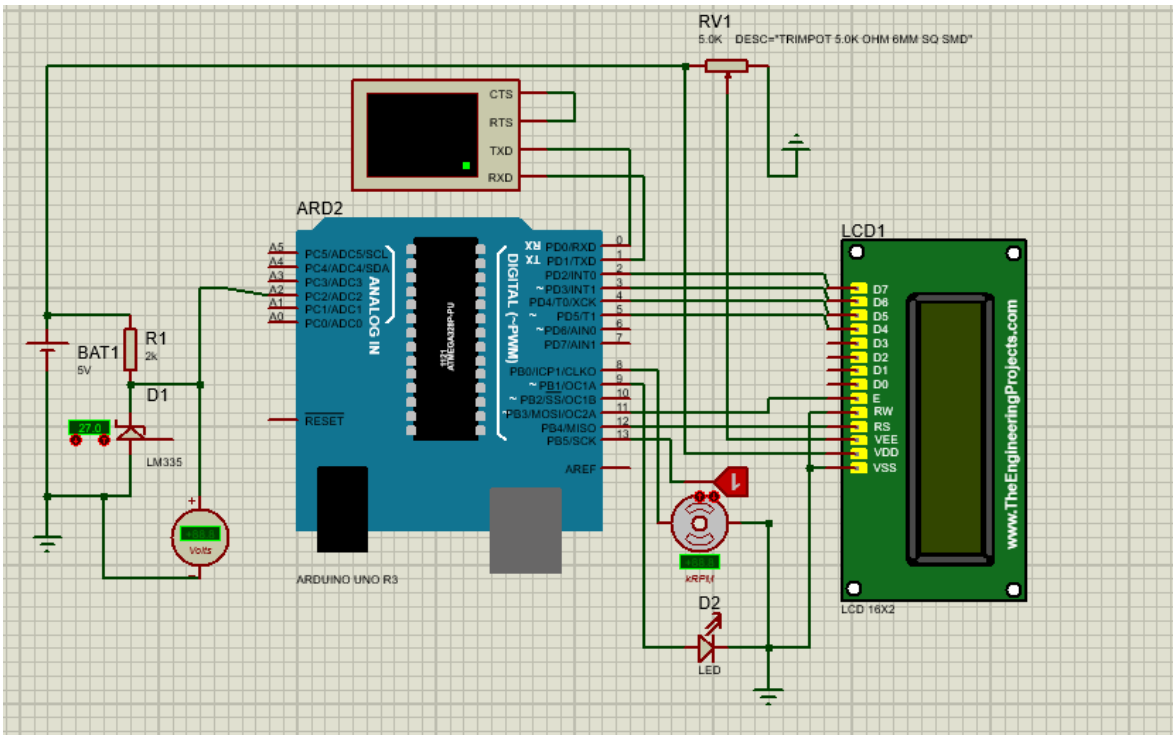

Figure 11: Proteus Simulation

RESULTS

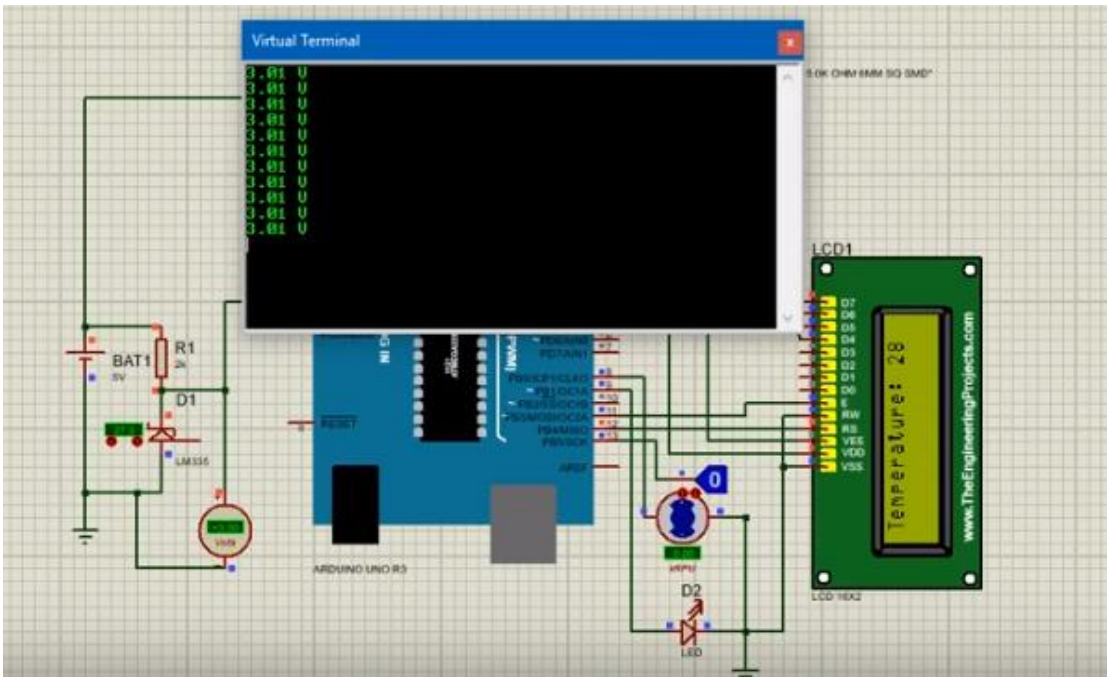

Figure 12: Temperature View

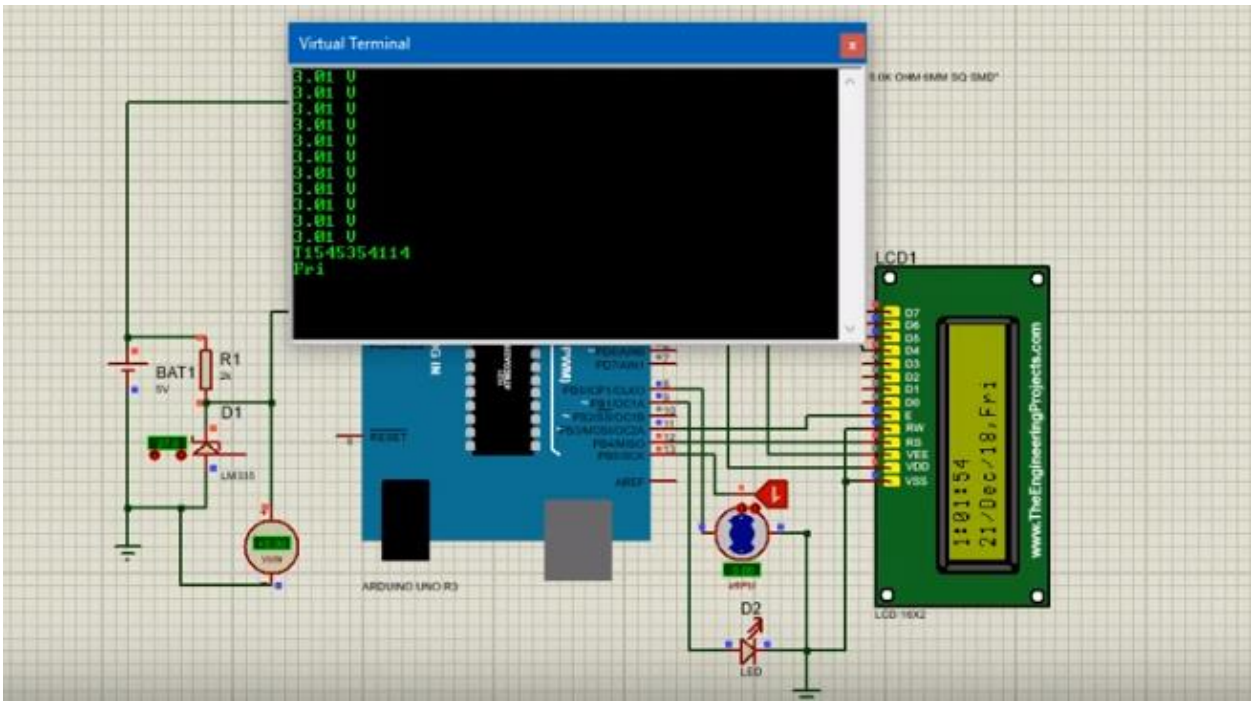

Figure 13: Time view

NOTE: To give input we are using the virtual terminal of proteus.

MECHANICAL MODEL DESIGNING

In this section we are going to overview the idea of mechanical designing. Actually, the planning in this was already decided, but makes it to completion was a difficult task. Now, we complete the project with general designing, which was not in plan. But later we'll develop our mechanical model as we planned before. These developments are given in 'Development' chapter.

The basic modelling of design was not done on any tool, and thus it's not done as predicted. Here, we are going to discuss the present mechanical design of structure. This design is given in figure below:

PLACE FIGURE

Inside of system of system could be called as un-arranged system? We try our best to fix all the stuff with the material we had.

CHAPTER 3:

CODING ON ARDUINO

In this chapter we'll observe the logics behind the circuit. A general model of temperature control is easy, but building a platform for epoch time was not a child play for those who are working on Arduino for the first time.
Briefly, in this chapter we'll observe how we build the logical operations to perform operations.

LCD CONTROL

To control LCD we need to send specific signals to LCD pins that perform different actions. As, shown in above figure 7, that shows the used data pins from which signal is coming from Arduino. The above two pins are contrast control pins, and below six pins are for turning LCD on and for management while using controller.

We used a library Liquid Crystal of Arduino to perform this. And add functions like scrolling, contrast control in it using code. Setup of LCD is given by below statements:

```
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
```

LCD SCROLL

```
void LCD_Scroll(int n) { //n = no. of chars
  //For single Line mutiply n by 2, in this case I use clear()
  delay(2000);
  for (int pos = 0; pos < n; pos++) {
    // scroll one position left:
    lcd.scrollDisplayLeft();
    delay(500);
  }
  lcd.clear();
}
```

TEMPERATURE CONTROL

To control temperature we need to perform some calculations, which are given in below code in tempM function. But before performing calculations we need data from sensor. Our sensor given voltage that are equivalent to temperature. In Appendix one can see the data sheet of LM335. But after attaching the output one can observe that the Arduino was not receiving voltage, but receiving some signals. So, first of all we need to change these signals to voltage value, the code of that conversion is given below:

SIGNAL TO VOLTAGE CONVERSION

```
#define NUM_SAMPLES 10

int sum = 0;                // sum of samples taken
unsigned char sample_count = 0; // current sample number
float voltage = 0.0;        // calculated voltage

void setup() {
  Serial.begin(9600);
}
void loop() {
  while (sample_count < NUM_SAMPLES) {
    sum += analogRead(A2);
    sample_count++;
    delay(10);
  }
  voltage = ((float)sum / (float)NUM_SAMPLES * 5.015) / 1024.0;
  Serial.print(voltage);
  Serial.println (" V");
  sample_count = 0;
  sum = 0;
}
```

[1]

TEMPERATURE MEASURMENT

The merged function of taking voltage of value and convert it from voltage to Celsius is given below:

```
int TempM() {              //Temperature Measurement

  //First Find Voltages
  while (sample_count < NUM_SAMPLES) {
    sum += analogRead(A2);
    sample_count++;
    delay(10);
  }
  voltage = ((float)sum / (float)NUM_SAMPLES * 5.015) / 1024.0;
  Serial.print(voltage);
  Serial.println (" V");
  sample_count = 0;
  sum = 0;

  tempOV = voltage * 100;          //In Kelvins
  tempOV = tempOV - 273;           //In Celcius
  return tempOV;

}
```

After this we made some functions for printing on LCD, and showing accurate values (by taking large observations and taking average). This can be observed in final code given below:

TIME MEASUREMENT

Here, a simple code is present that shows how we create logics to show time on LCD.

REQUESTING TIME FOR GIVEN EPOCH VALUE

We used a built in library for this called "timeLib", the two function code for epoch converter by requesting value on Serial monitor is given below:

```
void processSyncMessage() {
  unsigned long pctime;
  const unsigned long DEFAULT_TIME = 1357041600; // Jan 1 2013

  if (Serial.find(TIME_HEADER)) {
    pctime = Serial.parseInt();
    if ( pctime >= DEFAULT_TIME) { // check the integer is a valid time (greater than Jan 1 2013)
      setTime(pctime); // Sync Arduino clock to the time received on the serial port
    }
  }
}
time_t requestSync() {
  Serial.write(TIME_REQUEST);
  lcd.print("Add GMT Time");
  lcd.clear();
  return 0; // the time will be sent later in response to serial mesg
}

void printDigits(int digits) {
  // utility function for digital clock display: prints preceding colon and leading 0
  lcd.print(":");
  if (digits < 10)
    lcd.print('0');
  lcd.print(digits);
}
```

TIME MEASUREMENTS

LEAP YEAR

```
int leapV = 0;
int Leap(int y) {   //Input year
  if (y % 4 == 0 && y % 400 == 0) {
    //Year is leap
    leapV = 1;
    if (y % 100 == 0) {
      //Year is not leap
      leapV = 0;
    }
  }
  else {
    //Year is not leap
    leapV = 0;
  }
  return leapV;
}
```

WEEKDAY AND MONTH CONVERSION CALCULATIONS

To convert date to weekday we used Gauss's algorithm with some other calculations. [2]

Gauss's method gives us the first day of week by giving values from 0-6, in which '0' shows Sunday and this sequence goes on with increment of each number. So, after this, we need to take each month day and calculate our weekday in according. We should also need to take care of leap year. To this we built the following logic.

```
int R(int A, int B) {
  return A % B;
}
int monthDays[13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
String daysName[7] = {"Sun", "Mon", "Tue", "Wed", "Thurs", "Fri", "Sat"};
String
monthsName[12] = {"Jan", "Feb", "Mar", "Apr", "May", "June", "July", "Aug", "Sep", "Oct", "Nov", "Dec"
};

int SumMonth(int n) { //n = no. of times you want to sum = month()-2 (-1 for array)
  int sum = 0, store = 0;
  for (int i = 0; i < n; i++) { //A month less than given
    sum = store + monthDays[i];
    store = sum;
  }
  return sum;
```

```
}

int WEEKDAY(int d, int m, int y) {
  //1st January weekday is:
  //https://en.wikipedia.org/wiki/Determination_of_the_day_of_the_week (Guass Method)
  int weekdayV = R(1 + R(y - 1, 4) * 5 + R(y - 1, 100) * 4 + R(y - 1, 400) * 6 , 7);
  int L = Leap(y);
  if (L == 1) { //Year is leap
    weekdayV = R(weekdayV + R(SumMonth(m) + d + 1, 7), 7) - 1;
  }
  if (L == 0) {
    weekdayV = R(weekdayV + R(SumMonth(m) + d, 7), 7) - 1;
  }
  return weekdayV;
}

String MONTHDAY(int m) {
  return monthsName[m - 1];
}
```

After this we used printing function to finalize it.

EEPROM INSERTION

EEPROM is used to take time data which user put in Serial monitor and save it in a ROM, so that we can access it later and user will not need to enter it again after tuning the device again.

In this case we not used any crystal oscillator for continuous time processing, so after adding it the time will never stop. So, this function is not much beneficial in this case.

CODE

We used EEPROM library for this.

```
void EEPROM_Write() {
  String val = Serial.readString();
  for (int i = 0; i < 11; i++) {
    EEPROM.write(addr, val.charAt(addr));
    addr++;
  }
  addr = 0;
}

void EEPROM_Read() {
  //Remember to put this function after Serial.available()
  lastStore[0] = 'T';
  for (int i = 0; i < EEPROM.length(); i++) {
```

```
   lastStore[i] = EEPROM.read(i);
   Serial.print(lastStore[i]);
  }
}
```

FINAL WORKING CODE

```
#include <LiquidCrystal.h>
#include <TimeLib.h>
#include <EEPROM.h>
#define NUM_SAMPLES 10

int tempL_A = 30, tempL_H
= 25;     //Temperature limits

int outputP_AC = 8;
int outputP_HT = 9;

int timeP = 13;      //To check time (INPUT)
//Temperature Output Value, 2.7315V per
273.15k
float tempOV = 0;     //No voltage at start

//_____________________________________
int sum = 0;                // sum of samples
taken
unsigned char sample_count = 0; // current
sample number
float voltage = 0.0;          // calculated
voltage

//_____________LCD INIT_____________

const int rs = 12, en = 11, d4 = 5, d5 = 4, d6
= 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

//_____________Setup_____________
void setup() {
 Serial.begin(9600);
 pinMode(outputP_AC, OUTPUT);
 pinMode(outputP_HT, OUTPUT);

 //_____________LCD_SETUP_____________
 lcd.begin(16, 2);
 // Print Message to LCD
 lcd.print("____Welcome!____");
 delay(3000);
 lcd.clear();

 //_____________TIME_SETUP_____________
 setSyncProvider( requestSync);

 //_____________TIME INIT_____________
#define TIME_HEADER "T"
#define TIME_REQUEST 7

}
int TempM() {          //Temperature
Measurement

 //First Find Voltages
 while (sample_count < NUM_SAMPLES) {
  sum += analogRead(A2);
  sample_count++;
  delay(10);
 }
 voltage = ((float)sum
/ (float)NUM_SAMPLES * 5.015) / 1024.0;
 Serial.print(voltage);
 Serial.println (" V");
 sample_count = 0;
 sum = 0;

 tempOV = voltage * 100;       //In Kelvins
 tempOV = tempOV - 273;        //In Celcius
 return tempOV;
}

int avg = 0, count1 = 0;
void LCD_Scroll(int n) { //n = no. of chars
 //For single Line mutiply n by 2, in this case I
use clear()
 delay(2000);
 for (int pos = 0; pos < n; pos++) {
```

```
    // scroll one position left:
    lcd.scrollDisplayLeft();
    delay(250);
  }
  lcd.clear();
}

//You can Define more to print
String str = " ", str1 = "Temperature: ";
void LCD_Print() {
  String tempStr = String(avg , DEC);
  String str = String(str1 + tempStr);
  lcd.print(str);

  //ADD OTHER PRINTS

  LCD_Scroll(str.length());
}

char lastStore[11] = {0};
void EEPROM_Read() {
  //Remember to put this function after
Serial.available()
  lastStore[0] = 'T';
  for (int i = 0; i < EEPROM.length(); i++) {
    lastStore[i] = EEPROM.read(i);
    Serial.print(lastStore[i]);
  }
}

int count_ = 0;
void loop() {

  bool timeR = digitalRead(timeP);    //Time
Read
  if (timeR == true) {

    if (Serial.available()) {
      lcd.clear();
      processSyncMessage();
    }

    if (timeStatus() != timeNotSet) {
      digitalClockDisplay();
    }
    delay(1000);

    // set the cursor to column 0, line 1
    // (note: line 1 is the second row, since
counting begins with 0):
    lcd.setCursor(0, 0);
  }
  else {
    //Average Temperature Loop
    lcd.clear();
    tempOV = TempM();
    int store = tempOV, n = 10;    //'n' = no. of
samples for avg
    for (int i = 0; i < n; i++) {
      tempOV = TempM();
      avg = (store + tempOV) / 2;
      store = avg;
      delay(100);   //10 * 100 = 1sec (FOR
NOW)
    }

    if (tempOV > tempL_A) {
      digitalWrite(outputP_HT, LOW);
      digitalWrite(outputP_AC, HIGH);   //Turn
on AC
    }
    if (tempOV < tempL_H) {
      digitalWrite(outputP_AC, LOW);
      digitalWrite(outputP_HT, HIGH);   //Turn
on Heater
    }
    if (tempOV < tempL_A && tempOV >
tempL_H) { //Turn off
      digitalWrite(outputP_AC, LOW);
      digitalWrite(outputP_HT, LOW);
    }

    //Print on LCD
    LCD_Print();
  }
}

int val = 0, addr = 0;
void EEPROM_Write() {
  String val = Serial.readString();
  for (int i = 0; i < 11; i++) {
    EEPROM.write(addr, val.charAt(addr));
    addr++;
```

```
  }
  addr = 0;
}

//________TIME_DETERMINATION_____________
void processSyncMessage() {
  unsigned long pctime;
  const unsigned long DEFAULT_TIME
= 1357041600; // Jan 1 2013

  if (Serial.find(TIME_HEADER)) {
    pctime = Serial.parseInt();
    if ( pctime >= DEFAULT_TIME) { // check
the integer is a valid time (greater than Jan
1 2013)
      setTime(pctime); // Sync Arduino clock
to the time received on the serial port
    }
  }
}


time_t requestSync() {
  Serial.write(TIME_REQUEST);
  lcd.print("Add GMT Time");
  lcd.clear();
  return 0; // the time will be sent later in
response to serial mesg
}

//_____________LEAP_____________________
int leapV = 0;
int Leap(int y) {   //Input year
  if (y % 4 == 0 && y % 400 == 0) {
    //Year is leap
    leapV = 1;
    if (y % 100 == 0) {
      //Year is not leap
      leapV = 0;
    }
  }
  else {
    //Year is not leap
    leapV = 0;
  }
  return leapV;
}

//___________ TIME CALCULATIONS___________
int R(int A, int B) {
  return A % B;
}
int
monthDays[13] = {0, 31, 28, 31, 30, 31, 30, 31,
31, 30, 31, 30, 31};
String
daysName[7] = {"Sun", "Mon", "Tue", "Wed", "T
hurs", "Fri", "Sat"};
String
monthsName[12] = {"Jan", "Feb", "Mar", "Apr",
 "May", "June", "July", "Aug", "Sep", "Oct", "Nov
", "Dec"};

int SumMonth(int n) { //n = no. of times you
want to sum = month()-2 (-1 for array)
  int sum = 0, store = 0;
  for (int i = 0; i < n; i++) { //A month less than
given
    sum = store + monthDays[i];
    store = sum;
  }
  return sum;
}

int WEEKDAY(int d, int m, int y) {
  //1st January weekday is:
  //https://en.wikipedia.org/wiki/Determinati
on_of_the_day_of_the_week (Guass
Method)
  int weekdayV = R(1 + R(y - 1, 4) * 5 + R(y
- 1, 100) * 4 + R(y - 1, 400) * 6 , 7);
  int L = Leap(y);

  if (L == 1) { //Year is leap
    weekdayV = R(weekdayV +
R(SumMonth(m) + d + 1, 7), 7) - 1;
  }
  if (L == 0) {
    weekdayV = R(weekdayV +
R(SumMonth(m) + d, 7), 7) - 1;
  }
```

```
  return weekdayV;
}

String MONTHDAY(int m) {
  return monthsName[m - 1];
}

//_____TIME_PRINT_____________________________
void digitalClockDisplay() {
  // digital clock display of the time

  lcd.print(hour()+5);
  printDigits(minute());
  printDigits(second());

  //  lcd.print(daysName[WEEKDAY(month(),
year())]);
  lcd.setCursor(0, 1);
  Serial.print(daysName[WEEKDAY(day(),
month(), year())]);
  Serial.println();
  lcd.print(day());
  lcd.print("/");
  lcd.print(MONTHDAY(month()));
  lcd.print("/");
  lcd.print(year() % 100);
  lcd.print(",");
  lcd.print(daysName[WEEKDAY(day(),
month(), year())]);
}

void printDigits(int digits) {
  // utility function for digital clock display:
prints preceding colon and leading 0
  lcd.print(":");
  if (digits < 10)
    lcd.print('0');
  lcd.print(digits);
}
```

CHAPTER 4:

MATHMEMATICAL MODELING AND TRANSFER FUNCTION REPRESENTATION OF SYSTEM

After developing the basic block diagram model of our system we need to model it mathematically.

TAKING VALUES FROM SENSOR (METHOD)

In the above case we had developed a function that shows the behavior of time with temperature all along the day. So, we can see how this function is behaving with respect to time. So, if atmospheric temperature increases from 6:00 - 15:00, and decrease other times (but not linearly), then we can observe how our output behaves in response to this input. It should move in opposite direction of input, i.e for increment it temperature it should decrease the temperature. So, we'll observe that how we develop the time dependent mathematical model of our system. In this section we'll see how we had observe can observe the values from sensor and relate it with the average weather graph, to show if it's behaving correctly or not. This can be done mathematically, but here can observe that how we could use it as practical case too.

In this project we haven't used compressor OR heater to change the output temperature of room, so in this section one can observe the method we developed to analyze the system practically. To do this we need to perform the following procedure:

- Observe the data of weather report
- Fit this data by using any polynomial fitting tool
- Observe the data room
- Fit this data
- Compare both data, and observe if system is working as required or not.

This process can also be used for testing. Otherwise one can also built a model from these observations, which are surely couldn't by accurate for the small sample space. But for large observations they will surely work out.

MODELLING USING MATHEMATICAL PARAMETERS

MODEL 1

In this model we will observe how our output should behave with the change in atmospheric temperature as time exceeds.

SEARCH FOR MODEL AND PLACE IT HERE

MODEL 2

We can have an alternate model to the last one that relates the behavior of our sensor temperature directly with the room temperature. It can be found in the video [3]. We not work out this model, however it could be more efficient than the last one.

At first we have temperature at t = 0, and we considered it as an initial temperature.

AT T=0

At this point out sensor Temperature and system temperature are equal, so:

$$T(0) = T_s(0)$$

If the room temperature raised then from heat balance equation, we have:

$$Rate\ of\ change\ of\ heat\ sensor = Rate\ of\ heat\ in - Rate\ of\ heat\ out$$

AT $T_R > T_S$

Now, consider our sensor temperature is greater than room temperature, it means our room temperature is low and sensor temperature is high. So, we have positive error. And we want to increase our room temperature.

$$T_r > T_s$$

So, for this condition:

$$W = UA(T_r - T_s) \qquad (i)$$

Here,

- U: Heat Constant
- A: Area

Now, change in temperature sensor is:

$$Rate\ of\ change\ of\ sensor\ temperature = MC(T_s - T_s(0)$$

$$Rate\ of\ change\ of\ sensor\ temperature = MC\left(\frac{d\Delta T}{dt}\right) \qquad (ii)$$

Solving equation (i) and (ii), we have: $UA\left(T_r - T_s\right) = MC\left(T_s - T_s(0)\right)$

$$\boxed{\Delta T_r - \Delta T_s = \frac{MC}{UA} \times \frac{d\Delta T_s}{dt}}$$

CHAPTER 6:

DEVELOPMENT

MOTION SENSOR SECURITY

The motion sensor is already attached to the circuit but we are also thinking of adding some more precise techniques for power saving and security, using this sensor. We can also use it as sensor that when a person enters room (direction inward), then sensors turn on the signal otherwise if the direction is in the other way (outward), it turns off the signal.

In this case we need to develop a circuit that turns on and off alternatively.

KEYBOARD CONTROL

We are thinking of adding keypad with LCD, to enter date and time manually. Otherwise one can also set limits of temperature from this. It's not a difficult task, because their already built library for keypad in Arduino. But we were failed to implement this.

BRIGHTNESS AND CONTRAST AUTO-CONTROL

Brightness control of LCD with respect to time. As time increases to night, the brightness of light increases and contrast decreases. Vice versa happens for the day case.

IR SENSOR CONTROL

With IR sensor we can wirelessly send signals to send temperature limits and time, this is what keyboard do. Otherwise it's somewhat complex, because in this case we need to manage the frequency of IR transmitter and need to synchronize it with receiver.

APPENDIX

COMPONENTS USED

LM335

LM741

MOTION SENSOR

ARDUINO UNO

7808—AND GATE

TEMPERATURE CALCULATIONS

Our sensor measures 10mV for a kelvin temperature. It means we have:

$$0.01V \rightarrow 1K$$

$$\Rightarrow \quad 3V \rightarrow \frac{1K}{0.01} \times 3$$

$$\Rightarrow \quad 3V \rightarrow 300K \rightarrow (300 - 273.15)^{\circ} C \approx 27^{\circ} C$$

We took the 3V reference for convenience. So, from this reference 0.01 (10mV) of voltage change causes a Celsius degree of change in temperature. Like we have 30°C temperature for 3.03V, and 25°C for 2.98V.