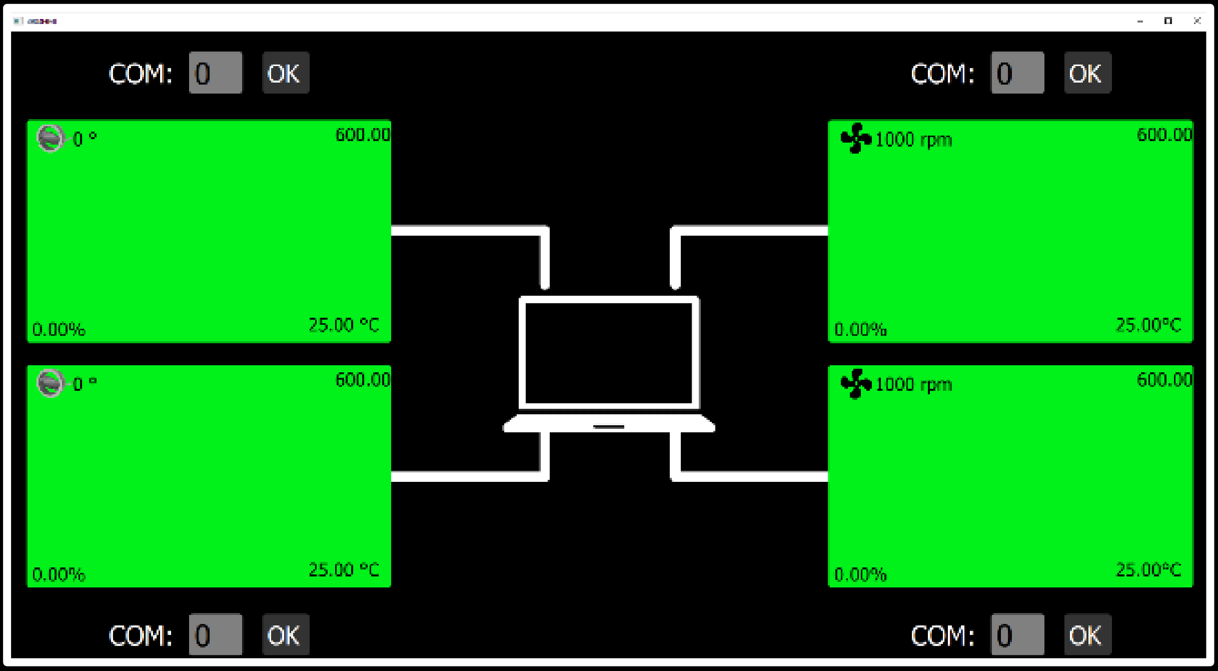
Smart Alfresco Cooling System

Distributed thermal sensors



Goh Kheng Xi, Jevan
(+65) 96453955
e0406787@u.nus.edu

Toh Wei Wen
Lock Mei Lin

ABSTRACT

This project aims to optimise the airflow for cooling indoor and Al Fresco dining areas. Temperature sensors are made of diodes connected in a Wheatstone bridge configuration, then distributed around both outdoor and indoor areas to get the temperature distribution. A central unit collects data from all indoor and outdoor sensors. When the sensor detects a temperature change, the cooling system will adjust cooling effect accordingly. For indoors, the cooling systems are dampers that varies their angle to control the airflow from a fixed cool air source (simulating a diffuser). For outdoors, the cooling systems are Direct Current (DC) fans that varies their speed according to the temperature. The actuators of cooling systems are driven by microcontrollers. Using this system, up to 29% of the energy required to power the DC fan at standard room conditions can be saved.

1. Executive Summary

1. Executive SummaryIn Singapore, Air Conditioning and Mechanical Ventilation (ACMV) systems are commonly installed to combat the tremendous tropical heat. On average, an ACMV system accounts for more than 40% of a building's energy consumption [1].

1.1 Objectives

This project proposed the installation of sensors to control centralised ACMV systems automatically and better optimise cost and energy efficiency.

However, as this system can also be integrated in an outdoor environment with decentralised cooling system, this project focuses on environments with both indoor and outdoor elements.

1.2 Current Situation in Indoor and Al Fresco Dining Areas

Today, indoor ACMV systems employ different types of VAV (Variable Air Volume) diffusers to optimize cooling efficiency by controlling the air flow into each room through dampers [2], while commercial variable speed fans (VSF) that adapt to the surrounding temperature (e.g., AirEffect) are used in outdoor areas.

However, for both scenarios, the air flow into a room and the fan speed often rely on a localized thermal sensor for adjustments. This might not be a good representation of environment, causing energy inefficiency and inconvenience [3]. Moreover, the setpoint temperature and fan speed requires manual adjustments by the user, making it inconvenient.

Sensor PerformanceThe design, workings and algorithm of the sensors are described in Appendix A1. The sensor system is relatively economical for its performance due to the low-cost materials used. Real time processing of the signals allows the sensor system to estimate temperatures with 99.2% confidence at 1℃ sensitivity and a respond of 100 ms. The results are based on operating temperatures of between 20℃ and 38℃.

2. Cooling Systems

For indoors, the cooling system is a damper that controls the cool air flow from the diffuser (simulated by an AC fan); for outdoors, the cooling system is a VSF. This section reviews the relevant considerations for both the indoor and outdoor cooling systems.

3.1 Indoor Damping System

The indoor damping system was designed to reduce construction costs. The materials used for each component of the damper are inexpensive, and mostly re-usable for other purposes. Figure 4 in Appendix B1 shows the final design of the damping system.

The shell of the damper is constructed with cork material which has a high specific heat capacity that helps retain the "coldness" excess cool air (air that is blocked by the diffuser to be circulated to other areas). The high impermeability of cork to gases is excellent in retaining the excess cool air.

Furthermore, cork is also acoustically insulating and provides a more conducive environment for the user as mechanical noise from the cooling systems are attenuated.

The acrylic damper provides air resistance for air volume control and is sufficiently sturdy to withstand the stepper motor forces and forces from the cool air pushing out from the diffuser.

When the sensors detect a change in temperature of the environment, the stepper motor turn the damper to adjust the amount of cool air supplied. The higher the temperature, the larger the degree of the damper's opening.

3.2 Outdoor VSF System

The 12 V DC fans are powered by a voltage supply and connected to the sensors. When the surrounding temperature increases, the fans will draw more power and increases their RPM (Revolutions Per Minute), thereby increasing the cooling effect. The 12V DC fans have a minimum of 1000 RPM and a maximum of 2000 RPM.

3. System's Server

The server's HMI (Human Machine Interface) monitors all sensors (indoors and outdoors) and cooling systems, and allows real time calibration of each sensors individually. The HMI is intuitive and user friendly and the user only needs to input the sensor's readings for 2 temperature points.

As such, should any sensor become inaccurate over time due to wear and tear, the user can easily recalibrate the sensor without affecting the operations of other sensors.

5. Energy Savings

The smart ACMV system developed optimises cooling in both indoor and outdoor systems through thermal sensors operating at around $23 - 32$°C. As temperature changes, the dampers and VSFs optimises airflow discharge.

At 25℃, the power drawn by VSF without varying speed is:

$$P_{constant} = 12V \times 0.2A = 2.4W \qquad (1)$$

When the fan speed is allowed to vary at the same temperature, it consumes approximately 1.70 W:

$$P_{vary} = 12V \times 0.144A = 1.728W \qquad (2)$$

This suggests that the outdoor system conserves about 29% of energy at 25℃ per fan, as calculated by:

$$P_{savings} = \frac{(2.4 - 1.7)W}{2.4W} \times 100\% = 29.2\% \qquad (3)$$

Examining the indoor damping system, the amount of energy supplied to the room through each damper can be derived by:

$$Q = \dot{V}\rho C_p \Delta T \qquad (4)$$

Where $\dot{V}$ is volume flow rate of air through the damper. Given that the density and specific heat capacity of air is remains relatively constant, the only variable affecting the change in energy supplied is $\dot{V}$. Hence, it can be denoted as:

$$Q = Q(\dot{V}) \qquad (5)$$

This means that when the temperature of the room is lower and less chilled air is required in the room, the damper will close such that less chilled air from the AHU will flow into the room. Hence, if the damper is 50% closed, only 50% of the chilled air will flow through the damper, into the room.

6. Conclusion and Discussions

The indoor and outdoor ACMV control system is a unique and promising concept that is yet acknowledged by the market. This system is promising in dining areas with both indoor and Al Fresco elements where customers' comfort is of paramount importance. Other than saving energy costs, commercial industries with both outdoor and indoor elements can save on installation and maintenance cost by using a single control system with identical sensor systems. The sensors work independently of one another and thus, the system can be easily scaled up to cover a larger area or to better control the temperature distribution.

Calibration of the sensor might not be accurate due to the thermocouple available. However, each sensor can be conveniently recalibrated using the software designed.

6.1 Limitations and Future Improvements

The sensors require frequent recalibration due to the prototyping materials used. This system should be deployed with more permanent solutions by soldering all the components together or by using PCBs.

Furthermore, the VSF deployed in this system has a minimum operating RPM and cannot be turned off completely. This can be resolved by integrating a transistor to cut off the power supply below the minimum RPM.

7. References

[1] D. Zhai and Y. C. Soh, “Balancing indoor thermal comfort and energy consumption of air-conditioning and mechanical ventilation systems via sparse Firefly algorithm optimization,” 3 July 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7966028.

[2] Price Industries, “Engineering Guide VAV Diffusers,” 2011. [Online]. Available: https://www.priceindustries.com/content/uploads/assets/literature/engineering-guides/vav-diffusers-engineering-guide.pdf.

[3] MacroAir, “AirEffect,” MacroAir, [Online]. Available: https://macroairfans.com/controls-automation/aireffect/.

[4] B. Schweber, “The Wheatstone Bridge: Still the preferred sensor-interface topology after 180 years,” 7 April 2016. [Online]. Available: https://electronics360.globalspec.com/article/6522/the-wheatstone-bridge-still-the-preferred-sensor-interface-topology-after-180-years#:~:text=The%20Wheatstone%20bridge%20allows%20the,the%20voltmeter%2C%20ammeter%20or%20ohmmeter..

[5] AWV, “6 Types of Industrial Dampers and When to Use Them,” AWV, [Online]. Available: https://awv.com/blog/types-of-dampers/.

[6] M. Long, “18 - Sound Reinforcement Systems,” 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123982582000180.

[7] Cadence PCB Solutions, “What’s a 4-Pin PWM Header and How Does It Work?,” [Online]. Available: https://resources.pcb.cadence.com/blog/2021-what-s-a-4-pin-pwm-header-and-how-does-it-work.

Appendix A: Sensor Design

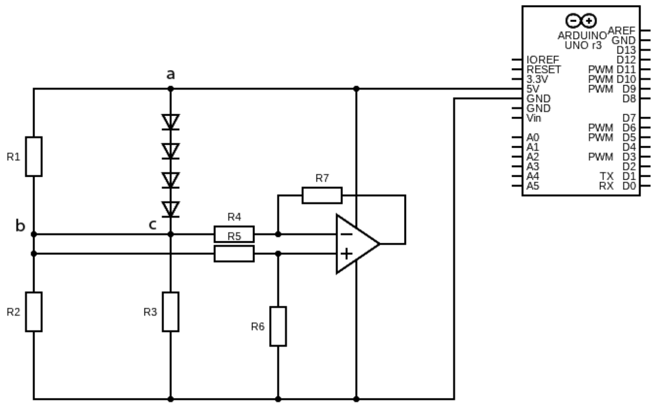


Figure 1. Sensor circuit design.

For the sensor (Fig. 1), a series of diodes is connected in a Wheatstone bridge configuration, in conjunction with a differential amplifier. Four of these sensor systems are built identically and distributed around the site, where two of which are responsible for the indoor area, and the other two are responsible for the outdoor area.

A1. Diode

Diodes are inexpensive and their resistance respond to changes in temperature, making them the perfect sensor for the ACMV system. As temperature increases, resistance of diode decreases, thus, changing the voltage across it. Four diodes are connected in series to increase the diodes’ response to temperature. Furthermore, they are operated at lower current values as the lower the current, the larger the temperature response.

A2. Wheatstone Bridge

Wheatstone bridges allow for more precise and accurate voltage readings [4]. Noises in supply voltage and time-related resistance drifts are effectively cancelled out by the bridge.

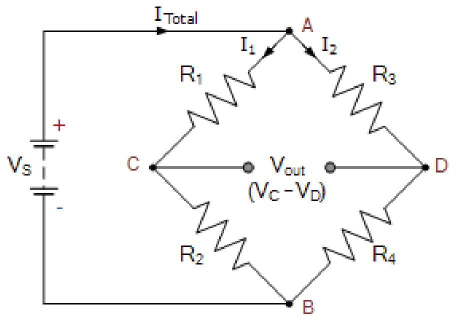


Figure 2. Wheatstone bridge topology.

$$V_c = V_a + \left(\frac{V_s R_1}{R_1 + R_3}\right) \tag{6}$$

$$V_d = V_a + \left(\frac{V_s R_4}{R_4 + R_2}\right) \tag{7}$$

Figure 2 shows the configuration of the Wheatstone bridge for the designed sensor where R_4 represents the series of resistance-varying diodes. As temperature increases, R_4 decreases, causing more imbalance in the Wheatstone bridge and thus, voltage difference across c and d increases according to Equations 6 and 7. The exact resistance values for R_1, R_2 and R_3 are dependent on each individual sensor system as they have different internal resistance. The resistance values are calibrated by balancing the Wheatstone bridge (output voltage of 0V).

A3. Differential Amplifier

Output signal from the Wheatstone bridge is processed through a differential amplifier (LM358) to suppress noise and amplify the voltage output for better temperature resolution. LM358 has a Common-Mode Rejection Ratio (CMRR) of 80dB at the sensor's operating voltage (between 0 to 5V). CMMR is a measure of the amplifier's ability to filter noise while amplifying desired signals and is calculated as follows:

$$CMRR = 20 \log \left(\frac{A_{v(d)}}{A_{cm}}\right) \tag{8}$$

Where $A_{v(d)}$ represents the differential voltage gain (desired signal) and A_{cm} represents the common-mode gain (desired signal). Thus, the higher the CMMR, the better the amplifier is at rejecting unwanted common-mode noise. The differential amplifier is configured with negative feedback to achieve a gain of about 10.

A4. Sensor Algorithm

As the voltage readings fluctuates, the system uses an averaged value to attenuate abnormal readings and smoothen the signal. As such, the average value of 5000 samples is being transmitted each cycle where each sample are taken approximately 4 ms apart.

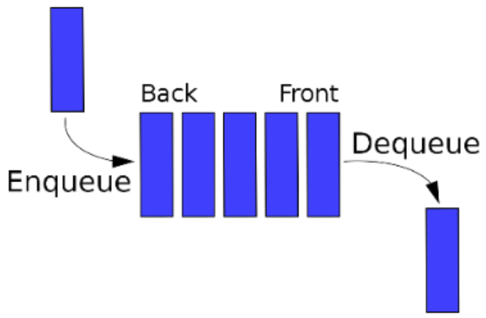


Figure 3. First-In-First-Out (FIFO) operation of a queue data structure.

However, this causes a sluggish update where the user can only see the updated temperature once every 20 seconds. Hence, the queue data structure is implemented as shown in Figure 3 where the samples are stored in an array and in each cycle, the front sample of the array (earliest sample in the array) will be popped off (dequeued) while a new sample reading will be appended (enqueued) to the back of the data structure. The average of this queue is then calculated and transmitted, allowing an update once every 4 ms approximately.

However, to avoid stack overflow error when calculating the average of the 5000 samples, the size of the queue is limited to 200 and each data pushed into the queue is an averaged reading of 25 samples by itself. The sensor thus updates once every 100ms, which is still very responsive for an ACMV system.

Appendix B: Mechanical Design

B1. Fabrication of Dampers

The indoor damping system was designed to replicate industrial butterfly dampers used in round ducts. This is because butterfly dampers with only a single blade require less materials to construct, making it less costly [5]. Also, since the fan outlet is circular, a round duct can ensure more efficient air volume control.

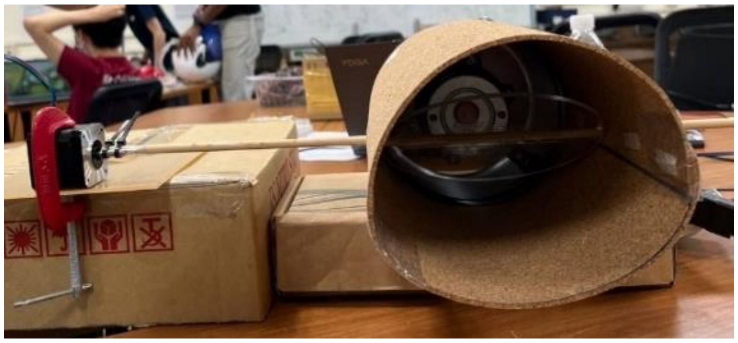


Figure 4. Final design of damper.

To fabricate the circular dampers, cork sheets were used as the shell of the damper, while 3 mm-thick acrylic sheets were laser-cut into circular "blades" that could fit into each cork shell. Wooden rods were inserted through two ends of each shell and secured to the acrylic blade inside using hot glue and tape. Lastly, stepper motors were attached to one end of the wooden rod with tape and cable ties to control the damper. Cardboard boxes were also reused to function as a platform for the fan and a support structure to clamp the motor down. Figure 4 shows the final design of the fabricated damper.

B2. Electronics of Indoor Damping System

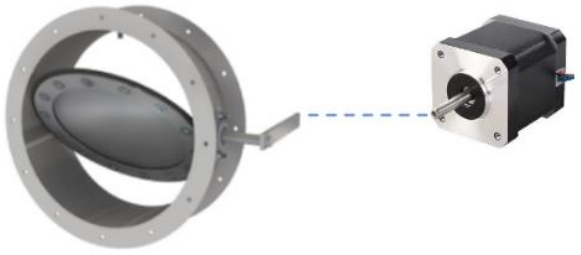


Figure 5. Damper connection to stepper motor.

The round Volume Control Damper (VCD) is connected to the stepper motor as shown in Figure 5. The damper is placed in front of a fan, simulating the airflow from the AHU. The stepper motor is controlled by an Arduino Uno to turn the damper, thus adjusting the airflow. Since the Arduino Uno's operating voltage (5V) is below the stepper motor's operating voltage (12V), a driver is needed to control the switching of stepper motor voltages.

The A4988 is a micro stepping driver for controlling bipolar stepper motors which has a built-in translator. The stepper motor can be controlled with just 2 pins from the controller, one for controlling the rotation direction and the other for controlling the steps. The micro steps of the motor can be controlled by sending pulses to the step pin from the Arduino Uno. With each pulse sent, the motor moves one step. Operating the driver in full step mode by leaving the 3 MS pins disconnected, the motor turns one full revolution with 200 steps.

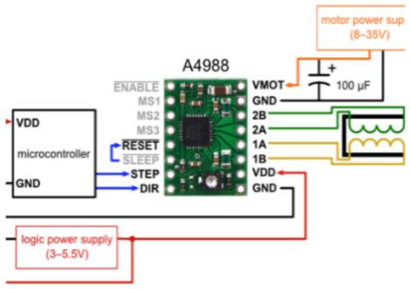


Figure 6. Wire connections of the stepper motor to Arduino and A4988 driver.

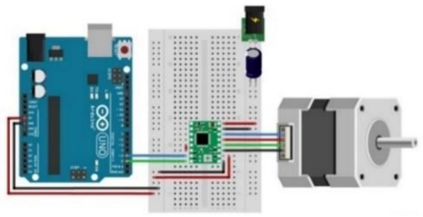


Figure 7. Final overall stepper motor connection.

Following Figure 6, the stepper motor is powered by connecting the Ground and VMOT pins to a 12 V power supply with a 47 µF decoupling capacitor to protect the driver board from any voltage spikes. 1A,1B pins are connected to one coil of the motor and 2A,2B pins to the other coil of the motor. The VDD and Ground pins are connected to the Arduino Uno to power the driver board. Figure 7 displays the final stepper motor connections that will be controlling the damper system.

B3. Electronics of Outdoor VSF System

Pulse-Width Modulation (PWM) fans were employed in the outdoor VSF system to regulate the fan speed according to the temperature sensor readings.

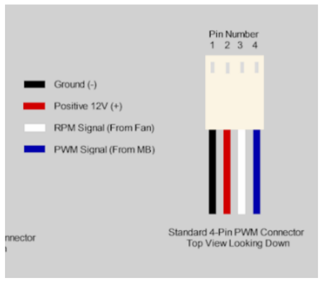


Figure 8. Wiring of PWM fan.

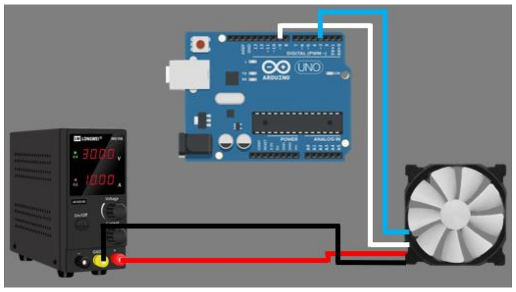


Figure 9. Wire connections of the PWM fan to Arduino and +12V DC supply.

Each PWM fan has four connections for: ground, +12 V power supply, Revolutions-Per-Minute (RPM) signal from the fan, and PWM signal from the Arduino. Figure 8 depicts the colour configuration for each connection, while Figure 9 displays how they were connected to the DC power supply and Arduino.

PWM turns digital value into analog voltage by sending pulses with a duty cycle – which is a measure of the input signal's peak-to-average ratio over time – that results in the desired analog voltage for the fan . The speed of the VSF is proportional the applied PWM duty cycle. When the surrounding temperature increases, the temperature sensor picks up this rise in temperature, causing the control algorithm to increase the fan's duty cycle. This then increases the speed of the PWM fan, increasing the outdoor cooling effect.

Appendix C: System's Server Design

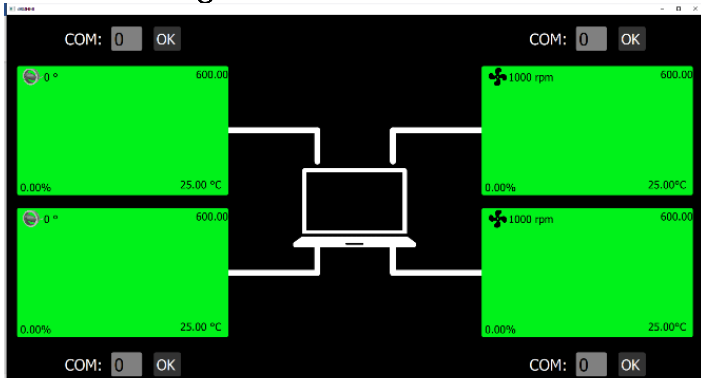


Figure 10: Server HMI (Human Machine Interface).

The sensors and actuators are controlled by a single centralised server (Fig 10). All readings from the sensor are sent from the Arduino to a customed-built software which will then send control signals to the respective actuators.

C1. Sensor Reading

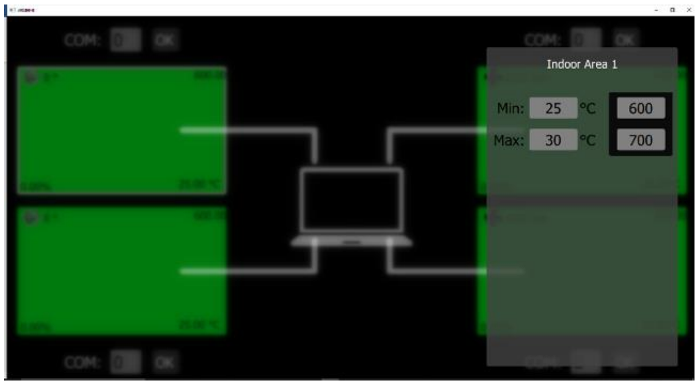


Figure 11. Server HMI controls.

The software allows for individual in-place calibration of the sensor by setting the values received at the minimum and maximum temperature points (Fig 11). The temperature value for each area is displayed according to the settings configured. As the temperature increases, the color of the area will become redder until it reaches the maximum temperature, beyond which, the color stops changing (but the temperature displayed still increases – i.e., there are no minimum and maximum values for temperature sensed).

C2. Actuator Control Signals

Every 10 seconds, the system will send out a control signal in the form of a percentage to each of the 4 actuators. This percentage decides the angle of the damper for the indoor system and the revolutions-per-minute (RPM) of the fan for the outdoor system. The percentage is calculated using Equation 9:

$$Percentage = \frac{T_{current} - T_{min}}{T_{max} - T_{min}} \times 100\% \qquad (9)$$

The angle of the damper and RPM of the fan can be calculated using Equations 10 and 11, respectively:

$$Angle = percentage \times 90° \tag{10}$$

$$RPM = (percentage \times 1000) + 1000 \tag{11}$$

Appendix D: Experimental Results

The sensors are calibrated by measuring the voltage output at different temperature points. Although the temperature-resistance responses of the sensors are non-linear, the linear results are expected due to the small temperature range.

D1. Calibration of Temperature Sensor

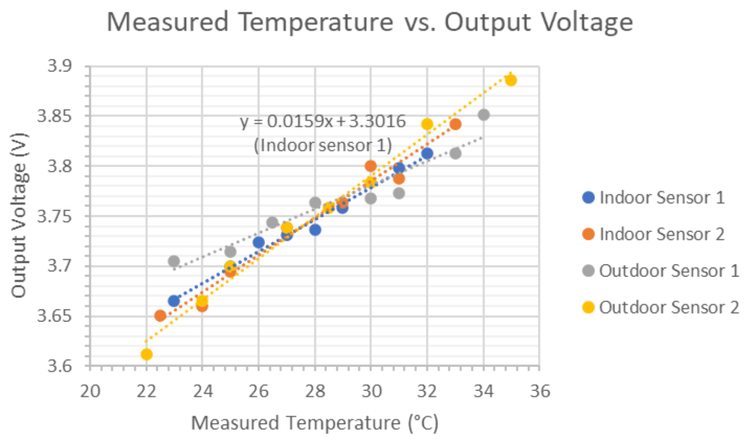


Figure 12. Calibration graph of output voltage (V) against measured temperature (℃).

A thermocouple was used to measure the true temperature and heat was directed at the sensors using a hair dryer. Control of heat applied onto the sensors to achieve different temperature points is performed by varying the distance of the hair dryer away from the sensor. The calibration graph is seen in Figure 12.

D2. Final Testing

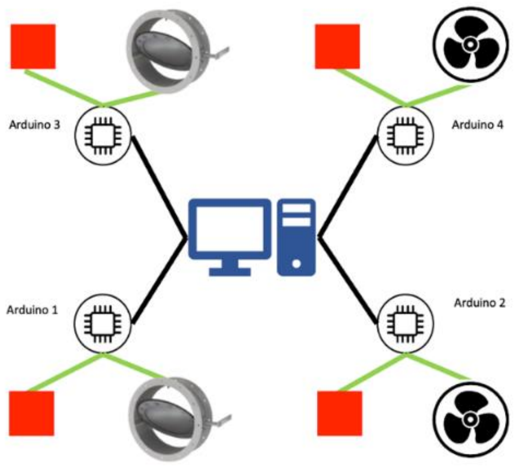


Figure 13. Connections of the sensors, Arduinos, fans, and dampers.

The final testing is performed on the fully assembled system as shown in Figure 13. The four temperature sensors are distributed around the area at different locations to better represent the temperature distribution in the room.

For the indoor cooling system, each Arduino is connected to a stepper motor and a thermal sensor. For the outdoor cooling system, each Arduino is connected to a DC fan and a thermal sensor.

Two types of testing – high temperature range and small temperature range tests are performed. In the high range temperature test, a hair dryer is used as heat source to apply a wide range of temperature onto the sensor from 22°C to 41°C by varying its distance away from the sensors. As the temperature increases (measured by thermocouple) and maintained over a period of approximately 10s, the actuators increased their cooling effect. For the indoor damper system, the increased cooling effect can be felt physically due to the strong AC fan used to simulate the cool air from the AHU. For the outdoor VSF, the increased cooling effect can be observed through the increased current drawn displayed on the VSF power supply.

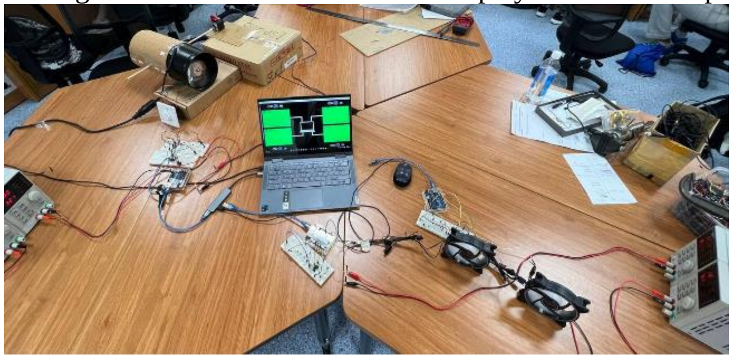


Figure 14. Final setup.

In the small temperature range test, an infrared lamp is used as a heat source to apply a small change in temperature. The infrared lamp is brought close to the sensor and the position is maintained. The change in temperature recorded by the thermocouple is less than $0.5°C$. For the indoor damper system, the motor can be audibly heard adjusting for the change in the small temperature. However, the visual change in angle and thus airflow, is too subtle to be observed. Changes in anemometer readings can be observed but is too small and unstable to be confidently attributed to the change in airflow due to the damper's angle. For the outdoor VSF, the increased cooling effect can be observed through the increased current drawn displayed on the VSF power supply. The final set up is shown in Figure 14.

Appendix E: Arduino Code for Motor

```
#include <cQueue.h>


/////////////////////// LAYOUT ///////////////////////

//  1. DESCRIPTION          #T_DESCRIPTION          //

//                                                  //

//  2. GLOBAL VARIABLES                             //

//                                                  //

//  3. FUNCTIONS                                    //

//                                                  //

//  4. MAIN                                         //

//                                                  //

//////////////////////////////////////////////////////

//
==============================================================================
==                              GLOBAL                                VARIABLES
==============================================================================
======== //

/**/  typedef union {                                                          //
/**/    float floatingPoint;                                                  //
/**/    byte binary[4];                                                       //
/**/  } binaryFloat;                                                          //
/**/                                                                          //
/**/                    const        int        BAUD_RATE        =        9600;
//
/**/  const float PIN_ONE = A0;                                               //
/**/          const     int     SAMPLES     =     200,     SUB_SAMPLES     =     25;
//
/**/  const byte numChars = 32;                                               //
/**/                                                                          //
/**/      //   ------------------------------------   SERIAL   RECIEVE   ----------------------------------   //
//
/**/   /**/       char  receivedChars[numChars];                              //
//
/**/   /**/       char  tempChars[numChars];        //  temporary  array  for  use  when  parsing       //
//
/**/  /**/                                          //                        //
/**/   /**/       char  messageFromPC[numChars]  =  {0};                      //
//
/**/  /**/                                          //                        //
/**/  /**/    boolean newData = false;                    //                  //
/**/      //   ------------------------------------   SERIAL   RECIEVE   ----------------------------------   //
//
/**/                                                                          //
/**/ // ---------- MOTOR -------------- //
/**/    /**/const int stepPinOne = 4;  /**/
/**/    /**/const int dirPinOne = 3;   /**/
/**/                 /**/     float     motor_one_angle     =     0;                  /**/
//
/**/    /**/ float new_motor_angle_one = 0.0;
/**/    /**/ int angle_to_move_motorOne;
/**/  // ----------------------------- //                                       //
/**/                                                                          //
/**/                  //---------------           SERIAL           SEND           -------------------//
//
/**/  /**/    binaryFloat sensor_1;       //                                  //
```

```
/**/ /**/   float sum_1, throw_away;        //                                                              //
/**/                    //--------------          SERIAL          SEND          ------------------//
//
/**/                                                                                                   //
/**/  Queue_t voltage_readings_0; // size = SAMPLES //Each element is an average value with number of
samples denoted SUB_SAMPLES                        //
/**/                                                                                                   //
//    ===========================================================================
GLOBAL                                      VARIABLES                                      END
===========================================================================
========= //


//===========================================================================
=====                                                                              FUNCTIONS
===========================================================================
===============//
                ////////////////////////////////////////////////////////////////////////////////////////////////
                //                    COMMUNICATIONS                    //
                ////////////////////////////////////////////////////////////////////////////////////////////////
/**/void                              recvWithStartEndMarkers()                              {
//
/**/                   static       boolean       recvInProgress       =       false;
//
/**/   static byte ndx = 0;                                                                      //
/**/   char startMarker = '<';                                                                   //
/**/   char endMarker = '>';                                                                     //
/**/   char rc;                                                                                  //
/**/                                                                                             //
/**/          while    (Serial.available()    >    0    &&    newData    ==    false)    {
//
/**/     rc = Serial.read();                                                                     //
/**/                                                                                             //
/**/     if (recvInProgress == true) {                                                           //
/**/       if (rc != endMarker) {                                                                //
/**/         receivedChars[ndx] = rc;                                                            //
/**/           ndx++;                                                                            //
/**/       if (ndx >= numChars) {                                                                //
/**/           ndx = numChars - 1;                                                               //
/**/          }                                                                    //
/**/       }else {                                                                 //
/**/                        receivedChars[ndx] = '\0'; // terminate the string
//
/**/        recvInProgress = false;                                                //
/**/         ndx = 0;                                                              //
/**/         newData = true;                                                       //
/**/       }                                                                       //
/**/     } else if (rc == startMarker) {                                           //
/**/       recvInProgress = true;                                                  //
/**/     }                                                                         //
/**/   }                                                                           //
/**/  }                                                                            //
/**/                                                                               //
/**/          void parseData() {              // split the data into its parts
//
/**/                                                                               //
/**/                  char * strtokIndx; // this is used by strtok() as an index
//
/**/                                                                               //
/**/            strtokIndx = strtok(tempChars,",");        // get the first part - the string
//
/**/           new_motor_angle_one = atof(strtokIndx);      // convert this part to an integer
//
/**/                                                                               //
/**/                                                                               //
//
/**/                                                                               //
/**/                                                                               //
/**/  }                                                                            //
/**/                                                                               //
//==========================================================================
=========                          FUNCTIONS                              END
==========================================================================
=======//
void setup() {
  //================== INITIALISE SENSOR ====================//
```

```
    Serial.begin(BAUD_RATE);
    q_init(&voltage_readings_0, 4, SAMPLES, FIFO, false);
    float init_reading_0 = 0;
    for (int i = 0; i < SUB_SAMPLES; ++i){
      init_reading_0 += analogRead(PIN_ONE);
    }
    init_reading_0 /= SUB_SAMPLES;
    for (int i = 0; i < SUB_SAMPLES; i++){
      q_push(&voltage_readings_0, &init_reading_0);
    }
  // ================ INITIALISE SENSOR END =================//
  // =================== INITIALISE MOTOR ===================//
    pinMode(stepPinOne,OUTPUT);
    pinMode(dirPinOne,OUTPUT);
  // ================ INITIALISE MOTOR END ==================== //
}

void loop() {
  ////////////////////////////////////////////////////////////////////////////////////////
  //                     1. RECIEVEING DATA FROM SENSOR                    //
  ////////////////////////////////////////////////////////////////////////////////////////
        //==================================                SENSOR                 1
===============================//
        /**/  float voltage_reading_0 = 0;
        /**/  //---------------- SUB SAMPLING --------------------//
        /**/  /**/  for (int i = 0; i < SUB_SAMPLES; ++i){        //
        /**/  /**/    analogRead(PIN_ONE);                       //
        /**/  /**/    delay(2);                                  //
        /**/  /**/    voltage_reading_0 += analogRead(PIN_ONE);    //
        /**/  /**/    delay(2);                                  //
        /**/  /**/  }                                            //
        /**/  /**/  voltage_reading_0 /= SUB_SAMPLES;              //
        /**/  //------------------------------------------------//
        /**/  q_pop(&voltage_readings_0,&throw_away);
        /**/  q_push(&voltage_readings_0, &voltage_reading_0);
        /**/
        /**/  sum_1 = 0;
        /**/  for (int i = 0 ; i < SAMPLES ; i++){
        /**/    float data;
        /**/    q_peekIdx(&voltage_readings_0, &data, i);
        /**/    sum_1 += data;
        /**/  }
        /**/  sum_1 /= SAMPLES;
        /**/
        //==================================                SENSOR                1
==============================//
  ///////////////////////////////////////////////////////////////////////////
  //                        2. SENDING DATA TO SERVER                        //
  ///////////////////////////////////////////////////////////////////////////
      sensor_1.floatingPoint = sum_1;
      delay(2);
      Serial.write(sensor_1.binary,4);
      delay(2);
  ////////////////////////////////////////////////////////////////////////////////////
  //                        3. RECIEVING DATA FROM SERVER                           //
  ////////////////////////////////////////////////////////////////////////////////////
      recvWithStartEndMarkers();
      if (newData == true) {
                  strcpy(tempChars, receivedChars);
                    // this temporary copy is necessary to protect the original data
                    //   because strtok() used in parseData() replaces the commas with \0
                  parseData();
                  newData = false;
                  ////////////////////////////////////////////////////////////////////////////////////
                  //                        4. UPDATING FAN                           //
                  ////////////////////////////////////////////////////////////////////////////////////
                    //          ========================          MOTOR          ONE
================================ //
                   new_motor_angle_one *= 50;
                   angle_to_move_motorOne = new_motor_angle_one - motor_one_angle;
```

```
                        if (angle_to_move_motorOne >0 ){
                          digitalWrite(dirPinOne,HIGH);
                        }
                        if (angle_to_move_motorOne <0 ){
                          digitalWrite(dirPinOne,LOW);
                        }

                        for(int x = 0; x < abs(angle_to_move_motorOne); x++) {
                          digitalWrite(stepPinOne,HIGH);
                          delayMicroseconds(500);
                          digitalWrite(stepPinOne,LOW);
                          delayMicroseconds(500);
                        }
                        motor_one_angle += angle_to_move_motorOne;
        }

}
```

Appendix F: Arduino Code for VSF

```
#include <cQueue.h>

/////////////////////// LAYOUT ///////////////////////
//  1. DESCRIPTION          #T_DESCRIPTION          //
//                                    //
//  2. GLOBAL VARIABLES                          //
//                                    //
//  3. FUNCTIONS                          //
//                                    //
//  4. MAIN                                //
//                                    //
//////////////////////////////////////////////////////
//
================================================================
======== GLOBAL VARIABLES
================================================================
============  //
/**/  typedef union {
//
/**/    float floatingPoint;
//
/**/    byte binary[4];
//
/**/  } binaryFloat;
//
/**/
//
/**/  const int BAUD_RATE = 9600;
//
/**/  const float PIN_ONE = A0;
//
/**/  const int SAMPLES = 200, SUB_SAMPLES = 25;
//
/**/  const byte numChars = 32;
//
/**/
//
/**/  // ------------------------------------ SERIAL RECIEVE ------------------------------------//
//
/**/  /**/      char receivedChars[numChars];                                        //
//
/**/  /**/      char tempChars[numChars];        // temporary array for use when parsing          //
//
/**/  /**/                                                                 //
//
/**/  /**/      char messageFromPC[numChars] = {0};                                    //
//
/**/  /**/                                                                 //
//
/**/  /**/      boolean newData = false;                                         //
//
/**/  // ------------------------------------ SERIAL RECIEVE ---------------------------------- //
//
/**/
//
/**/ // --------- FAN --------------//
//
/**/  /**/  float pwm_ratio_1 = 0.0; //
//                                                                                       //
/**/
//
/**/  //--------------- SERIAL SEND -------------------//
//
/**/  /**/    binaryFloat sensor_1;
//
```

```
/**/  /**/    float sum_1, throw_away;
//
/**/  //-------------- SERIAL SEND -------------------//
//
/**/
//
/**/  Queue_t voltage_readings_0; // size = SAMPLES //Each element is an average value with
number of samples denoted SUB_SAMPLES                                //
/**/
//
//
===========================================================================
=== GLOBAL VARIABLES END
===========================================================================
=============== //

//=========================================================================
=========== FUNCTIONS
===========================================================================
=====================//
                    ////////////////////////////////////////////////////////////////////////////////////////////
                    //                        COMMUNICATIONS                         //
                    ////////////////////////////////////////////////////////////////////////////////////////////
/**/void recvWithStartEndMarkers() {
//
/**/    static boolean recvInProgress = false;
//
/**/    static byte ndx = 0;
//
/**/    char startMarker = '<';
//
/**/    char endMarker = '>';
//
/**/    char rc;
//
/**/
//
/**/    while (Serial.available() > 0 && newData == false) {
//
/**/      rc = Serial.read();
//
/**/
//
/**/      if (recvInProgress == true) {
//
/**/        if (rc != endMarker) {
//
/**/          receivedChars[ndx] = rc;
//
/**/            ndx++;
//
/**/        if (ndx >= numChars) {
//
/**/            ndx = numChars - 1;
//
/**/          }
//
/**/        }else {
//
/**/          receivedChars[ndx] = '\0'; // terminate the string
//
/**/          recvInProgress = false;
//
/**/          ndx = 0;
//
/**/          newData = true;
//
/**/        }
//
/**/      } else if (rc == startMarker) {
//
/**/        recvInProgress = true;
//
/**/      }
//
/**/    }
//
/**/  }
//
/**/
//
/**/  void parseData() {      // split the data into its parts
//
/**/
//
/**/    char * strtokIndx; // this is used by strtok() as an index
//
/**/
//
/**/    strtokIndx = strtok(tempChars,",");      // get the first part - the string
//
```

```
/**/    pwm_ratio_1 = atof(strtokIndx);     // convert this part to an integer
//
/**/
//
/**/
//                                                                                //
/**/
//
/**/
//
/**/  }
//
/**/
//
                          ////////////////////////////////////////////////////////////////////////////////////////////////
                          //                              FAN                                   //
                          ////////////////////////////////////////////////////////////////////////////////////////////////
void setupTimer1(){
   //Set PWM frequency to about 25khz on pins 9,10 (timer 1 mode 10, no prescale, count to
320)
   TCCR1A = (1 << COM1A1) | (1 << COM1B1) | (1 << WGM11);
   TCCR1B = (1 << CS10) | (1 << WGM13);
   ICR1 = 320;
   OCR1A = 0;
   OCR1B = 0;
}
//configure Timer 2 (pin 3) to output 25kHz PWM. Pin 11 will be unavailable for output in this
mode
void setupTimer2(){
   //Set PWM frequency to about 25khz on pin 3 (timer 2 mode 5, prescale 8, count to 79)
   TIMSK2 = 0;
   TIFR2 = 0;
   TCCR2A = (1 << COM2B1) | (1 << WGM21) | (1 << WGM20);
   TCCR2B = (1 << WGM22) | (1 << CS21);
   OCR2A = 79;
   OCR2B = 0;
}
//equivalent of analogWrite on pin 9
void setPWM1A(float f){
   f=f<0?0:f>1?1:f;
   OCR1A = (uint16_t)(320*f);
}
//equivalent of analogWrite on pin 10
void setPWM1B(float f){
   f=f<0?0:f>1?1:f;
   OCR1B = (uint16_t)(320*f);
```

```
}
//equivalent of analogWrite on pin 3
void setPWM2(float f){
   f=f<0?0:f>1?1:f;
   OCR2B = (uint8_t)(79*f);
}
//================================================================
=============== FUNCTIONS END
================================================================
=============//
void setup() {
  //================== INITIALISE SENSOR ====================//
      Serial.begin(BAUD_RATE);
      q_init(&voltage_readings_0, 4, SAMPLES, FIFO, false);
      float init_reading_0 = 0;
      for (int i = 0; i < SUB_SAMPLES; ++i){
        init_reading_0 += analogRead(PIN_ONE);
      }
      init_reading_0 /= SUB_SAMPLES;
      for (int i = 0; i < SUB_SAMPLES; i++){
        q_push(&voltage_readings_0, &init_reading_0);
      }
  // ================ INITIALISE SENSOR END =================//
  // =================== INITIALISE FAN ===================//
      pinMode(9,OUTPUT); //1A
//      pinMode(10,OUTPUT); //1B
      setupTimer1();
      //enable outputs for Timer 2
//      pinMode(3,OUTPUT); //2
      setupTimer2();
      //note that pin 11 will be unavailable for output in this mode!
      //example...
      setPWM1A(1.0f); //set duty to 50% on pin 9
  // ================ INITIALISE FAN END ==================== //
}

void loop() {
  /////////////////////////////////////////////////////////////////////////////////////////
  //                          1. RECIEVEING DATA FROM SENSOR                                //
  /////////////////////////////////////////////////////////////////////////////////////////
          //================================= SENSOR 1
==============================//
          /**/  float voltage_reading_0 = 0;
          /**/  //----------------- SUB SAMPLING ---------------------//
          /**/  /**/  for (int i = 0; i < SUB_SAMPLES; ++i){          //
          /**/  /**/    analogRead(PIN_ONE);                          //
```

```
    /**/  /**/    delay(2);                              //
    /**/  /**/    voltage_reading_0 += analogRead(PIN_ONE);     //
    /**/  /**/    delay(2);                              //
    /**/  /**/  }                                      //
    /**/  /**/  voltage_reading_0 /= SUB_SAMPLES;              //
    /**/  //--------------------------------------------------//
    /**/  q_pop(&voltage_readings_0,&throw_away);
    /**/  q_push(&voltage_readings_0, &voltage_reading_0);
    /**/
    /**/  sum_1 = 0;
    /**/  for (int i = 0 ; i < SAMPLES ; i++){
    /**/    float data;
    /**/    q_peekIdx(&voltage_readings_0, &data, i);
    /**/    sum_1 += data;
    /**/  }
    /**/  sum_1 /= SAMPLES;
    /**/
    //================================== SENSOR 1
==============================//

  //////////////////////////////////////////////////////////////////////////////
  //                      2. SENDING DATA TO SERVER                       //
  //////////////////////////////////////////////////////////////////////////////
      sensor_1.floatingPoint = sum_1;
//      delay(2);
//      Serial.write('a');
      delay(2);
      Serial.write(sensor_1.binary,4);
      delay(2);
//    Serial.println(sum_1);
  //////////////////////////////////////////////////////////////////////////////////
  //                      3. RECIEVING DATA FROM SERVER                          //
  //////////////////////////////////////////////////////////////////////////////////
      recvWithStartEndMarkers();
      if (newData == true) {
                  strcpy(tempChars, receivedChars);
                     // this temporary copy is necessary to protect the original data
                     //   because strtok() used in parseData() replaces the commas with \0
                  parseData();
                  newData = false;
                  //////////////////////////////////////////////////////////////////////////////////
                  //                        4. UPDATING FAN                            //
                  //////////////////////////////////////////////////////////////////////////////////
                  setPWM1A(pwm_ratio_1); //set duty to pwm_ratio_1 on pin 9

      }
```

```
}
```

Appendix G: Server Sided Code

Server is coded using Qt framework. The project folder is too large to include in the annex.
Serial is transmitted in Little Endian encoding, with float values transmitted from Arduino to server in a byte array of size 4. Control signals from server to the Arduino is transmitted in string and parsed in the Arduino every 10s.