Blake Bourque
ECE577 - Final Project
Spring 2015

F.L.C. Vent Fan

High humidity levels are often considered bad for homes. Many companies sell in-ceiling vent-fans. These vent fans are efficient at removing hot moist air from bathrooms and offensive odors from restrooms. The trouble with these vent-fans is that we often leave them on for too long, or just forget to turn them off. However there is also the problem of not leaving the vent fans on long enough which over the course of years can cause damage to the home.

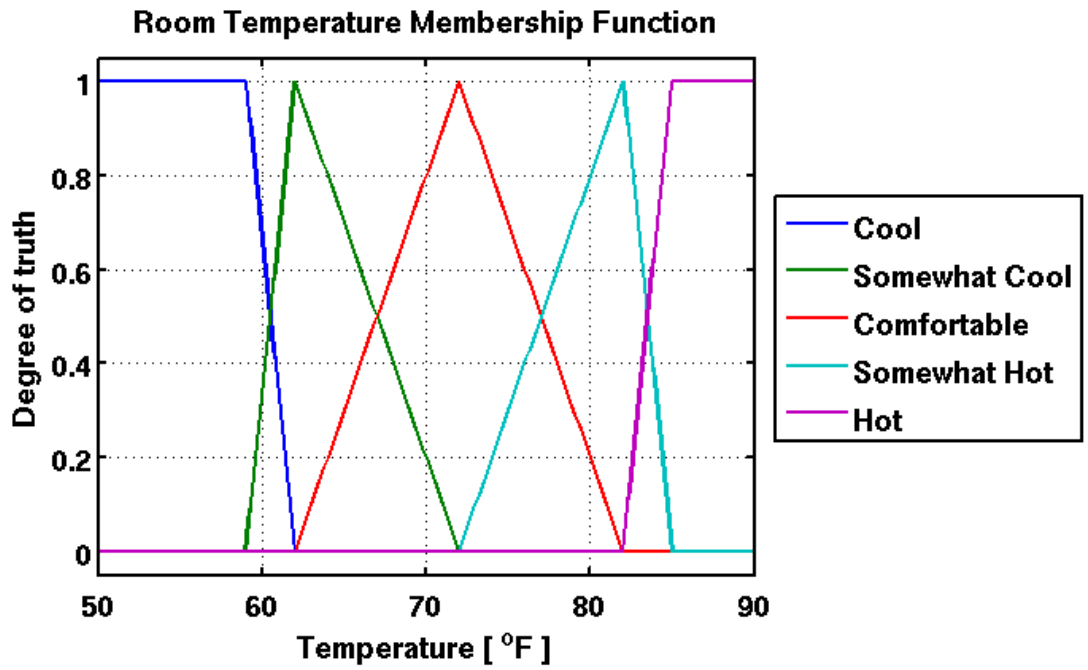
The proposed project remedies both of these cases by using a Fuzzy Logic Controller and a humidity/temperature sensor to control the vent fan. The system will have these inputs and outputs:

- Inputs:
  - Relative Humidity
  - Temperature
- Outputs:
  - Vent fan speed. 0 is off 100 is full speed

We can describe the rules for which to run the fan summarized in tabular form:

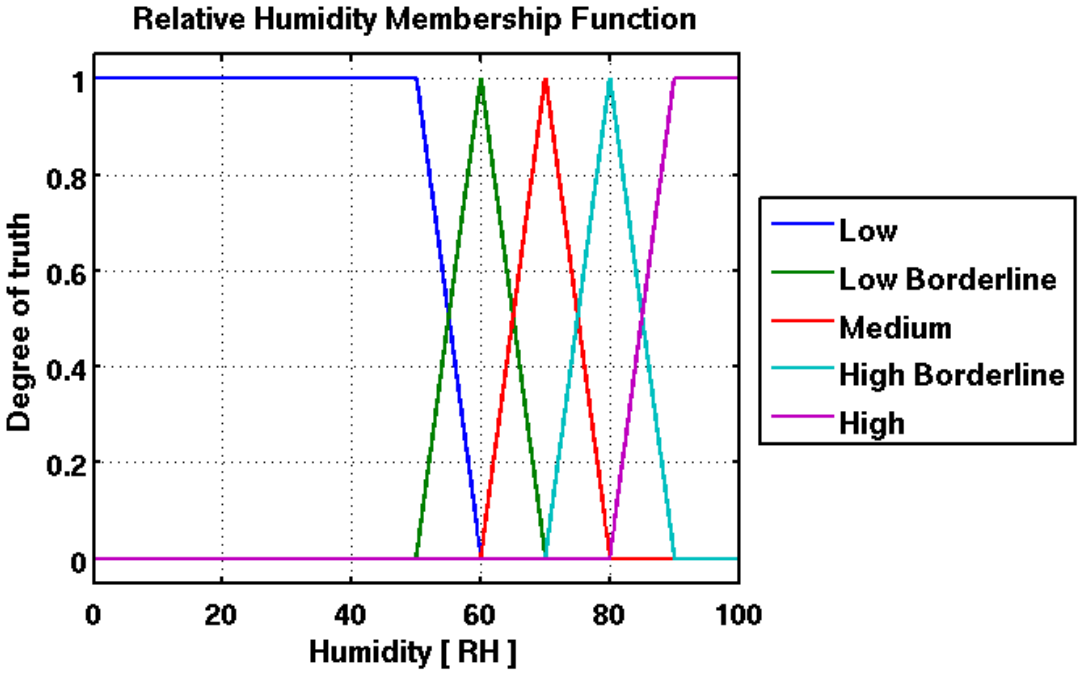
Temperature ▸ Humidity ▼	Cool	Somewhat Cool	Comfortable	Somewhat Hot	Hot
Low	Off	Off	Off	Off	Off
Low Borderline	Off	Off	Slow	Slow	Slow
Medium	Slow	Medium	Medium	Fast	Fast
High Borderline	Medium Fast	Medium Fast	Fast	Fast	Fast
High	Medium Fast	Fast	Fast	Fast	Fast

Based on data from the Lawrence Berkeley National Laboratory(http://www.iaqscience.lbl.gov/performance-temp-office.html) the following temperature membership function is defined:

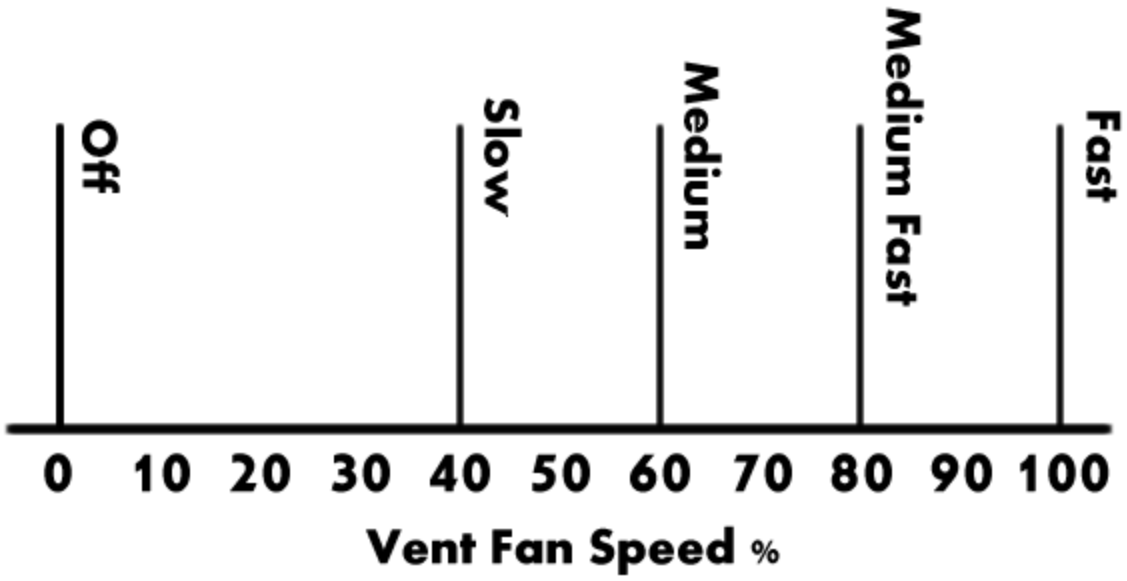


The membership function has a large width Comfortable zone as average home temperatures range ± 3 degrees around 72℉.

Relative humidity is a value which measures how much water the air is holding at a given temperature. Warmer air holds more water. For every degree change in temperature relative humidity changes by 2%. The EPA recommends an indoor relative humidity of 50-60%. Although this recommendation changes slightly with time of year, the average is between 50 and 60% RH.



The output of the FLC will be defuzzified using Center of Area defuzzification to determine the crisp output. Below is the Crisp Consequent from which the defuzzification will be based.



To achieve the above F.L.C. and collect the needed data an Arduino and a Combination temperature Humidity sensor(HIH-6130) will be used.

There are two distinct programs the first is the prototype JS program used as a proof of concept and testing. The second is the c program which runs on the arduino.The code follows.

```
/**
 * Evaluate a trapezoidal membership function at point n given corner points:
 * a b, c, d.
 *         b         c
 *         +---------+
 *        /           \
 *       /             \
 *      /               \
 * ----+                 +----
 *     a                 d
 * @param  n the x-value at which to evaluate the membership function.
 * @param  a the corner a value
 * @param  b the corner b value
 * @param  c the corner c value
 * @param  d the corner d value
 * @return   the degree of truth of the membership function at the value n
 */
function trapmf( n, a, b, c, d ) {
    var out = 0;
    if (n > a && n < b) {
        m=(0-1)/(a-b)
        out = m*(n-a)+0;
    }
    else if (n >= b && n <= c ) {
        out = 1;
    }
    else  if (n > c && n < d) {
        m=(1-0)/(c-d);
        out = m*(n-c)+1;
    }
    else if (n <= a || n >= d ) {
        out = 0;
    }

    return out;
}


var temp = 75;
var humidity = 65;

// evaluate the membership functions at the measured temperature
var cool            = trapmf(temp, 0,  0,  59, 62);
var somewhatCool    = trapmf(temp, 59, 62, 72, 72);
var comfortable     = trapmf(temp, 62, 72, 72, 82);
var somewhatHot     = trapmf(temp, 72, 82, 82, 85);
var hot             = trapmf(temp, 82, 85, 90, 90);

// evaluate the membership functions at the measured relative humidity
var low             = trapmf(humidity, 0,  0,  50,  60);
var lowBorderline   = trapmf(humidity, 50, 60, 60,  70);
var medium          = trapmf(humidity, 60, 70, 70,  80);
var highBorderline  = trapmf(humidity, 70, 80, 80,  90);
var high            = trapmf(humidity, 80, 90, 100, 100);

/**
 * Compute the Fuzzy AND of two numbers
 * @param  a first number
 * @param  b second number
 * @return   the AND value (min)
 */
function AND(a,b) {
    x = Math.min(a,b);
```

```
    return x;
}

/**
 * Compute the Fuzzy and of two numbers
 * @param  a first number
 * @param  b second number
 * @return   the OR value (max)
 */
function OR(a,b) {
    x = Math.max(a,b);
    return x;
}

var out_off = [];
var out_slow = [];
var out_medium = [];
var out_mediumFast = [];
var out_fast = [];

//row1 (of rule table)
out_off.push(AND(low, cool));
out_off.push(AND(low, somewhatCool));
out_off.push(AND(low, comfortable));
out_off.push(AND(low, somewhatHot));
out_off.push(AND(low, hot));
//row2 (of rule table)
out_off.push(AND(lowBorderline, cool));
out_off.push(AND(lowBorderline, somewhatCool));
out_slow.push(AND(lowBorderline, comfortable));
out_slow.push(AND(lowBorderline, somewhatHot));
out_slow.push(AND(lowBorderline, hot));
//row3 (of rule table)
out_slow.push(AND(medium, cool));
out_medium.push(AND(medium, somewhatCool));
out_medium.push(AND(medium, comfortable));
out_fast.push(AND(medium, somewhatHot));
out_fast.push(AND(medium, hot));
//row4 (of rule table)
out_mediumFast.push(AND(highBorderline, cool));
out_mediumFast.push(AND(highBorderline, somewhatCool));
out_fast.push(AND(highBorderline, comfortable));
out_fast.push(AND(highBorderline, somewhatHot));
out_fast.push(AND(highBorderline, hot));
//row5 (of rule table)
out_mediumFast.push(AND(high, cool));
out_fast.push(AND(high, somewhatCool));
out_fast.push(AND(high, comfortable));
out_fast.push(AND(high, somewhatHot));
out_fast.push(AND(high, hot));

/**
 * find the max value in an array
 * @param  arr    array to search
 * @param  arrlen length of array
 * @return        maximum value
 */
function ArrMax(arr) {
    max = arr[0];
    for (var i = 1; i < arr.length; i++) {
        if (arr[i] > max)
            max = arr[i]
    }
```

```
    return max;
}


//Calculate the OR of the array
console.log(out_off, ArrMax(out_off));
console.log(out_slow, ArrMax(out_slow));
console.log(out_medium, ArrMax(out_medium));
console.log(out_mediumFast, ArrMax(out_mediumFast));
console.log(out_fast, ArrMax(out_fast));

// now that we have anded the D.o.T. we need to find the OR(max) of each
var out_off_max         = ArrMax(out_off);
var out_slow_max        = ArrMax(out_slow);
var out_medium_max      = ArrMax(out_medium);
var out_mediumFast_max  = ArrMax(out_mediumFast);
var out_fast_max        = ArrMax(out_fast);

// Calculate the weighted average using the crisp consequent
var out = (out_off_max*0 + out_slow_max*40 + out_medium_max*60 + out_mediumFast_max*80 + out_f
ast_max*100)
out /= (out_off_max+ out_slow_max+ out_medium_max+ out_mediumFast_max+ out_fast_max);

console.log(out);
```

```
// HIH_6130_1  - Arduino
//
// Arduino                HIH-6130
// SCL (Analog 5) ------- SCL (term 3)
// SDA (Analog 4) ------- SDA (term 4)
//
// Note 2.2K pullups to 5 VDC on both SDA and SCL
//
// Pin4 ----------------- Vdd (term 8)
//


#include <Wire.h> //I2C library

#include "temp.h"
#include "printFloat.h"
#include "trapmf.h"

byte fetch_humidity_temperature(unsigned int *p_Hum, unsigned int *p_Temp);
void print_float(float f, int num_digits);
// float c2f(float f);

#define TRUE 1
#define FALSE 0

void setup(void)
{
        Serial.begin(9600);     // start serial output
        Wire.begin();           // start i2c communication

        delay(5000);            // wait for init to complete
        Serial.println(">>>>>>>>>>>>>>>>>>>>>>>>>");
}

void loop(void)
{
        byte _status;
        unsigned int H_dat, T_dat;
        float RH, temp;

        float out_off[8]                = {0};
        float out_slow[4]               = {0};
        float out_medium[2]     = {0};
        float out_mediumFast[4] = {0};
        float out_fast[12]              = {0};

        float cool            = 0;
        float somewhatCool    = 0;
        float comfortable     = 0;
        float somewhatHot     = 0;
        float hot             = 0;
        float low             = 0;
        float lowBorderline   = 0;
        float medium          = 0;
        float highBorderline  = 0;
        float high            = 0;

        float out_off_max         = 0;
        float out_slow_max        = 0;
        float out_medium_max      = 0;
        float out_mediumFast_max  = 0;
        float out_fast_max        = 0;
```

```
        while(1)
        {
                _status = fetch_humidity_temperature(&H_dat, &T_dat);
                if(_status != 0){
                        Serial.println("Abnormal state");
                }
                RH = (float) H_dat * 6.10e-3;
                temp = (float) T_dat * 1.007e-2 - 40.0;
                temp = c2f(temp);

                // evaluate the membership functions at the measured temperature
                cool            = trapmf(temp, 0,  0,  59, 62);
                somewhatCool    = trapmf(temp, 59, 62, 62, 72);
                comfortable     = trapmf(temp, 62, 72, 72, 82);
                somewhatHot     = trapmf(temp, 72, 82, 82, 85);
                hot             = trapmf(temp, 82, 85, 90, 90);

                // evaluate the membership functions at the measured relative humidity
                low             = trapmf(RH, 0,  0,  50,  60);
                lowBorderline   = trapmf(RH, 50, 60, 60,  70);
                medium          = trapmf(RH, 60, 70, 70,  80);
                highBorderline  = trapmf(RH, 70, 80, 80,  90);
                high            = trapmf(RH, 80, 90, 100, 100);

                //row1 (of rule table)
                out_off[0] = AND(low, cool);
                out_off[1] = AND(low, somewhatCool);
                out_off[2] = AND(low, comfortable);
                out_off[3] = AND(low, somewhatHot);
                out_off[4] = AND(low, hot);
                //row2 (of rule table)
                out_off[5]      = AND(lowBorderline, cool);
                out_off[6]      = AND(lowBorderline, somewhatCool);
                out_slow[0] = AND(lowBorderline, comfortable);
                out_slow[1] = AND(lowBorderline, somewhatHot);
                out_slow[2] = AND(lowBorderline, hot);
                //row3 (of rule table)
                out_slow[3]   = AND(medium, cool);
                out_medium[0] = AND(medium, somewhatCool);
                out_medium[1] = AND(medium, comfortable);
                out_fast[0]   = AND(medium, somewhatHot);
                out_fast[1]   = AND(medium, hot);
                //row4 (of rule table)
                out_mediumFast[0] = AND(highBorderline, cool);
                out_mediumFast[1] = AND(highBorderline, somewhatCool);
                out_fast[2]       = AND(highBorderline, comfortable);
                out_fast[3]       = AND(highBorderline, somewhatHot);
                out_fast[4]       = AND(highBorderline, hot);
                //row5 (of rule table)
                out_mediumFast[2] = AND(high, cool);
                out_fast[5]       = AND(high, somewhatCool);
                out_fast[6]       = AND(high, comfortable);
                out_fast[7]       = AND(high, somewhatHot);
                out_fast[8]       = AND(high, hot);

                // now that we have anded the D.o.T. we need to find the OR(max) of each
                out_off_max         = ArrMax(out_off, 8);
                out_slow_max        = ArrMax(out_slow, 4);
                out_medium_max      = ArrMax(out_medium, 2);
                out_mediumFast_max  = ArrMax(out_mediumFast, 4);
                out_fast_max        = ArrMax(out_fast, 12);

                // Calculate the weighted average using the crisp consequent
```

```
                float out = (out_off_max*0 + out_slow_max*40 + out_medium_max*60 + out_mediumF
ast_max*80 + out_fast_max*100);
                out /= (out_off_max+ out_slow_max+ out_medium_max+ out_mediumFast_max+ out_fas
t_max);

                print_float(RH, 1);
                Serial.print("  ");
                print_float(temp, 2);
                Serial.print("  ");
                print_float(out, 2);
                Serial.println();

                delay(1000);
        }
}

/**
 * Read the temperature and humidity from the sensor over the i2c bus.
 * based on code by Peter H Anderson, Baltimore, MD, Nov, '11
 * @param  p_H_dat [out] Relative Humidity
 * @param  p_T_dat [out] Temperature data. In celsius
 * @return         status. 1 indicates normal operation
 */
byte fetch_humidity_temperature(unsigned int *p_H_dat, unsigned int *p_T_dat)
{
                byte address, Hum_H, Hum_L, Temp_H, Temp_L, _status;
                unsigned int H_dat, T_dat;
                address = 0x27;;
                Wire.beginTransmission(address);
                Wire.endTransmission();
                delay(100);

                Wire.requestFrom((int)address, (int) 4);
                Hum_H = Wire.read();
                Hum_L = Wire.read();
                Temp_H = Wire.read();
                Temp_L = Wire.read();
                Wire.endTransmission();

                _status = (Hum_H >> 6) & 0x03;
                Hum_H = Hum_H & 0x3f;
                H_dat = (((unsigned int)Hum_H) << 8) | Hum_L;
                T_dat = (((unsigned int)Temp_H) << 8) | Temp_L;
                T_dat = T_dat / 4;
                *p_H_dat = H_dat;
                *p_T_dat = T_dat;
                return(_status);
}
```

```
#ifndef trapmf_h
#define trapmf_h

/**
 * Evaluate a trapezoidal membership function at point n given corner points:
 * a b, c, d.
 *          b         c
 *          +---------+
 *         /           \
 *        /             \
 *       /               \
 * ----+                 +----
 *     a                 d
 * @param  n the x-value at which to evaluate the membership function.
 * @param  a the corner a value
 * @param  b the corner b value
 * @param  c the corner c value
 * @param  d the corner d value
 * @return   the degree of truth of the membership function at the value n
 */
float trapmf (int n, int a, int b, int c, int d )
{
    float out = 0;
    if (n > a && n < b) {
        float m=(0-1)/(a-b);
        out = m*(n-a)+0;
    } else if (n >= b && n <= c ) {
        out = 1;
    } else if (n > c && n < d) {
        float m=(1-0)/(c-d);
        out = m*(n-c)+1;
    } else if (n <= a || n >= d ) {
        out = 0;
    }

    return out;
}

/**
 * find the max value in an array
 * @param  arr    array to search
 * @param  arrlen length of array
 * @return        maximum value
 */
float ArrMax(float arr[], int arrlen) {
    float max = arr[0];
    for (int i = 1; i < arrlen; i++) {
        if (arr[i] > max)
            max = arr[i];
    }
    return max;
}
/**
 * Compute the Fuzzy AND of two numbers
 * @param  a first number
 * @param  b second number
 * @return   the AND value (min)
 */
float AND(float a, float b) {
    if (a < b) {
        return a;
    } else {
```

```
        return b;
    }

}

/**
 * Compute the Fuzzy and of two numbers
 * @param  a first number
 * @param  b second number
 * @return   the OR value (max)
 */
float OR(float a, float b) {
    if (a > b) {
        return a;
    } else {
        return b;
    }
}

#endif
```

```
#ifndef Temp_h
#define Temp_h

/**
 * Celsius to Fahrenheit
 * @param  f Celsius temperature to convert
 * @return   Fahrenheit temperature
 */
float c2f(float f)
{
  float _temp = f*9.0;
  _temp /= 5;
  _temp += 32;
  return _temp;
}

#endif
```

```
#ifndef print_float_h
#define print_float_h

/**
 * Routine to print floating point numbers over serial.
 * @param f          the number to print
 * @param num_digits decimal precision
 */
void print_float(float f, int num_digits)
{
    int pows_of_ten[4] = {1, 10, 100, 1000};
    int multiplier, whole, fract, d, n;

    multiplier = pows_of_ten[num_digits];
    if (f < 0.0) {
        f = -f;
        Serial.print("-");
    }
    whole = (int) f;
    fract = (int) (multiplier * (f - (float)whole));

    Serial.print(whole);
    Serial.print(".");

    // print each digit with leading zeros
    for (n=num_digits-1; n>=0; n--) {
         d = fract / pows_of_ten[n];
         Serial.print(d);
         fract = fract % pows_of_ten[n];
    }
}

#endif
```