Condition monitoring and Management Systems for Agricultural Crops.

Kavan Mehta (823999846) Email - mehtakan117@gmail.com

Preetam Bamanalli (824684803) Email -pbamanalli4154@sdsu.edu

Proposed Idea:

In this project we had proposed that we would model, design and analyse the system which will control the temperature in the green house plantation.

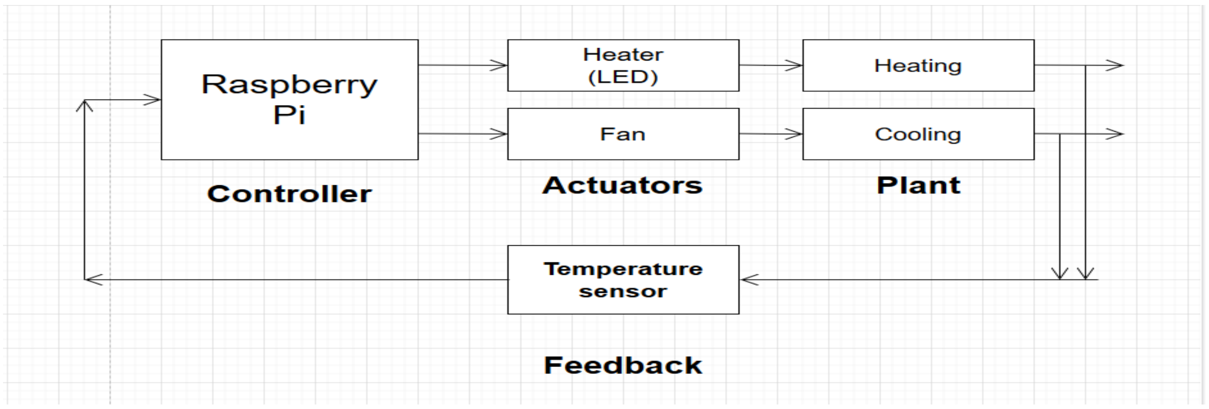
Accordingly, we have modelled the system using FSM, designed the system using raspberry pi, temperature sensors, cooling fan and LED and analysed the system using temporal logic and measured the response time of the system.

For decreasing the temperature, we have used fan. For increasing the temperature, we have used the LED to represent the heating effects. We have implemented the system using Raspberry pi which will either turn on the fan or the LED based on the data received from the temperature sensor.

Also, we had proposed to model the system for the varying fan speed for effective control of temperature.

CPS Concept developed:

We have developed all the three important themes of Cyber physical system i.e. Modelling of the system, Design of the system and Analysis of the system which will be explained in detail in next parts. The physical part of the system is LED(Heating) and Fan. The feedback is achieved by temperature sensor.

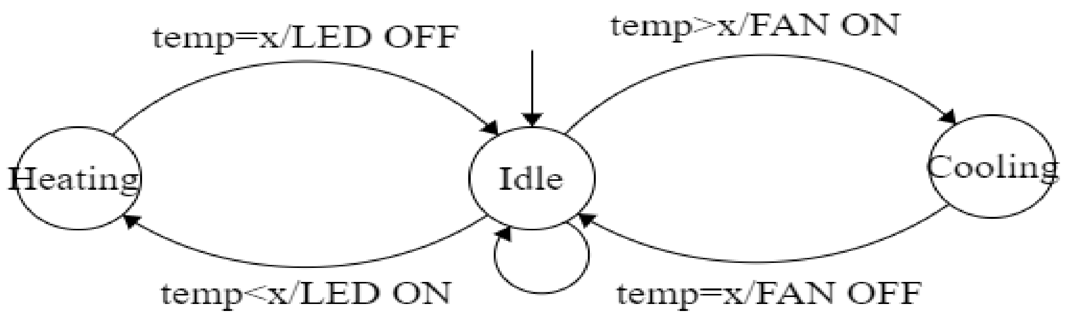


Schematics of Closed Control System

Modelling of the system:

First, we modelled our system using Finite state machine (FSM). This FSM have 3 states i.e. ‘Idle’, ‘Cooling’ and ‘Heating. So, suppose the standard temperature required is 25°C and if the temperature sensor reads temperature more than 25°C than FAN will be ON, and it will go to Cooling state.If the temperature is equal to 25°C than it will go to Idle state. And if the temperature is below 25°C than it will go to Heating state and LED will be ON.

Condition monitoring and Management Systems for Agricultural Crops.



Finite State Machine showing 3 states: Idle, Heating & Cooling

Notations:

Temp = current temperature;

X = standard required temperature;

Initial state: Idle;

Input: Temp;

Outputs: fan: pure (on/off), light: pure (on/off);

Update:

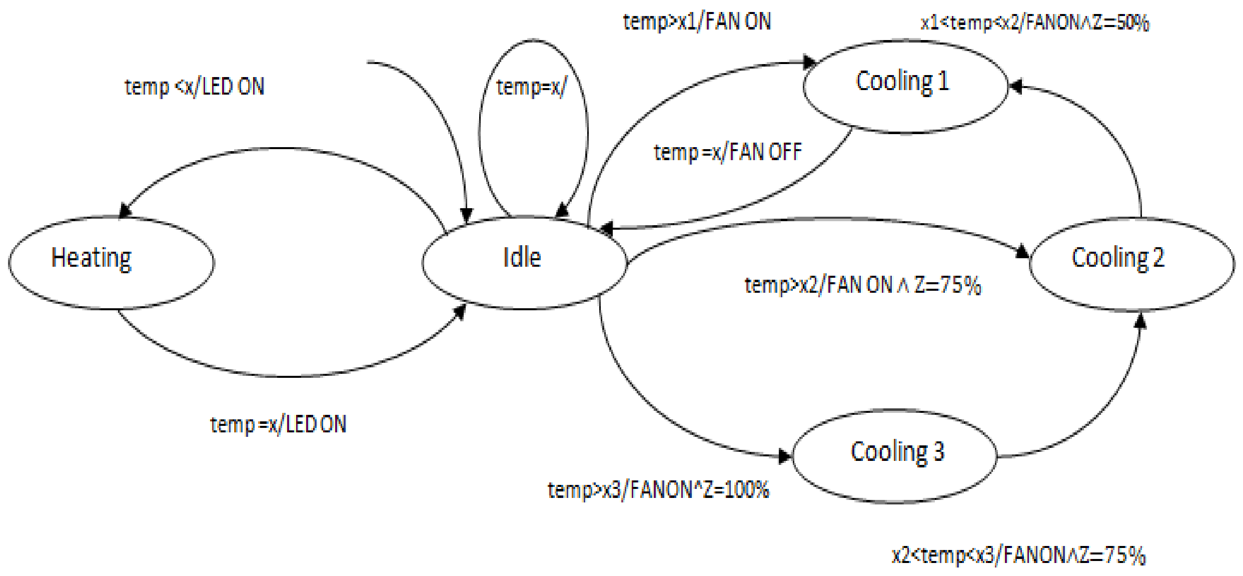
(Cooling, fan on),	when Idle ^ i(Temp > x),
(Heating, light on),	when Idle ^ i (Temp < x),
(Idle, fan off),	when Cooling ^ i (Temp = x),
(Idle, light off),	when Heating ^ i (Temp = x),
(Idle ;),	when Idle ^ i (Temp = x);

Modelling the system with varying Fan Speed:

In this part we decided to change the fan speed as per the temperature requirements. For doing this we decided to change the fan speed by changing its duty cycle.

So, suppose the standard temperature required is 25°C and if the temperature sensor reads 26°C than the FAN will be on with duty cycle 50% (State = Cooling 1).If the temperature is raised to 28°C than FAN speed will be increased to 75% duty cycle(State = Cooling 2).If the temperature is raised to 30°C than the FAN speed will be increased to 100% duty cycle (State = Cooling 3).Accordingly, the state transitions and fan speed variation will occur depending on the temperature variation.

Condition monitoring and Management Systems for Agricultural Crops.



Extended FSM for varying FAN rpm

Notations:

Temp = current temperature;

X = standard required temperature;

Where x1>x2>x3 (varying temperatures)

Initial state: Idle;

Input: Temp;

Outputs: fan: pure (on/off), light: pure (on/off);

Update:

(Cooling 1, fan on with duty cycle = 50%), when Idle ^ i(Temp > x1),

(Cooling 2, fan on with duty cycle = 75%), when Idle ^ i(Temp > x2),

(Cooling 3, fan on with duty cycle = 100%), when Idle ^ i(Temp > x3),

(Cooling 1, fan on with duty cycle = 50%), when Cooling 2 ^ i(x1<temp<x2),

(Cooling 2, fan on with duty cycle = 75%), when Idle ^ i(x2<temp<x3),

(Heating, light on), when Idle ^ i (Temp < x),

(Idle, fan off), when Cooling1 ^ i (Temp = x),

(Idle, light off), when Heating ^ i (Temp = x),

(Idle ;), when Idle ^ i (Temp = x);

Condition monitoring and Management Systems for Agricultural Crops.

Designing of the System:

Raspberry Pi 3: For the designing of the system we used Raspberry pi 3 as it has GPIO pins which is required for interfacing temperature sensors and led and Fan. It has PWM pins which we required for controlling Fan speed.

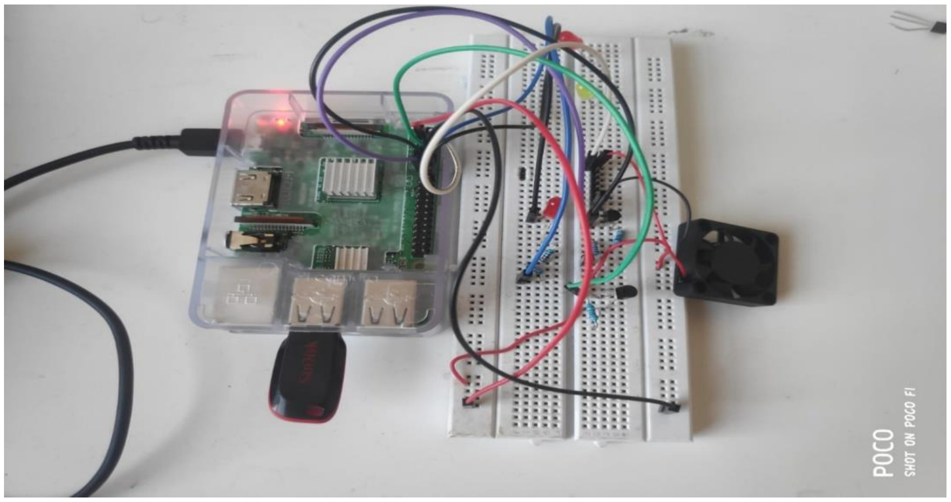
Temperature Sensor: We used temperature sensor DS18B20 for sensing the environment temperature. We specifically used DS18B20 for two main reasons:

- The DS18B20 communicates with the "One-Wire" communication protocol, a proprietary serial communication protocol that uses only one wire to transmit the temperature readings to the microcontroller. So, by sing "One-Wire" protocol we can connect multiple sensors to the microcontroller as we wanted to connect multiple sensors for handling inaccurate sensor reading for system robustness.
- The second reason is that DS18B20 calculates the CRC value and provides it to the microcontroller; this is helpful in detecting the transmission error of the data which was again a major requirement of our project.
- Third reason is that the range of the temperature it can detect is good, -55 to 125°C

Led: For increasing the temperature will have used LED to represent the heating effect.

Fan: For decreasing the temperature, we have used small 5V cooling fan.

NPN transistor: We have used NPN transistor PN2222 for switching the Fan on and off.The Pi's GPIO pins are only capable of supplying 3.3V and the Pi's 5V pins are connected directly to the power supply and cannot be controlled via software. Therefore, we needed a way to power the fan using the 5V pin and switch it on and off using a GPIO pin and accordingly we used NPN transistor.



Analysis of the system:

For the analysis of the system we have used Linear Temporal Logic to create LTL formulas which is:

G ((x) ==> F(y))

This is valid for all the trace of our extended FSM which will be explained below.

Condition monitoring and Management Systems for Agricultural Crops.

Temporal Logic (LTL):

Inputs: State (Initial state), temp;

Output: y (fan, Led), Z (Duty cycle);

LTL formulas:

G ((temp=x ∧ State = Heating) ==> F(y=LED OFF))

G ((temp<x) ==> F(y=LED ON))

G ((temp=x ∧ State = Cooling) ==> F(y=FAN OFF))

G ((x1<temp<x2 ∧ State = Idle) ==> F(y= FAN ON ∧ Z = 50%))

G ((x2<temp<x3 ∧ State = Idle) ==> F(y= FAN ON ∧ Z = 75%))

G ((temp>x3) ==> F(y= FAN ON ∧ Z = 100%))

G ((x1<temp<x2 ∧ State = Cooling 2) ==> F(y= FAN ON ∧ Z = 50%))

G ((x2<temp<x3 ∧ State = Cooling 3) ==> F(y= FAN ON ∧ Z = 75%))

Approach to the solution:

Temperature sensor DS18B20 directly provides us with the digital temperature reading. To access the temperature reading we must access the ROM of the sensor where the value is store. To do that we have created methods "*read_temp_raw_1*" and *"read_temp_raw"* and "*read_temp*". In the read_temp method id used to give the other methods that make use of temperature with the correct value. (The method to identify the faulty sensor and provide other methods with correct value is explained in this section later)

The software is implemented based on the modelling. There are three methods that we have used to represent the states i.e. *"idle_read ()"*, *"led_heating ()"* and *"fan_cooling ()*". In modelling the initial state is idle likewise in the software implementation idle_read is called (executed) first. Based on the temperature reading (from the model) the methods led_heating and fan_cooling is called. In the model we get back to the idle state from the heating and cooling state when the desired temperature is achieved likewise in the software at the end of the end of the methods led_heating and fan_cooling, idle_read method is called. The guards in the modelling are implemented using the while and if statements. While statements are used in all the above-mentioned methods. While statements keep the methods cooling, heating and idle running until the desired temperature is achieved. If statements are used to change the states from idle to either heating or cooling.

The varying fan RPM is implemented using the PWM pins of the raspberry pi and PWM software to control the duty cycle. The RPM of the Fan is controlled using different duty cycles. Statement "p = GPIO.PWM (17, 50)" is used to set GPIO pin 17 of the PWM mode with 50% duty cycle, "p. start (duty cycle)" function is used to change the duty cycle. Multiple if and while statements are used to implement the guard similar to the model.

To get the time at which control of program goes into the idle_read, fan_cooling and led_heating method, time.time () function is used. The difference in the time when the control enters the idle_read and when the control enters led_heating method, gives us the time to move the state from idle to heating. Likewise, time to move the state from idle to cooling is determined.

Condition monitoring and Management Systems for Agricultural Crops.

System robustness is of most important for monitoring and controlling the temperature of greenhouse. System shouldn't heat the environment when it is supposed to be cooling or vice versa due to the inaccurate sensor reading, which could be hazardous to the crops. Actuator (fan and heater) failing to activate even when sensor readings are accurate also can be hazardous. To handle the sensor inaccuracies and actuator failures we have used redundancy.

The temperature sensor DS18B20 calculates the CRC value and provides it to the microcontroller. This leveraged in our software to eliminate transmission error. The value 'CRC = YES' will be provided by the sensor if the CRC check holds good. If the value is anything other than YES, we wait for 0.2 sec until and read the CRC value again. We keep doing this until the CRC value says YES (statement in the code while lines [0].strip () [-3:] != 'YES'). This solution is implemented in the read_temp_raw method.

To detect the faulty sensors among the three we have implemented the following algorithm in *"read_temp"* method:

1. Get the reading from the 3-temperature sensor.
2. Subtract the values from one another i.e. t1-t2, t2-t3 and t3-t1 and store it in three variables a, b and c respectively.
3. When one of the temperature sensors is faulty either a,b or b,c or c,a will have values greater than 1.
4. If values a and b are greater than 1 then it means that t2 is faulty.
5. If values b and c are greater than 1 then it means that t3 is faulty.
6. If values a and c are greater than 1 then it means that t1 is faulty.
7. After determining the faulty sensor, the temperature reading from the correct sensor is taken for the rest of the process.

To detect the actuator failures, we have developed the following algorithm. This is not implemented in our code. But we have the sudo code in the archive folder.

For Fan

- Calculate the average time required by the fan to reduce the temperature by 1 degree
- Wait for the average time that a fan takes to reduce the temperature by 1 degree
- Check if the temperature is reduced to 1 degree.
- Turn off the first fan and turn on the second fan.

For Heater

- Calculate the average time required by the heater to increase the temperature by 1 degree
- Wait for the average time that a heater takes to increase the temperature by 1 degree
- Check if the temperature is increased to 1 degree.
- Turn off the first heater and turn on the second heater.

The codes for single sensor, multiple sensors, varying fan speed and time response are available in the archive folder.

<u>Experimental Design:</u>

1. **Success metrics:**
   Our system works as we modelled.

1. Temperature readings are accurate.
2. Fan turns on when the temperature is more than 25°C and should turn off when the temperature is 25°C.
3. LED turns on when the temperature is less than 25°C and should turn off when the temperature is 25°C.

Condition monitoring and Management Systems for Agricultural Crops.

Below is the time response of the system with single sensor implementation vs the multi-sensor implementation.

With Redundant sensors		
Experiments	Idle to Cooling	Idle to Heating
1	2.644	2.626
2	2.695	2.681
3	2.735	2.709
4	2.676	2.669
5	2.689	2.641
6	2.652	2.649
7	2.697	2.668
8	2.63	2.653
9	2.701	2.65
10	2.667	2.637
Average	2.6786	2.6583

With Redundant sensors		
Experiments	Idle to Cooling	Idle to Heating
1	0.861	0.924
2	0.9	0.93
3	0.927	0.92
4	0.895	0.868
5	0.927	0.911
6	0.867	0.904
7	0.889	0.931
8	0.874	0.915
9	0.907	0.881
10	0.894	0.892
Average	0.8941	0.9076

Redundancy adds overhead to the software. We access the temperature from the ROM of the sensor and there are two methods to do this in our software implementation. i.e. *"read_temp_raw_1" and* "*read_temp_raw"*.In the method *"read_temp",* where identifying the faulty sensor algorithm is implemented each of the earlier mentioned methods ("*read_temp_raw_1" and* "*read_temp_raw"*) are called three times (because of three sensors). So, you can see that the average time response of system with redundancy is three times the system without redundancy, which is expected.

2. Experiments performed:

Different number of experiments we performed to check if the system is working to our specification. Below is the list of experiments.

1. We checked if the fan can be turned on using the GPIO pin, with a small code that is for this purpose only.
2. We checked if the temperature readings are accurate.
3. We checked if the RPM can be changed using PWM GPIO pins and programming.
4. We checked if the Fan turned on when the temperature is greater than 25°C by manually increasing the temperature of the sensor.
5. We checked if the LED turned on when the temperature is less that 25°C by manually cooling the temperature sensor using a frozen carrot.
6. We checked if the Fan is running at duty cycle of 50% when the temperature is >25°C but less than <27°C.
7. We checked if the Fan is running at duty cycle 75% when the temperature is >27°C but less than <29°C
8. We checked if the Fan is running at duty cycle 100% when the temperature is >29°C.
9. We checked if the system is detecting the faulty sensor among the three.
10. We checked the time if the time response of the single sensor implementation is less than the time response of the multi-sensor implementation.

Condition monitoring and Management Systems for Agricultural Crops.

All the above-mentioned experiments were successful. And the system we developed works as proposed.

3. Results and their significance:

The robustness of our system depends on the reliability of the sensors. The system can detect the faulty sensor and take the values from the working sensor.

However, the average time response of multi-sensor is much greater than the single sensor implementation, reliability is of most importance. The average difference of 2 sec does not impact the system much since we are controlling the temperature for every single degree it crosses the threshold mark

4. Missing Milestones:

There are no missing milestones. We had only proposed to model the varying fan speed system. But we have implemented it and tested if the system works.

We have delivered more than what was initially proposed.

5. Challenges faced:

We got the modelling of varying fan speed after a lot of tries. Coming up with an algorithm to handle sensor inaccuracies and actuator failures were challenging. Also implementing the algorithm in software was quite challenging.

6. Possible extensions to the project:

Moisture, water level and many other conditions required for the crops can monitor and controlled. Varying heater levels can be added to save some power and to have effective control over the temperature.

Many such devices can be networked, and data can be collected on how to perfectly control all the required conditions for higher yield of crops using Machine learning.

References:

1. An Embedded Fuzzy Logic Microcontroller for Plant Condition Monitoring: A Wireless Sensor Network Odyssey (By Malusi Sibiya, Mbuyu. Sumbwanyambe).
2. A Wireless Inductive Heater System for Horticulture Crop Protection. (SDSU CompE 571 project by Luis and Jerald).
3. Images from Introduction to Embedded Systems, a cyber-physical systems approach by Lee and Seshia.
4. https://howchoo.com/g/ote2mjkzzta/control-raspberry-pi-fan-temperature-python.
5. https://www.circuitbasics.com/raspberry-pi-ds18b20-temperature-sensor-tutorial/.
6. https://www.raspberrypi.org/forums/viewtopic.php?f=63&t=244194.
7. https://gist.github.com/enwi/79954163abecd1d98af0cdeff13c358d