ESIOT 2024-2025

Assignment #3

Smart Temperature Monitoring

Authors: Jiekai Sun, Weijie Fu

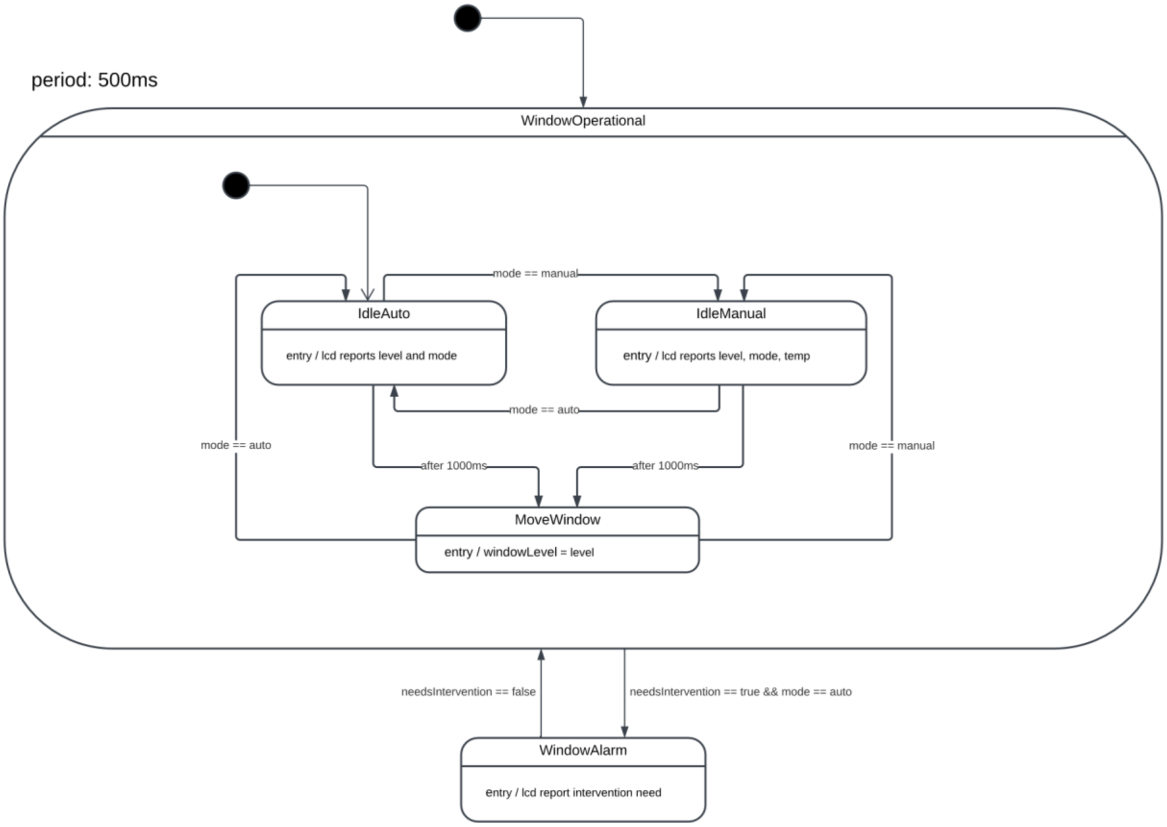
Window Controller:

The Arduino system has been implemented with a synchronous FSM and a cooperative scheduler assuming a tick period of 100ms, enough to handle all tasks even in the worst case execution time.

Below is a table with each WindowController task and their corresponding period:

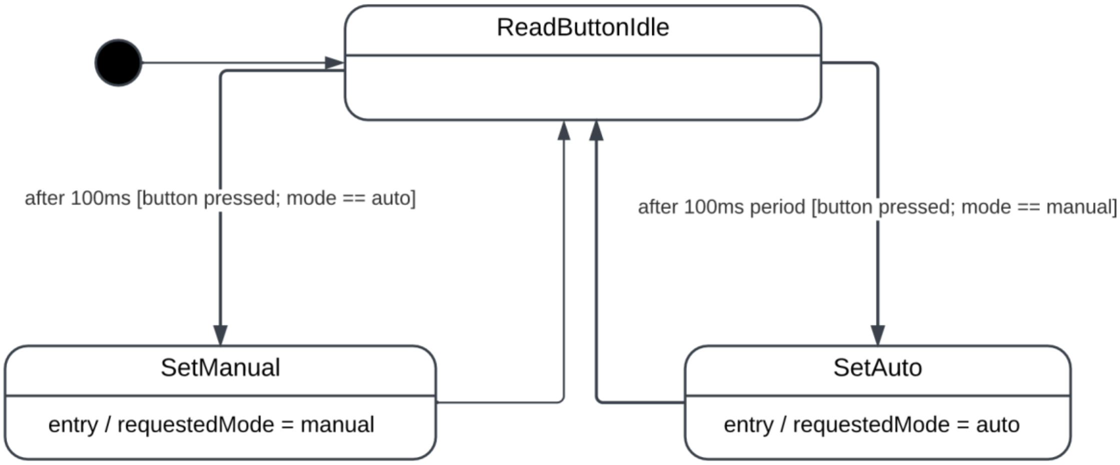
Priority	Task	Description	Period
1	ReadPotentiometer	Reads the potentiometer value	1000ms
2	ReadButton	Handles the operating mode switch request	100ms
3	SendMessage	Sends windowLevel and requests to switch mode to the backend	1000ms
4	ReceiveMessage	Parses incoming message and sets new parameters	1000ms
5	WindowControlTask	Handles the core logic of the window	500ms

Here is the FSM of the core logic of the WindowController:

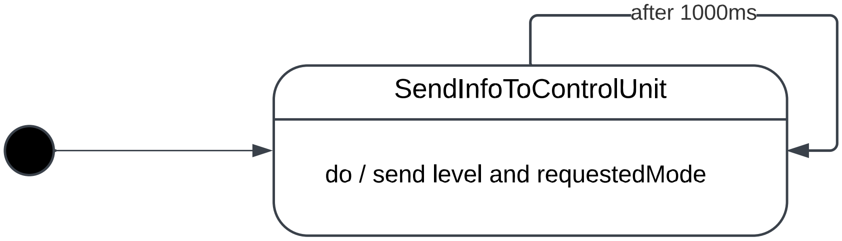

period: 500ms
WindowOperational
IdleAuto
entry / lcd reports level and mode
IdleManual
entry / lcd reports level, mode, temp
mode == manual
mode == auto
mode == auto
mode == manual
after 1000ms
after 1000ms
MoveWindow
entry / windowLevel = level
needsIntervention == false
needsIntervention == true && mode == auto
WindowAlarm
entry / lcd report intervention need

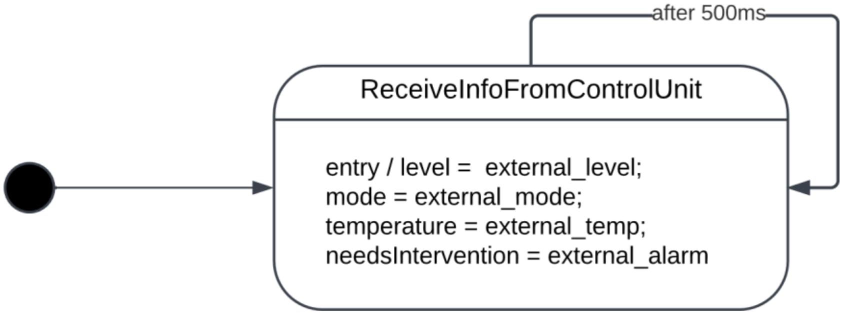
As for the ability to switch mode from AUTO to MANUAL and vice versa via button press, we realized that handling it directly in the Arduino subsystem led to synchronization issues with the backend.

Therefore, if the button is pressed a “request” to switch mode is queued.

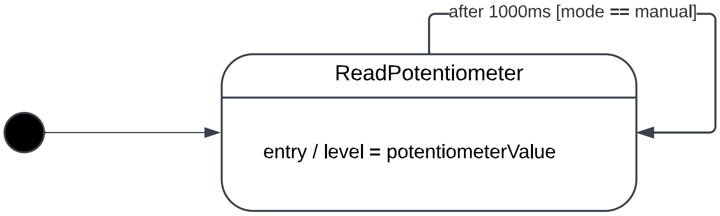

ReadButtonIdle
after 100ms [button pressed; mode == auto]
after 100ms period [button pressed; mode == manual]
SetManual
entry / requestedMode = manual
SetAuto
entry / requestedMode = auto

Such request is then sent upstream. The backend will then evaluate the switch request and the Arduino system will change its operating mode as dictated by the Control Unit along with other parameters.




after 500ms
ReceiveInfoFromControlUnit
entry / level = external_level;
mode = external_mode;
temperature = external_temp;
needsIntervention = external_alarm

Lastly, we also have a task specifically designed for the potentiometer:

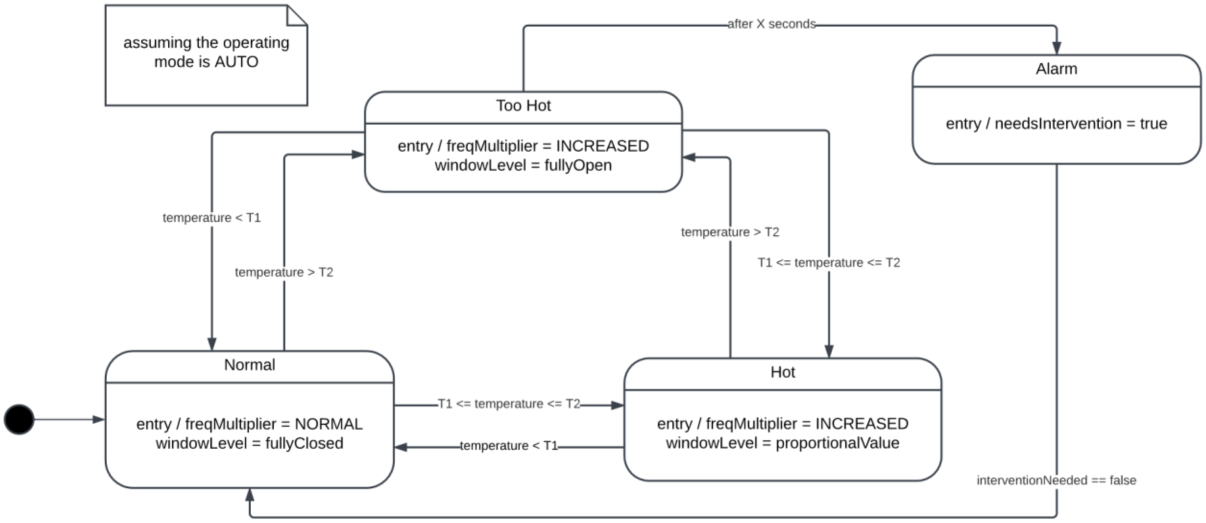

after 1000ms [mode == manual]
ReadPotentiometer
entry / level = potentiometerValue

Control Unit:

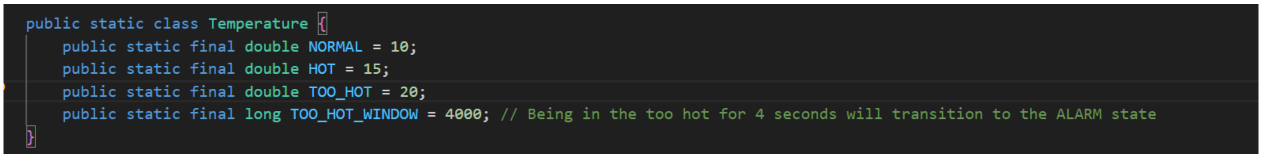
The control unit acts as the master in the whole system. It dictates the frequency at which the temperature is sampled, the operating mode and opening level of the window.

If any of the subsystems requires to change a parameter, they perform requests, not immediate changes. They will know if anything has changed through the data the control unit provides them.

Below is a FSM of the core decision logic of the backend:

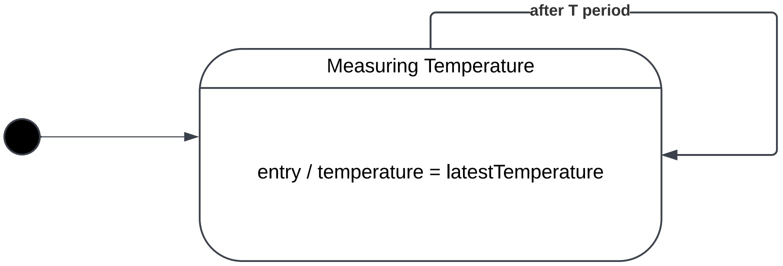

assuming the operating mode is AUTO
after X seconds
Alarm
entry / needsIntervention = true
Too Hot
entry / freqMultiplier = INCREASED
windowLevel = fullyOpen
temperature < T1
temperature > T2
temperature > T2
T1 <= temperature <= T2
Normal
entry / freqMultiplier = NORMAL
windowLevel = fullyClosed
T1 <= temperature <= T2
temperature < T1
Hot
entry / freqMultiplier = INCREASED
windowLevel = proportionalValue
interventionNeeded == false

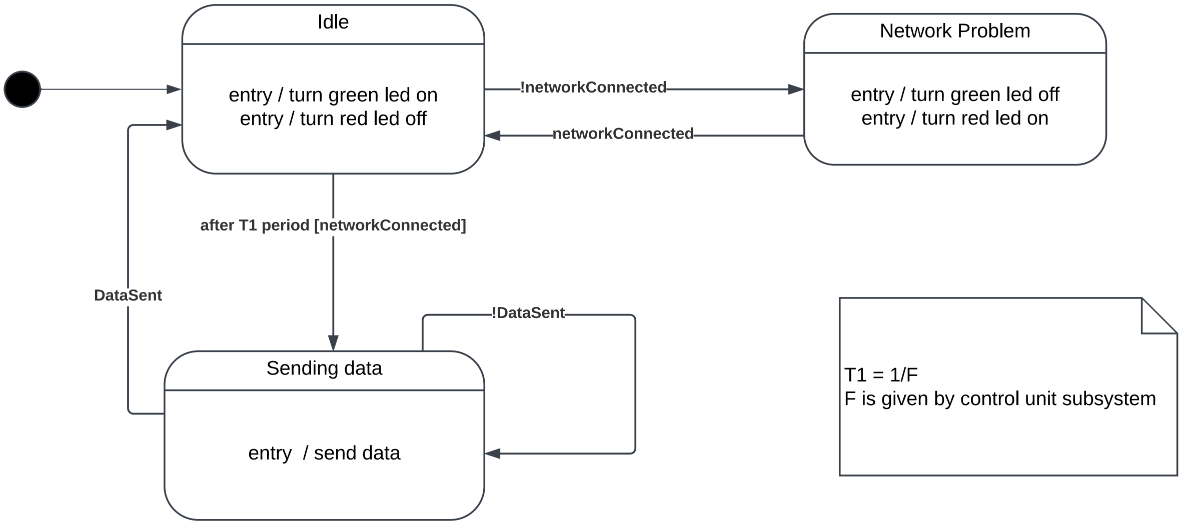
Below are the temperature thresholds and alarm trigger window:



```
public static class Temperature {
    public static final double NORMAL = 10;
    public static final double HOT = 15;
    public static final double TOO_HOT = 20;
    public static final long TOO_HOT_WINDOW = 4000; // Being in the too hot for 4 seconds will transition to the ALARM state
}
```

Temperature Monitoring Subsystem:





The ESP32 subsystem is supposed to measure the temperature and publish it to a MQTT broker under a topic named “temperature”.

It’s also subscribed to the topic “frequency”. Upon a change it will change the pace at which temperature samples are published.

By default temperature read and upload happen with a period of 1000ms.
When requested the frequency ratio is changed from 1 to 1.5, meaning a period of around 667ms.

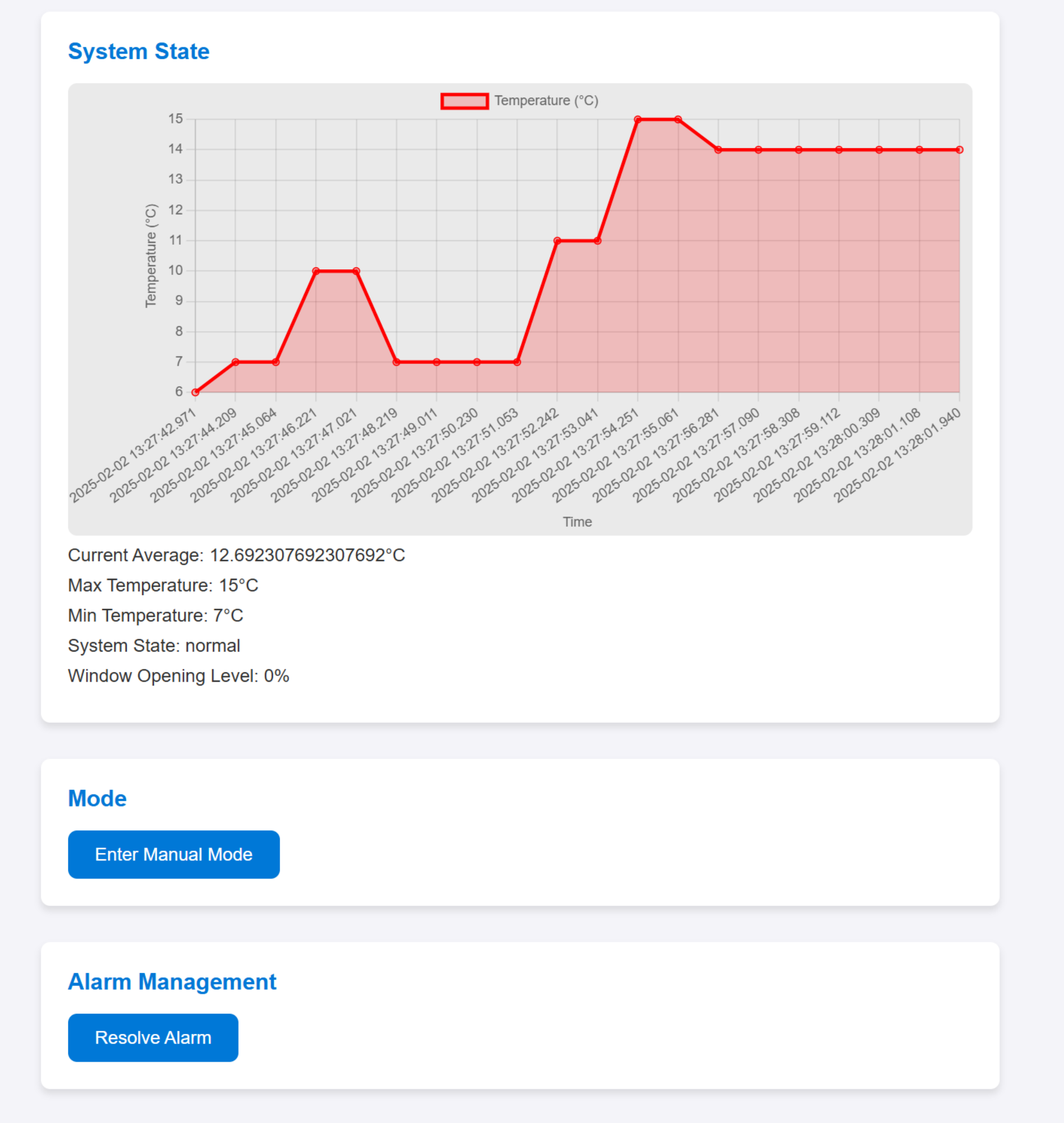
Such period can be reverted back once temperature is normal (decided by the Control Unit).

Dashboard:

The front-end is a very simple webpage made with Bootstrap and Chart.js .
It reports the temperature samples in a graph and updates aggregate information.

Note: aggregate statistics refers to specific time intervals and may not match the latest samples in the graph.

Below is a snapshot of a the system when the temperature is normal:



System State

Current Average: 12.692307692307692°C

Max Temperature: 15°C

Min Temperature: 7°C

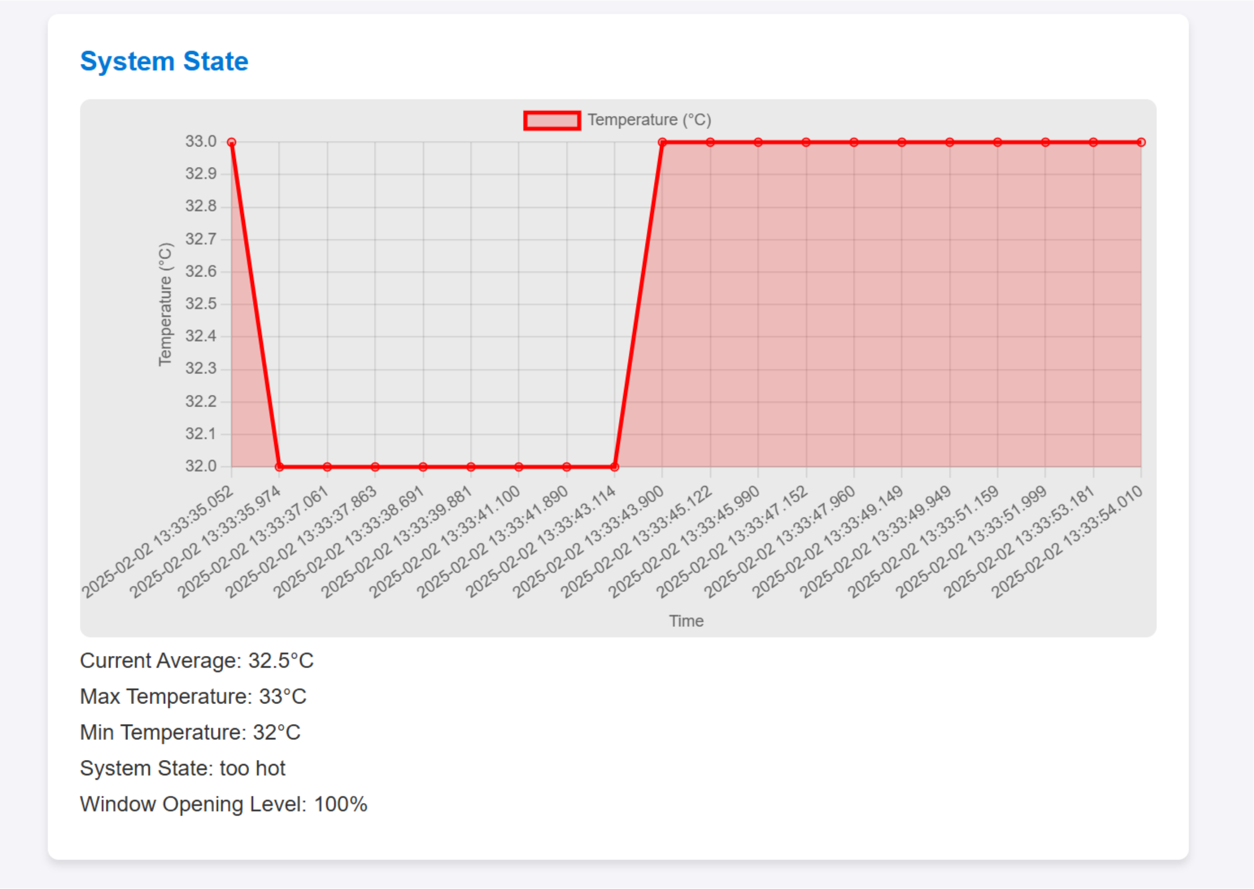
System State: normal

Window Opening Level: 0%

Mode

Alarm Management

And now a high temperature scenario:



System State

Current Average: 32.5°C

Max Temperature: 33°C

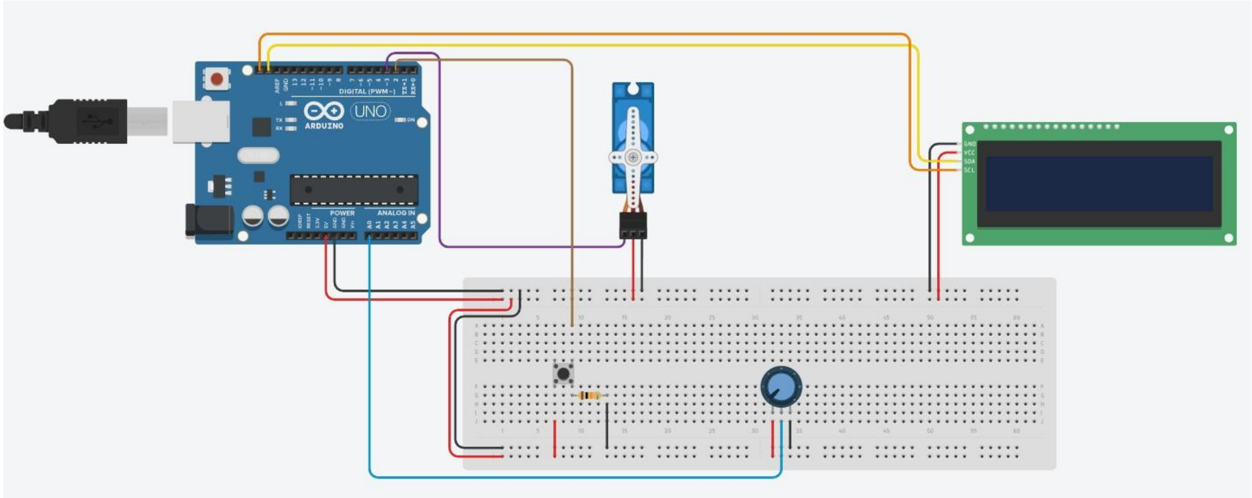
Min Temperature: 32°C

System State: too hot

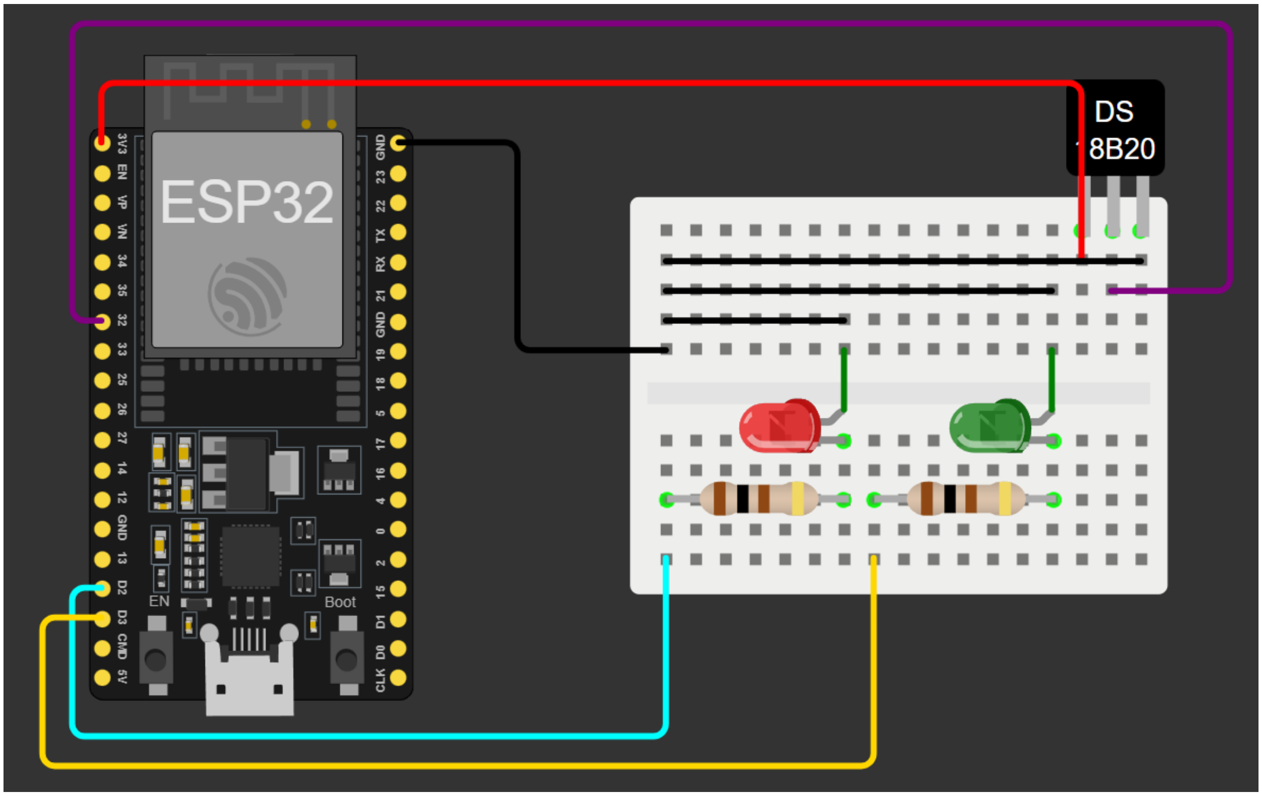
Window Opening Level: 100%

Note how the window opening level reports 100% (the window is supposed to be fully open in order maximize airflow and cool the system down).

Window Controller circuit:



ESP32 circuit:



Demo video: assignment3.mp4