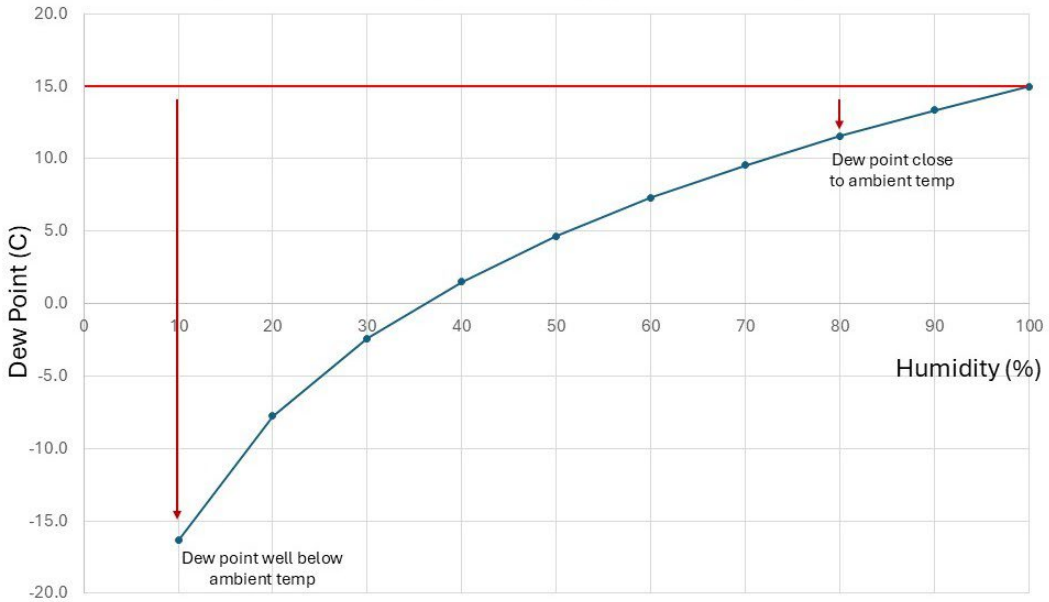
ARDUINO DEW HEATER CONTROLLER

Contents

1 INTRODUCTION

Dew formation is a major problem, especially under high humidity conditions, affecting the optics to degrade the images. Dew forms when the temperature of the glass is at or just below the dew point, and this gets worse as humidity increases. Things are made worse because the glass element often face towards the sky and can be colder than other parts of the telescope. At low humidity the dew point is well below ambient temperature, but at high humidity the dew point rapidly approaches ambient temperature (Fig 1.1, graph of dew point versus humidity at ambient temp of 15C).

Fig. 1.1. Dew Point (C) v Humidity (%) @ Ambient Temp 15˚C



Dew formation can be overcome by raising the temperature of the optics with a heater. Generally, more heating is required as humidity rises as the dew point is close to ambient. There are plenty of devices out there that help with these issues. The simplest is constant heating through a heater strap around the glass element to. You can use these on most parts of a telescope:

- around the objective of a refractor scope & guidescope (around the dew shield)
- the secondary but not the primary of a Newtonian (due the heating up eddy currents)

- the front corrector plate of an SCT (I had some success with this on my first build, but it can be a problem because of its size and exposed location).

1.1 Why use an Arduino for a dew heater:

Why use one when you could just get a PWM controller and modify it from light control to manual dew heater control.

- its fun and its not too complicated: just an Arduino and a few modules.
- Constant heating is fine, but how do you know how the level of heat needed to stop dew formation. Making it too hot wastes battery power and can degrade the image quality through excessive thermal currents; not hot enough and dew will form on the glass.

It's surprising how little heating is actually needed to stop dew formation. A device which uses information from the environment to set the heater power automatically can be used to keep the optics above the point at which dew forms. The aim here is an automatic heat controller to stop dew formation under varying conditions with minimal heating. With the automatic-heater control, I just always have it set running – half the time it doesn't even turn on because of low humidity, but on 99% of those bad nights it takes care of itself.

2 BUILD OPTION - BASICS

The are 2 basic things that affect build choice:

1. The level of automation in the control. There are 2 automatic heater modes:
    a. auto-ambient mode: this uses ambient temperature/humidity to determine the heater power level. This mode is available in several commercial dew heaters. While it is not too difficult to make, it is limited. ***This is in options 1 & 3.***
    b. auto-heater mode: this uses ambient temperature/humidity plus temperature of the optics to determine heater power level. This feedback control is much better than (a) but is more complicated to make than to make. ***This is in options 2 & 4.***
2. How the dew heater controller parameters are set (and displayed). There are two ways:
    a. Remote control using a PC windows app (USB2 connected): good if you want remote control and display. You can just display current settings and reading on the PC. Can also optionally add a display on the device. ***This is in options 1 & 2.***
    b. Local display and control on the device: good if you want display and control outside at the scope – good for visual obervers. ***This is in options 3 & 4.***

I am using ***option 2*** (auto-heater & PC control) without a display as I do imaging: scope is outside and I am inside. My rig has a mini-PC that runs everything (NINA etc). The dew heater can be checked from inside via wifi & remote desktop on my phone or indoor computer. Having said this, I rarely have the dew heater controller connected – it just runs as a standalone device (*but more on that later*). Many are using ***option 2*** with remote control and 16*2 LCD display.

2.1 Heater Control Modes

The project includes two levels of automatic control to keep optics above the dew point, plus 2 other backup/manual modes.

2.1.1 Auto-ambient mode

This mode senses the ambient temperature and humidity to determine the dew point. The Arduino device sets the heater power to a level determined by the difference between the ambient temperature and the dew point. Heater power increases as *'ambient temperature - dew point'* decreases. So,

- When ambient temperature is well above the dew point (at low humidity) there is a low chance of dew formation. Here the dew heater is turned OFF.
- When ambient temperature is equal to the dew point (~ 90-100% humidity) there is a high chance of dew formation. Here the dew heater is turned ON, maximally.
- At in-between ambient temperatures, the dew point (~ 50-80% humidity) there is an increasing chance of dew formation. Here the dew heater is turned ON at intermediate levels.

Advantages:

- the simplest auto control: heater control only needs a sensor to detect ambient conditions.
- simple cable to the heater – just a 2-core cable. So you can use premade heaters.
- it works quite well as an automatic controller.

Disadvantages:

- you really don't know the actual temperature of the glass element being heated. This will depend on how powerful the heater is, size of glass being heated, etc. So you have to try it in various conditions to determine the optimal parameters.
- Consequently, can possibly overheat glass – inefficient and can cause heat related distortions.
- So you have to set it up beforehand. Even then it will vary with different optics you are heating.

2.1.2 Auto-heater mode

This mode has the above ambient sensor plus a temperature sensor on each of the heaters to measure temperature of the glass element. The Arduino device calculates the temperature of the heater relative to the ambient dew point and uses a PID feedback controller to set the glass element to a set temperature (just above dewpoint). This mode determines heater power from *'heater temperature – set point'*** and uses PID feedback control to set the glass element to a set temperature. The heater power increases as that difference increases. So,

- When heater temperature is well above the heater set-point, the dew heater is turned OFF.
- When heater temperature is well below the heater set-point, the dew heater is turned ON.
- At in-between temperature differences, the dew heater is turned ON at levels determined by PID control.

** It's actually a bit more complicated. You choose a set-point value relative to dew point, e.g. a setting of +4C = 4C above dew point. The set-point temperature is calculated as the maximum of *'set point above dew point'* and *'ambient temperature* '. This takes into consideration how close the dew point is to the ambient temperature, so with a set point of +4C at ambient temp of 20C:

- When dew point is well below ambient temperature, the set-point target = ambient temperature, e.g.
  - At humidity 60% dew point=12C: set-point temp = ambient = 20C; as ambient – dew point >4C.
- When dew point is near ambient temperature, the set-point target = dew point + set-point, e.g.
  - At humidity 90% gives dew point=18C: set-point temp = dew point + 4C = 22C; as ambient – dew point <4C.

Advantages:

- feedback control keeps the heater at a temperature just high enough to stop dew formation.
- There's no need to figure out anything to set it up – just set the heater target temp to 2-4 degrees above dew point (as the glass will be slightly cooler than the heater).

Disadvantages:

- In addition to an ambient sensor, each heater needs a cable with 4x rather than 2x wires - 2 for the heater and 2 for the temperature sensor (or can use 2x separate 2-core wires). So if you have a

commercial dew heater strap you'll have to add another 2-core wire, or replace the 2-core wire with a 4-core one.

- Also, a heater sensor has to be wired up in the heater. So, more complicated to build.

2.1.3 Non-automatic (backup) modes

a. Manual mode: This mode just sets the power to a set level (% of maximum). Don't make this project if this is all you want - its overkill !! I have only included as a backup mode in case any of the sensors malfunction (can happen as we are mixing dew & electrical devices).
b. Off mode: This is also a backup mode in case the temperature of any of the glass elements increases above a cut-off level to avoid over-heating.

The auto-heating modes actually reduce the heating power. Over a 6-hour test period one night with ambient conditions ranging 8-12C & 70-80% humidity power was:

- Manual mode- 10% power.
- Auto-ambient - an average of 6.7% power
- Auto-heater- an average of 2.5%.

2.2 Display & control types

There are two things here that affect build choice, (i) control - via PC or button on device, and (ii) display - via the PC and/or and LCD/OLED display on your device.

2.2.1 PC control & display

If you are using a PC:

a. Control is via the PC: this can be used to set the heater mode and the settings of for modes.
b. Display is also on the PC: display (i) ambient temp, humidity & dew point, (ii) heater channel mode, temperature, power etc (varies with modes).
c. A few have said that they use PC-control but would also like an LCD/OLED display on the device. With the latest version (6.3) you can add a display with PC-control but it will only display dew heater conditions, it cannot be used to set heater mode/parameters.

2.2.2 Control & display on dew heater

If you control via the dew heater device (using mode button):

a) Control is via the button and LCD/OLED display. This is similar but less user friendly than PC-control.
b) Display is via the LCD/OLED display, similar to PC-control.

Use of these is described in (7).

3 BUILD OPTIONS - DETAILS

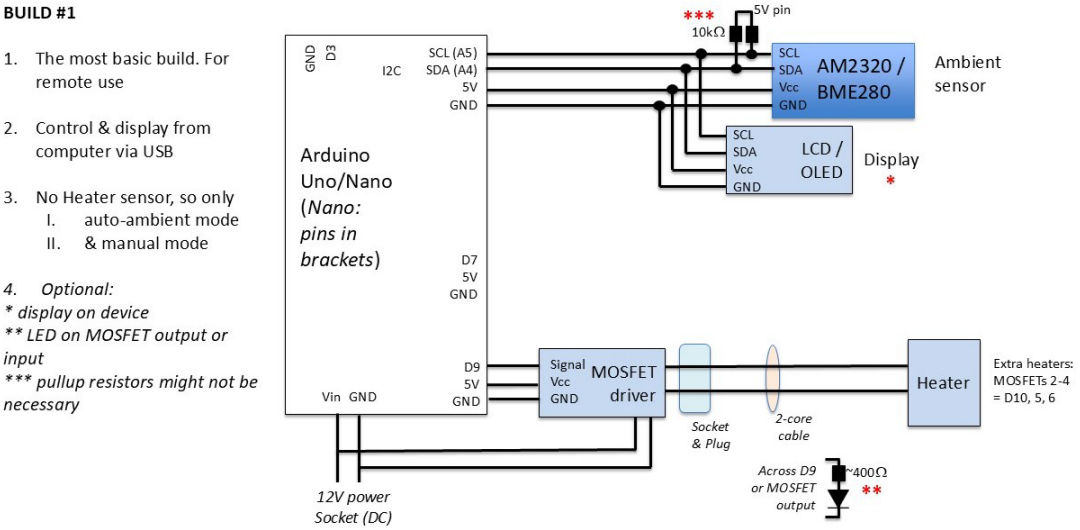
The choice of build depends upon selection of control (PC/button), display (yes/no) & heater mode (include auto-heater).

- Display & control
    - No display – all display & setting adjustments via a USB connected PC (or can run as a standalone – this is what I do)
    - Display on the dew-heater controller, plus push-on button to adjust settings.
- Heater mode types
    - Both auto-heater & auto-ambient modes (better control)
    - Just auto-ambient mode (simpler wiring).

3.1 BUILD #1: PC control, only auto-ambient mode

This is the simplest build with PC control/display and simple non-feedback heater control:

- Display & control:
  - Heating setting/reading are displayed & controlled via a PC (via USB).
  - If not connecting to a PC during use it can run as a standalone device after initial set up with a PC (section 2).
  - This makes it a good build for imaging if controlling the scope indoors/remotely.
- Automatic heating modes (in addition to manual/off modes):
  - Only the auto-ambient mode.
  - This provides a limited form of automatic control based on ambient temperature & humidity (dew point v ambient temperature). No feedback control.
  - Simpler wiring than the feedback auto-heater mode.
- Optional additions
  - * OLED/LCD display (4 in figure below). Displays heater data but cannot be used to set heater control.
  - ** LEDs on the heater outputs (6 in figure below): provides a simple display of the mode & heater power (# flashes & intensity).
  - *** pull-up resistors if have read failures.



Besides the Arduino, the main devices needed are:

- Ambient temperature/humidity sensor: BME280 or AM2320. BME280 is better.
- Heater driver: MOSFET modules (1x per channel)
- Optional additions: (6) LEDs on heater outputs (1x per channel), (4) LCD/OLED display

3.2 BUILD #2: PC control, full-feedback auto-heater control

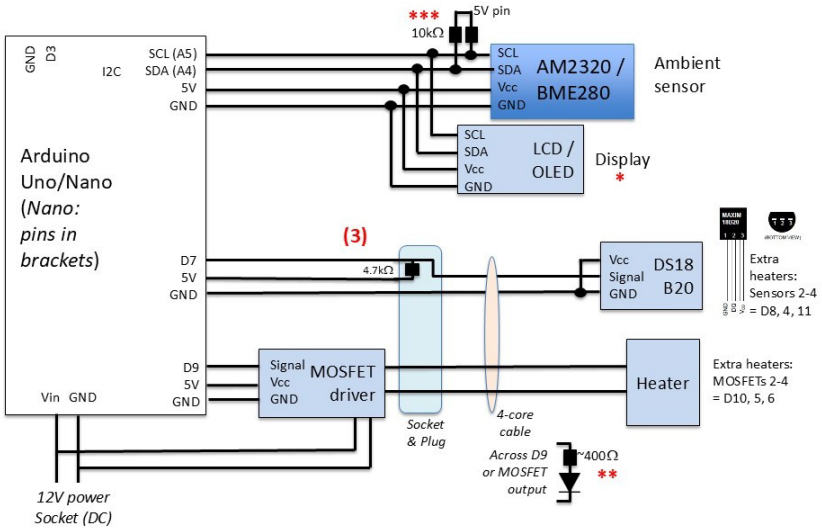
This has PC control/display and full-feedback heater control:

- Display & control: same as build #1.
- Automatic heating modes (in addition to manual/off modes): Same as build #1 but also has
  - Auto-heater mode provides a feedback control via a heater temperature sensor on the telescope
- Optional additions: same as build #1

BUILD #2

1. Best build for remote use
2. Control & display from computer via USB. *As in option #1.*
3. Has additional heater sensors so have
   I. auto-heater mode (the best control),
   II. auto-ambient & manual modes (as in option #1)
4. *Optional:*

** display on device*
*** LED on MOSFET output or input*
**** pullup resistors might not be necessary*



Besides the Arduino, the main devices needed are:

- Ambient temperature/humidity sensor: BME280 or AM2320. BME280 is better.
- Heater temperature sensor: DS18B20 (1x per channel)
- Heater driver: MOSFET modules (1x per channel)
- Optional additions: LEDs on heater outputs (1x per channel), LCD/OLED display

3.3 Build #3: PC control, only auto-ambient mode

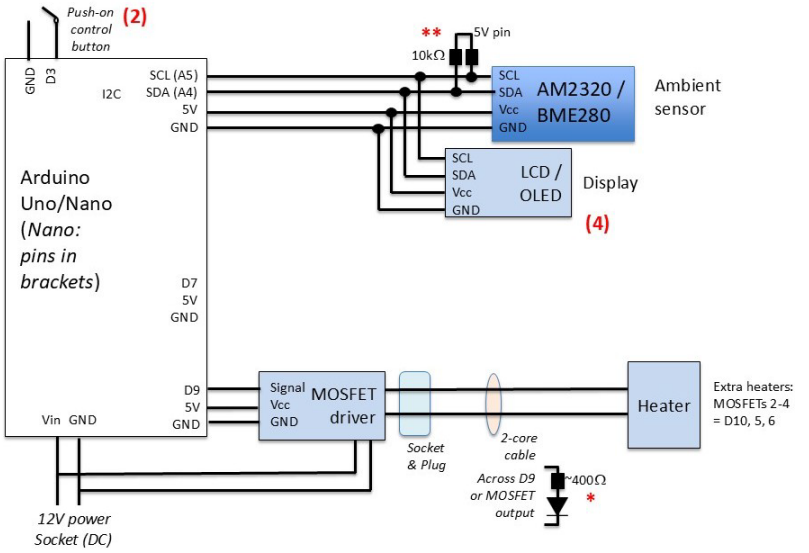
This has inbuilt display/control and simple non-feedback heater control:

- Display & control:
  - This build displays and sets parameters via a single push-button (no PC control) (2) on the device and an LCD or OLED display (4).
- Heating modes (in addition to manual/off modes): same as build #1.

BUILD #3

1. Simplest build for outdoor use at scope
2. control from push-on button & display
3. No Heater sensor, so only
   I. auto-ambient mode
   II. & manual mode
4. A display on device
5. *Optional:*

** LED on MOSFET output or input*
*** pullup resistors might not be necessary*



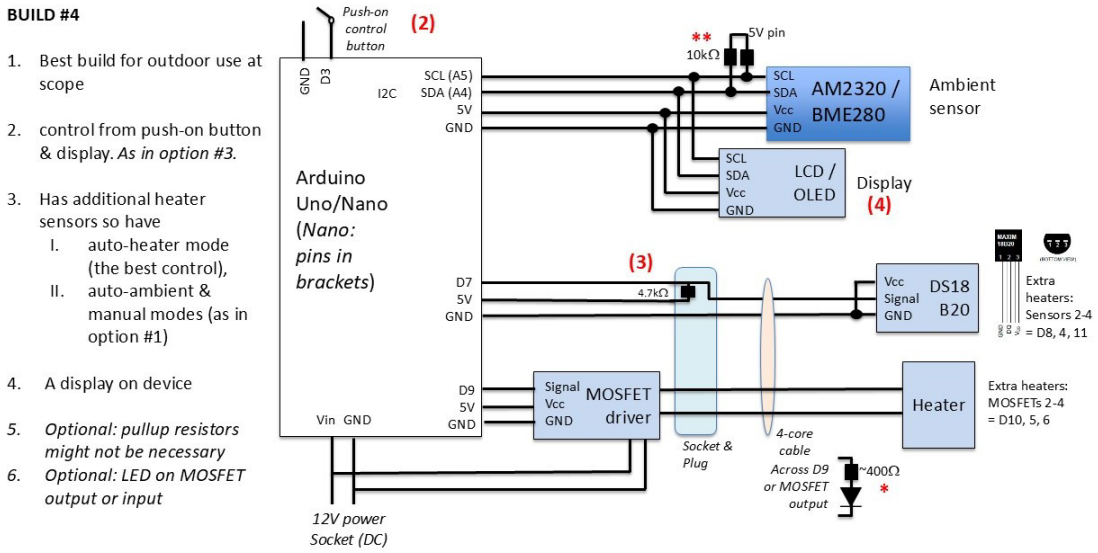
Besides the Arduino, the main devices needed are:

- Display: LCD (16*2, 20*4) or OLED (SSD1306 I2C)
- Push-button.
- Ambient temperature/humidity sensor: AM2320 or BME280. BME280 is better.
- Heater driver: MOSFET modules (1x per channel)
- Optional: LEDs on heater outputs (1x per channel).

3.4 Build #4: display, feedback auto control

This has inbuilt display/control and full-feedback heater control:

- Display & control: same as build #3.
- Heating modes (in addition to manual/off modes): same as build #2.



Besides the Arduino, the main devices needed are:

- Display: LCD (16*2, 20*4) or OLED (SSD1306 I2C)
- Push-button.
- Ambient temperature/humidity sensor: AM2320 or BME280. BME280 is better.
- Heater temperature sensor: DS18B20 (1x per channel)
- Heater driver: MOSFET modules (1x per channel)
- Optional addition: LEDs on heater outputs (1x per channel).

4 COMPONENTS

The 'dew heater controller' is built around the Arduino. It uses various sensors etc which are readily available from your local electronics shop, or on ebay, aliexpress etc. Some are cheaper/better than others. Some of these are optional – it all depends upon which build you go for:

1. Arduino Uno, or Nano.
2. MOSFET driver modules for the heater; 1 per channel.
3. Ambient temperature/humidity sensor.
4. A display, either an LCD (16x2, or 20x4) or an OLED (SSD1306). They must be I2C devices. OLEDs use less power. OPTIONAL.
5. A temperature sensor for each heater, DS18B20; 1 per channel. OPTIONAL.
6. Heater straps. You can buy these (e.g. cheap camera lens heaters on aliexpress) or make them using nichrome wire or resistors. I prefer to use nichrome. There's plenty of online resources to figure this out. I'll add instructions on how I do it.
7. Miscellaneous items (& there's probably a few others things I've forgotten):
    a. A box to put it all in.
    b. Insulated wire and/or dupont cables to wire up the devices. Heatshrink of varying sizes. Small screws, spacers, washers, bolts (M3, or even 2.5) – sometimes plastics ones are good
    c. 2/4-core wire for the heaters. Plus 2/4 pin plugs/sockets to connect heaters.
    d. A few resistors – for DS18B20s, LEDs & I2C devices pullups.
    e. LEDs – simple 5mm red (optional).
    f. Push-on momentary switch (optional).

g. Socket for the 12V DC input – connect to your 12V battery or power plug-pac.

More detail on the main components, 1-5, below (*I'm still working on 6*).

4.1 Components - Arduino

I've used both the Arduino Nano (Fig 3.1) & Uno.

- Nano with a terminal adaptor board:
  - simplest for beginner as easy to wire up, especially if you have to connect 2 devices to the I2C (display and ambient sensor). Little soldering for basic build.
  - This can be wired up to separate modules, e.g. red MOSFET modules below, plus BME280 sensor etc.
  - There is an equivalent terminal adaptor board for the Uno but it is much more expensive, so I haven't looked at it.

Fig 3.1 Parts – Arduino Nano

Arduino Nano

NB: I2C on Nano:
- SDA = A4; SCL = A5

& a terminal expansion board

- Uno with a bare expansion board:
  - Its not too difficult to place the new smaller MOSFET modules and connect them up (some soldering required). Plus header socket/plug connectors for I2C device, resistors for pullup & LEDs
  - You could also get MOSFETs & resistors (rather than modules) & make circuit on expansion board (this is my current build). But it's a bit messy.

PICS OF THIS – my build & unbuilt with new MOSFET drivers when I get them

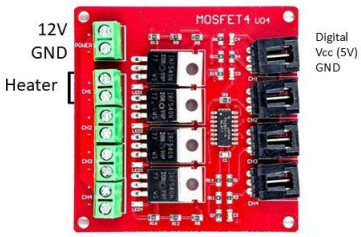
4.2 Components - MOSFET driver modules

Requires one driver per heater channel (Fig 3.2). You can use readily available modules, or you can make your own with a MOSFET and a few resistors (I currently have this on a bare prototype shield with an Arduino Uno).

Fig 3.2 Parts – MOSFET drivers for heaters

1-channel module - if have 1 (*or 2*) channels

4-channel module - if have 4 channels

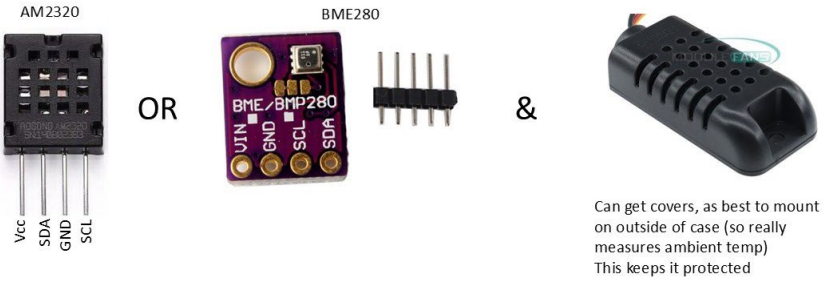


- The red ones above are good with terminal adaptor boards. These can be screwed onto box and wired up via their terminal blocks to the Nano terminal block board.
- The others are good with the Uno/bare expansion board. They can be soldered onto the expansion board using header pins & soldered connections to Arduino pins (on the expansion board).

4.3 Components - Ambient temperature/humidity sensor

One ambient sensor is required (Fig 3.3).

Fig 3.3 Parts – Ambient sensor



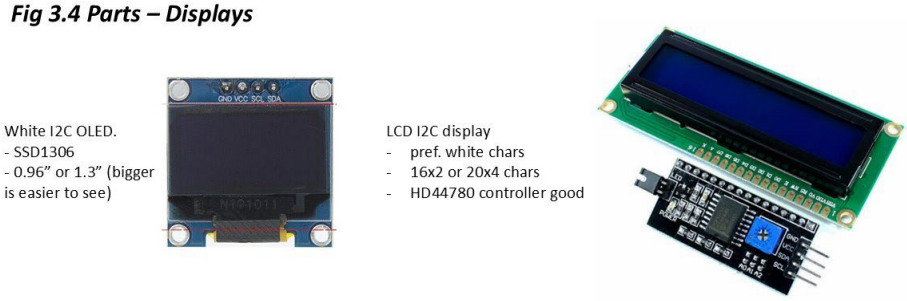
Note:

a. This can either an AM2320 or BME280. ***The BME280 is better***. It is more accurate and reliable. Some have reported issues with AM2320 read errors – but these have been fixed by adding pullup resistors or using an alternate Arduino library.
b. Be careful with wiring pins correctly. If your dew heater is not giving ambient temperature & humidity readings , the first cause is probably incorrect wiring (or an incorrect software address, or the software read issues with AM2320).
c. Placing it

4.4 Components - Display

There's a few options (Fig 3.4). All of these are good options, it's really personal preference. Best to get one with white, yellow or red lettering. You can put red plastic over screen with white lettering. This make it red which is good for night vision (if needed) and also insulates the it & other components inside the box from the weather.

Fig 3.4 Parts – Displays



Note:

a. If you are getting an LCD, make sure you get one with I2C !
b. The LCD is big compared to the smaller 0.96" OLEDs so is much easier to see. But its big – so a problem if you want a small build.
c. I like the bigger 1.3" OLEDs, and there's always some new great ones coming out. Just make sure its compatible with the SSD1306 driver, as I'm using an SSD1306Ascii driver (SSD1306Ascii version much smaller SSD1306 so no memory issues with the Nano/Uno). There are other OLED driver type – they are probably good but you would need to sort out drivers etc yourself.
d. Also care with wiring etc as in (b) & (c) for the ambient sensor.

4.5 Components - Heater temperature sensor (optional)

Need a DS18B20 for each heater channel. They have 3x pins but connection to the Arduino can be reduced to 2x pins by using the parasitic mode. I like to combine the cabling of this with the heater, so you need a 4-core cable. If you do so, make sure wire is enough to handle power requirements of your heater. Otherwise, you can have 2x separate 2-cire wires and strap them together

Add a pic of DS18B20, how to add resistor & adding heatshrink

4.6 Components - Heater straps

There are many options, such as:

a. commercial astro heater straps
b. cheap camera heater straps (e.g. on aliexpress), plus other options.
c. DIY it.

You just need to ensure that it will carry enough current (from a 12V supply) to provide enough power for your specific heater strap. Another things is that commercial heater straps have 2-core wire which provides power to the heater. If you want to use feedback auto-heater mode then you will have to

- Add an extra 2-core wire for the DS18B20 heater sensor, or
- Replace the 2-core wire with 4-core wire. 2 for power and 2 for heater sensor

The other option is to DIY heater straps. Its cheap and relatively simple. Everyone has their own way of doing it, so I will provide some very undetailed info (can provide more info later). I use nichrome wire – much simpler than resistors and its relatively easy to connect nichrome to wire (unlike some make out):

1. Get the correct length of nichrome (comes in 4 metre lots for $4-5 from well-known electronics stores) so that 12V makes its really warm but not ridiculous. The size/wattage really depends upon you scope and how bad your conditions get. Connect a heater strap – check that it heats up. Later on, you can run multiple wires in parallel if you feel that it needs more heat
2. You want the nichrome nice and straight (no kinks) – to do this pull it straight by making it red hot over your stove.
3. Cover the nichrome with 1.5mm heatshrink except for ~3-5 cm at each end – this will electrically isolate it. Again easier to do it over your stove.
4. Connect the nichrome to wire leads by twisting them together, soldering (it wont solder, it'll just hold it in place) then putting 2x layers of heatshrink over the top to hold the connection together. This works fine – mine has been good for a few years so far.
5. Lay the heatshrink-covered nichrome wire on aluminium tape - this will be on the scope side – holds it together & passes heat efficiently. You can get this from those well-known hardware stores.
6. On the other side a few layers of good gaffer tape then adhesive felt tape to keep in the heat. If I had a sewing machine I'd do it properly without all this sticky stuff.
7. Put a bit of Velcro on the ends to hold it together.

5 ASSEMBLY

5.1 Basic Steps for build

First, have a look through the build I have done and those I'm about to build (section 5.2). It's always a good idea to wire up the components before boxing them, this will help work out the bugs and potential issues with your build. Indeed, you could gradually build by adding a single device at time and testing just that device (e.g. ambient sensor, display). This can be done with the small arduino test codes included (section 6.1).

Probably the best approach is to build a bit, load the software and test it works.

1. Start with just the Uno or Nano. At these stages you can power it up by just the USB connection to your computer.
2. For the next few bits it may be easiest to run the windows app to check things work
3. Connect the BME280/AM2320. Check that the ambient temperature & humidity read OK.
4. Connect a MOSFET & heater. Run it in manual mode and check that the heater warms up (but see * below)

5. Then check that it works in ambient mode with the heater
6. If you have DS18B20 heater sensors, check that they are reading heater temperature.

Things to check

- * When putting it all together
    - make sure the 12V supply goes directly to the MOSFET modules - both +12V & GND.
    - And make sure GND from the 12V input also goes to the Arduino GND.
- BME280/AM2320 sensor. Make sure its on the outside of the box. Otherwise it will be measuring heat from the MOSFETs etc inside the box.
    - AM2320 & BME280: can be mounted inside a small chamber which has slits that allow outside contact. Its 4x wires can be run inside the box.
    - Best to solder the wires onto the sensor pins and cover them with fine heatshrink – these wire can be dupont cables with pins that could be connected to a socket inside the box (this makes it easy to put together and dismantle if you have problems).
    - I mount the sensor on the outside & put silicone over the hole to keep out moisture. Try not to have the sensor open AND facing upwards. You get condensation forming which is hard to budge and gives near prolonged near-100% humidity readings.

5.2 Builds I have done

There are so many ways this can be put together. Below are 3 different builds I've done. Eventually I will sort out how to design PCBs but I've been telling myself that for a few years … Here's a few example of my builds.

The first one has 2x channels & an LCD display. I attached this to the tripod. It has:

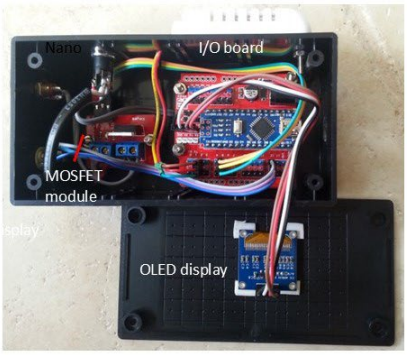
- Arduino nano with an expansion I/O board (not the terminal block board described above)
- 16x2 LCD display
- 2x MOSFET driver modules
- 5-pin DIN connectors for the 4-core cable to the 2x heaters & DS18B20 sensors
- 12V input from a cigarette plug

My build #1 – Arduino Nano expansion I/O board with AM2320, LCD display, 2x DS18B20-heater channels.

The next one was much smaller Nano build with only 1x channel & a 0.93" OLED display. This is really small/light and could be attached to a scope with Velcro strap.

- Also shown is a DIY nichrome heater strap, battery, & the DS18B20 heater sensor. NB: had two separate cables for the heater power and its DS18B20 sensor in this version, and a small rechargeable battery.

My build #2 – Arduino Nano with AM2320, OLED display, & 1x DS18B20/heater channel



Add the build I currently use

Add a build with Uno & expansion board with those new MOSFET DRIVERS [**BOTH INCOMPLETE**]

6 SOFTWARE INSTALLATION

6.1 Arduino

6.1.1 Install Arduino IDE & download Arduino app.

- Arduino IDE: there's instructions every on how to install & use the IDE. If you are undertaking this project, I imagine you have used the Arduino IDE before.
- Arduino DewHeater app:
  - Download from my github .
  - Put the whole folder DewHeaterController into the Arduino sketches folder

6.1.2 Install essential libraries

You'll need to upload the libraries used by this program. To do this: start Arduino & select Sketch/Include Library/Manage Libraries. The libraries needed are:

- LiquidCrystal_I2C – for the 16x2 LCD display (if used)
- DHT. For the DHT22
- Adafruit_sensor & Adafruit_AM2320 for the AM2320. NB: modification (see github)
- Adafruit_SSD1306 & Adafruit_GFX. – for the OLED display (if used)
- DallaTemperature. For the DS18B20
- OneWire. 1-wire for the DS18B20s [**ALL INCOMPLETE**]

Easiest way to check if you have correctly downloaded the libraries is to click verify the program. It will come up with errors on the #include statements for the libraries that need to be installed. When you figure out what library it is, open the library menu (on left), and type in its name (or part of it) – it should come up with the library, then install it.

6.1.3 Setup up the Arduino app settings & devices you are using

Next you will have to select what build/features you are using, such as the ambient sensor (2 types) or the display (a few types, or no display) you want to use. The basic ones are all at the start of config.h lines 1 - 25. There are instruction comments to help with this – read it first. These are:

- Number of heater channels: numChannels = between 1- 4.
- Ambient sensor type: BME280_ON or AM2320_ON for these sensors.

- Control type: PC_CONTROL or MODEBUTTON for the control being used
- Display type: OLED1306, LCD1602, DISPLAY_OFF (these are the most common – an OLED, LCD or no display). These are others that can be used. There are many other OLEDs – if there is something quite common, ask me & I'll look into including it.

On lines 26 – 36 there are some more options that can be altered (enter true or false), but these can be left as is, even if you aren't using them. They are:

- blankHeaterDuringRead = true. This turns off the heater(s) while reading their temps with the DS18B20 sensors. I found this essential if you have a long 4-core cable to the heater as the 2x heater wires create interference with the 2x DS18B20 sensor wires. A safe option = USE IT as the heaters are only off for a few secs.
- dewPointComplexCalc=true. You might as well, don't even know why I still include the simple calc option.
- outputFlashMode=true. This flashes the heater output 'n' times at the end of each 20sec cycle, to indicate the mode (1=off, 2=manual, 3=auto-ambient, 4=auto=heater). This is useful if you don't have a display on the device and aren't using the windows app. If you don't want this you can leave out adding the LED/resistor across each output.

There are other things you can set up (lines 37- 42):

- The PID controller settings used for auto-heater mode. These are described later (section 8.1).

Now you can verify the app and check it works, then upload. Before uploading it to your arduino you need to tell it what Arduino you have. So under Tools – select Board/Arduino Nano; Processor/ATmega238 & the COM Port (this will be different on a Mac). Sometimes if it doesn't upload check the processor type and even try the old bootloader under tools

6.1.4 Check that it loads & works

- Before you install it into a case, its best to put it all together in stages to check each bit works. I have sketches that test each bit individually if you have big problems. Try it without a heater strap connect. Blow in the AM2320 – should see temp and humidity go up and power output should increase. Then connect a heater strap – check that it heats up.

- Put it all together in a jiffy box. It has 3 modes, selected by the mode button on digital pin 3:
    - Automatic mode: heater output set by difference between scope temperature and ambient dew point. For heaters without sensors – the heater power can be set manually
    - Manual mode: heater mode manually set to 0 – 100% of maximum power
    - Off mode: heaters off, only displays ambient temp/humidity
    - Reset: set back to automatic mode/0% manual power.

6.1.5 Settings in Arduino IDE

There are some settings that can only be set in the Arduino app. So these must be setup before downloading to the Arduino. These are all in the file config.h. Settings:

- Line 12. The ambient sensor: choose AM2320_ON or BME280_ON.
- Line 19. The control type: PC_CONTROL or MODEBUTTON. Control from the PC or directly on the device.
- Line 24: display type: OLED1306, LCD1602, DISPLAY_OFF. Latter if no display.
- Lines 30 - 33. See description in config.h, best to leave as is.

- Line 41. These are for the PID feedback controller in auto-heater mode. Don't touch until you read & understand section 8.1.

6.2 Windows APP

The windows app can be downloaded from my google drive, here. Its in the folder <Windows_App>. Just place the folder in your programs folder, or anywhere.

7 Using the Dew Heater Controller

If you got here assume that you have built the device & it have loaded software successfully. Before describing control via the windows app on button/display on the device, it should be noted that

- All the basic settings are stored on the Arduino EEPROM memory. When the device is powered up at starts with the previous settings.
- Whenever you change these settings they are saved on the EEPROM.
- These setting include the mode (global), manual power, auto-ambient max power and threshold, auto-heater target temperature, temperature cutoff.

7.1 Windows APP

The window app allows remote control and display of the dew heater setting and readings (Figure 7.1).

Figure 7.1. PC Control app

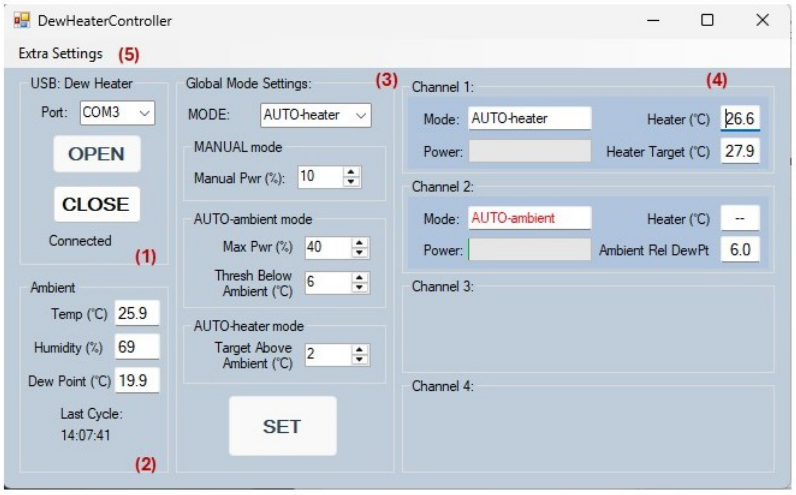
(1) USB control interface

(2) Ambient conditions *(display only)*

(3) Global Mode Settings

(4) Channel readings & settings *(display only)*

(5) Extra settings.



The app contains a number of function boxes. These boxes are only enabled when they can be used, e.g. group boxes (2) – (5) are not enabled until the USB port is opened in group box (1), auto-heater mode settings only enabled when global mode is set to auto-heater. These are the group boxes.

7.1.1 USB group box

This controls and displays the current USB connection to a PC. To use.

- Select the appropriate USB port which is connected to the dew heater
- Click OPEN to use and click CLOSE when finished. When it connects it will say 'connected' and all other group boxes will become enabled.

Sometimes opening can take a while if the dew heater is busy. If doesn't open after 10 sec or so, just close the app window and restart it.

7.1.2 Ambient readings group box

This displays the ambient temperature, humidity & dew point (degrees Celsius).

- It will also display the time at which the last cycle commenced.
- The Arduino dew heater has a cycle time of 20 sec. this is the time between updates of all readings (ambient & Heater Channels).

7.1.3 Global Mode Settings group box

This controls and displays the global mode settings. These global settings will apply to all of the heater channels, except when certain things aren't connected or there are read errors for individual heater channels (see section 7.1.4). Please note:

- Whenever you change any of these settings the SET button will turn red.
- You have to click this button to apply the changes to the Arduino dew heater (changes will not be applied until you do this). Once done, it will turn black again – this can take a while as it doesn't happen until the current cycle (20sec duration) finishes.

This group box includes:

- MODE: drop-down menu to select OFF, MANUAL, AUTO-ambient, OR AUTO-heater modes. When you apply this MODE one or more of the mode boxes will be enabled:
    - MANUAL mode: only this box is enabled
    - AUTO-ambient mode: this & MANAUAL modes boxes enabled.
    - AUTO-heater mode: this, MANUAL & AUTO-ambient boxes enabled.
- MANUAL mode: drop-down menu to select the power level in increments of 10% up to a maximum of 100%.
    - NB: The actual level of heat depends upon the actual heater strap characteristics (*this is the case for all modes*).
- AUTO-ambient mode: this mode uses the ambient conditions to determine heater power (see section 2.1.1). There are 2x drop-down menus:
    - Max Pwr (%): this is the maximum power that can be achieved. I have included this as heater dew straps can generate a lot of heat, so for prolonged use you really don't want that. You can determine what this level should be by running the dew heater in awful 100% humidity and sorting out what power level is required to stop dew formation in manual mode.
    - Threshold temperature below ambient (C): this is the temperature below ambient at which the heater starts to turn ON.
    - I find with mine, 20-40% of maximum power and a threshold of 6C below ambient is a good starting point to try – but sort this out yourself (especially the max power).
- AUTO-heater mode: this mode uses the heater temperature & ambient conditions to determine heater power (see section 2.1.2). There is 1x drop-down menu:
    - Target above dew point: this the set-point differential above dew point to use to set the set-point target temperature. But it also takes into account how close the dew point is to the ambient temperature
        - If humidity is low, dew point way below ambient temperature, so set-point target temperature = ambient temperature. In this case, the heater will rarely turn ON.
        - If humidity is high, dew point approaches ambient temperature, so set-point target temperature = dew point + set-point (which will be above ambient temperature). In this case, the heater will turn ON & OFF to maintain that temperature.
    - I find a value of +2-4C works well.

7.1.4 Heater Channel readings group box

This displays the data for each heater channel. The number of channels shown depends on the number of channel on you Arduino controller. The items displayed vary with each mode:

Mode	Heater Temp	Power Bar	Varies with mode
OFF	Value(C) or '—'	Range = 0 – 100%	Heater Power (%)
MANUAL	Value(C) or '—'	"	Heater Power (%)
AUTO-ambient	Value(C) or '—'	"	Ambient – Dew Point (C)
AUTO-heater	Value(C) or '—'	"	Heater Target Temp (C)

NB: the mode of any heater channel can differ to the global mode; if sensors are not used or if they have a read error:

a) AUTO-ambient: defaults to manual mode if ambient sensor no-read.
b) AUTO-heater mode: defaults to auto-ambient if heater sensor no-read, and then to manual as in (a).

Also

c) Any mode can default to OFF mode if heater goes above the cutoff temperature; this requires a DS18B20 heater sensor.

7.1.5 Extra Settings

There are 2 extra settings here which are not used very often:

- Temperature cutoff (C): temperature at which heater will cutoff. You want the heater too hot. This can happen with any automatic (ambient/heater) mode but you have to have a temperature sensor connected to that heater channel for it to work
- Save data file: this will save all of the data readings during each cycle. It will be stored in the user 'public' and will be named DewHeaterLog_*date*.csv. This can be viewed as an organised excel file, or opened as a text file.

7.2 Local display/control on device

7.2.1 Displaying readings & settings

Over a 20 sec cycle, ambient and heater channel data are displayed @ 10 sec intervals, similar to PC readings (section 7.1.4).

Cycle display – readings & settings

Displayed items & example values shown in black. At ~ 10sec intervals these 2 things are shown

Ambient - readings

Temp	Humidity	Dew Point
val C	val %	val C

Heater

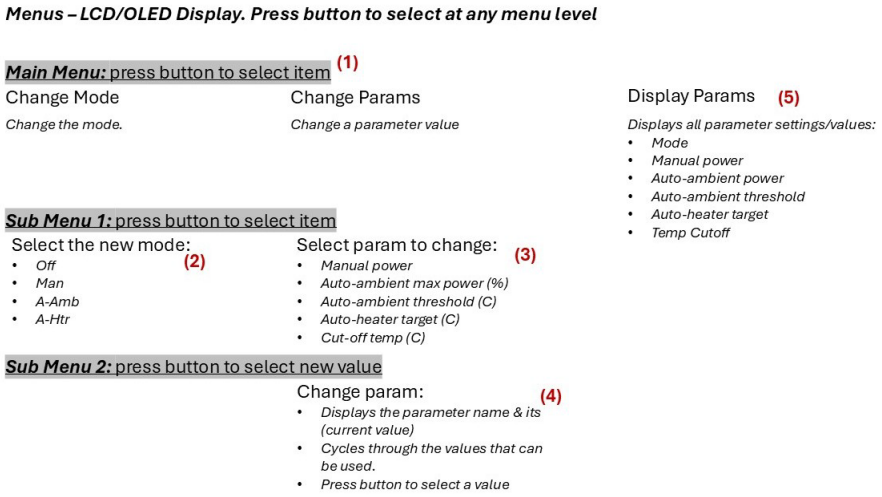
Chan	*Setting*	*Power*	*Mode*		*Setting display*
1	--	0%	Off	*Off*	--
2	--	10%	Man	*Manual*	--
3	+4C	11%	A-Amb	*Auto-Ambient*	*Ambient - DewPoint*
4	16C	5%	A-Htr	*Auto-Heater*	*Heater Temp*

7.2.2 Changing mode, parameters etc

To change the mode, change any parameters associated with the modes, or just display the current mode & all parameters press (& then release) the control button.

On the first button press the main menu will be displayed (1 in fig below). Press the button again to select what you want to do:

a. Change the mode
b. Change a parameter/setting for any mode
c. Display the current mode & all parameters/settings



This will come up with the following menus

(a) change the mode: the display will cycle through the modes (2 in fig above). Press the button again to select the mode you want. The mode will be selected and the display will then return to displaying readings & settings as in section 7.2.1.

(b) change a parameter/setting: the display will cycle through the parameters (3 in fig above). Press the button again to select the mode you want. This will come up with another display of:

a. the parameter/setting to change & its current value (in brackets) (4 in fig above).
b. the display will continuously cycle through the range of allowed values. Press the button again to select the mode you want (don't worry if you miss the value you want the first time around as it will continually re-cycle until a button is pressed). The mode selected will be displayed, saved to EEPROM (if it has changed). The display will then return to displaying readings & settings as in 7.2.1.

(c) Display parameters/settings: this will display the current mode & all the parameter settings (5 in fig avove). The display will then return to displaying readings & settings as in section 7.2.1.

8 Other things

8.1 Auto-heater mode: PID controller

This mode uses feedback control to set the heater temperature for each heater channel. The aim is to have the heater temperature at a set point (this is the value set by the PC or display) above dew point. NB: important to distinguish between the set-point target & temperature

- *Set-point target*: this is the value selected in the windows app or on the display. For example, a value of +3C.
- *Set-point temperature*: the set-point target is then used to determine the actual set-point temperature for each heater. So this might be a temp = dew point + 3C (if dew point is close to ambient temp) or ambient temp (if dew point way below ambient). See section 2.1.2 for more details.

8.1.1 PID parameter settings - basics

This feedback control could be achieved by a number of means.

- ***Proportional (P) control***. This is the simplest. It uses the difference the heater temperature and the set-point point to determine the heater power, the error (multiplied by a gain factor). This is what I originally used but it was limited as the heater temperature either (i) never reaches the set-point (there's always some steady-state error), or (ii) oscillates madly about the set-point; depending on the gain factor used.
- ***PID (proportional, integral, derivative) control***. This is more complex but works really nicely and there's a simple Arduino library for it. There's a simple description here. It uses 3 factors for feedback correct of the error (the difference between the heater temperature and your chosen set-point temperature). These are:
    - Kp (P-term). Just the error as stated above
    - Ki (I-term). Integral of the error.
    - Kd (D-term). Rate of change of the error (D-term).

These K values are the 'gain' of each of the factors. With this dew heater, I have opted for just the P&I terms because

- the controller does not have to work very fast. So the D-term isn't crucial.
- A really important this is that the controller needs to reduce the long-error (i.e. keep the difference between the heater temp and set-point temp near 0). So the I-term is crucial.

I have found that

- making the P & I-terms equal (with the D-term 0) produces reasonable control with the heater straps I have tested, i.e. Kp=Ki, Kd=0.
- Also, using Kp & Ki = 10 is a pretty good starting point for 3 different heater straps I have used. It's not particularly fast but doesn't have any overshoot with my hater straps.

So you can either

i. stick with these values, Kp, Ki, Kd = 10, 10, 0. Or,
ii. determine the optimal Kp & Ki values for your system. See next sections for more details

8.1.2 What are the optimal PID parameter settings

The optimal Kp, Ki (& Kd) terms are dependent upon several factors. The main ones are:

- The thermal characteristics of your heater straps. Given you'll have a few different heater straps you might end up with different Kp/Ki values.
- the cycle interval. So leave it at 20sec. These are determined by the characteristics of your heaters (and other stuff). They are setup so that the system has fast response, but not too fast that the system oscillates.

I've set up base Kp, Ki, Kd values of 10, 10, 0. To adjust the PID setting for your individual heater straps I've added a gain factor, **gainPID[]** term in config.h:

- gain 0.5: Kp, Ki, Kd values to 5, 5, 0.
- gain 1: Kp, Ki, Kd values to 10, 10, 0.
- gain 2: Kp, Ki, Kd values to 20, 20, 0.
- gain 4: Kp, Ki, Kd values to 40, 40, 0.
- gain 6: Kp, Ki, Kd values to 60, 60, 0.
- gain 8: Kp, Ki, Kd values to 80, 80, 0.
- gain 10: Kp, Ki, Kd values to 100, 100, 0.

Maybe there are heater straps out there that won't fit these values – if so, they can be changed in the Arduino app.

8.1.3 Determining the optimal PID parameter settings – using base Kp=10, Ki=10, Kd=0

To determine the optimal gainPID for each channel, use

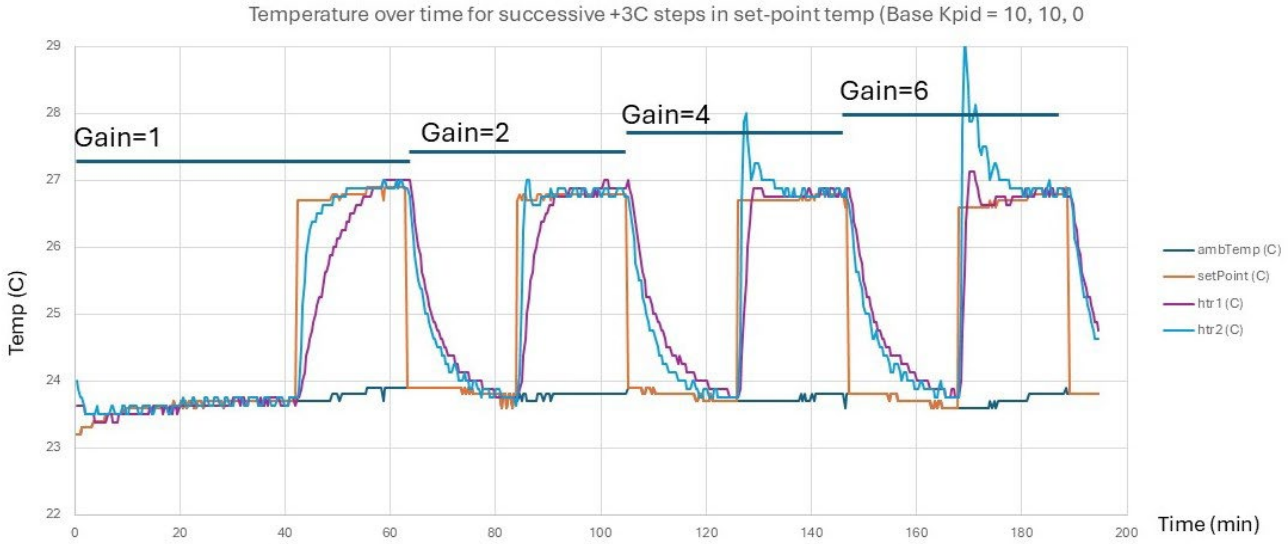
- the Arduino app <*PID_test_stepGains.ino*>. Its in the folder <*DewHeaterController_testingPID*>.
- the windows app <*PID_simple_testing.exe*>. It will collect the Arduino data & save it as a .csv file in your *user/public* folder. If you open the .csv file (in excel), it can be plotted & you can easily determine the optimal PID gain for each heater channel.

But remember, this takes a fair while to do (the run in the figure below took over 3 hours). If you go this far, then plotting the .csv file in excel, you'll get something like below. It shows:

- the ambient temp, set-point temp and heater temp during successive steps to the set-point (+3C) from ambient temperature.
- There are two different heaters connected.
- The PID gain increases with each step. I have annotated these gain values

You can see that

- The heater temp increases towards the set-point with each step-up & then dreceases back to ambient at the end of each step.
- At the end of each step: the rate at which heater temp returns to ambient doesn't change with the gain, as the heaters automatically turn off when using PI control. This is a rate limiting effect of each heater (at least when there are temperature drops).
- At the start of each step: the rate at which the heater temp increases more rapidly as the gain is increased. Also at higher gains there's an overshoot in the responses plus some oscillations.



To get the optimal gain, pick the heater response that reaches the set-point temperature with each step quickly, but not so fast that it also oscillates. So,

- heater 1 (purple) has an optimal gain=2. Even though gain=4 is faster with little overshoot it's best to be conservative.
- heater 2 (light blue) has an optimal gain=1. Again, this is a conservative choice.

To set this up on the Arduino, set these values in the config.h file line 41
int gainPID[4] {2, 1, 1, 1}; // gain factor for PID Kp/i/d for channels 1, 2, 3, 4

- *make sure you assign a value to all channels even if you aren't using them (stick to defaults = 1)*

But really given the rate limiting effect at the end of each step, I could just use the same gain=1 for both heaters. This is because you don't need a fast response (ambient temp and humidity don't change that quickly), just want long term accuracy.

8.1.4 Determining the optimal PID parameter settings – your own Kp, Ki, Kd

I won't give detailed instructions for this, but you can change these settings in other ways

(a) Change the relative values of Kp & Ki. For instance, you might like to try a higher Ki, e.g. Kp = 20, Ki = 40, Kd = 0, etc.
(b) Use a non-zero Kd value if you want. I have done this and have found that a Kd ~ 1/20th of Kp is good, e.g. Kp, Ki, Kd = 20, 20, 1.

But if try either of these (esp. Kd!=0), test it as above! And don't go too high with Kd otherwise it can

- oscillate wildly around the set-point
- introduce a large temperature increase/decrease when starting up the Arduino
- diminish its ability to reduce long-term errors.

Either way, the best approach is to follow the simple approach here.

I. Set Kp. With Ki, Kd = 0, gradually increase Kp until it starts to over shoot on the rising phase. Use the Kp values just below this point. NB: it won't quite reach the set-point. If you have multiple heaters that differ, you'll probably find they require different Kp values.
II. Set Ki. With your Kp value & Kd=0, gradually increase Ki until it starts to reach the set-point at steady-state.
III. Set Kd.

Later, I will upload Arduino apps to do this. They are very similar to <*PID_test_stepGains.ino*>.

Please note, if you change Ki relative to Kp (and/or add Kd), then this new ratio will apply to all gainPID settings. So, proceed with caution.