

Weather station

This is an arduino UNO home weather station with e-ink display.

The hardware used for this project is:

- Waveshare 4.3 e-ink display
- Arduino Uno
- Adafruit BME280 combined temperature, humidity and atmospheric pressure
- DS3231 based hardware clock

Functionality

- time
- date
- temperature
- station atmospheric pressure
- sea level atmospheric pressure
- humidity
- short-time local weather forecast
- pressure tendency

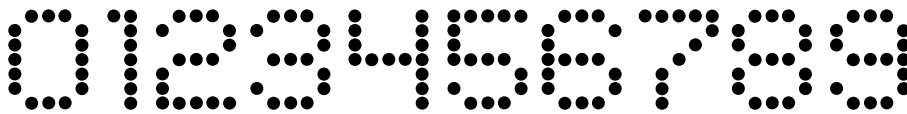
The main challenges of this projects were:

- using a font, different than the one build-in in the display
- creating images without using too much memory
- choosing and implementing the forecast algorithm

Font

The display has build-in fonts. The problems are, they are raster, non-scalable, have only Latin and Chinese symbols and have only 3 sizes. Even worse, typographic-wise the three sizes are different. I wanted a good-looking interface and mixture of serif and sans fonts makes this impossible. The solution was to define a font in the code as a 3-D array, where every symbol consists of a matrix of 5x7 pixels. At first I used one byte per pixel, but that almost topped up the available memory. So I decided to use 1 byte for every column and to access the pixel info with bit-wise operators. In this way I loose only one bit for column.

The display API library has primitives for drawing filled circle and I use this to draw the symbols. The numbers look like this:



The function I have created expects as a parameter the diameter of the circle and by it the diameter of the circle and space between the circles is controlled. As a result I got a variable-size font, not dependent of the fixed-sized fonts embedded in the screen.

Weather icons

The display supports uploading images to its internal memory, but they are again raster images and thus not scalable. So I decided to create functions in the code for drawing the icons, based on the available graphic functions in the display API and to be able to control the size of the image in the same way as the font characters. At the end I made images for sun, cloud and rain, that can be combined in different ways. I used the primitives for filled circle and rectangle for most of the image parts. Unfortunately the API function for drawing a line does not have parameter for the thickness. So I had to implement my own line drawing function based on the Bresenham algorithm, but instead of pixels I draw filled circles. The size of the circles is the thickness of the line.

Forecast algorithm

I quickly realize that the manufacturers of the existing home weather station does not publish any information about the algorithm they use for choosing what weather prediction icon to show. The only algorithm for that purpose with a description that I found was the Zambretti algorithm, named after its creators Negretti and Zambra and published in the year 1915 (Long live the copyright culture!). The input parameters of the algorithm are season, sea-level pressure, barometric tendency, hemisphere and wind direction. The algorithm is empirical and created for the northern hemisphere and more specifically Great Britain. Analysis made on the algorithm conclude, that wind direction has little effect to the final result, which is a good thing

because wind direction sensor in urban environment is hard to use. I get the temperature, humidity and station pressure from the sensor, but the relative pressure should be derived from the station pressure. There are many formulas for that, but the one I use is the following:

$$P_0 = P \left(1 - \frac{0.0065 * h}{T + 0.0065 * h + 273.15} \right)^{-5.257}$$

where

P_0 is the relative sea-level pressure

P is the station pressure in hPa

h is the altitude in meters

T is the temperature in Celsius.

The problem is how to get the altitude. The sensor I use returns the approximate altitude, but it is based on the barometric pressure and it is affected by it. It is like chicken-and-egg problem. In order to get the exact relative pressure I need the station pressure and the altitude, but in order to get the altitude I need the exact pressure. After some experiments it turned out that if I use the sensor for getting the altitude the calculated sea-level pressure stays almost the same – this is because when the pressure changes the altitude changes too. To solve this problem I came up with the following two ideas:

- to gather data for the barometric pressure for prolonged period of time and use some average value
- to use pressure, entered by the user

Both approaches have some limitations – the first one needs long time to gather the data and some memory for it, and the second needs the user to know the altitude and the device to have some input interface for it. Unfortunately this is the price that have to be paid to get an adequate short-time forecast.

Determining the tendency of barometric pressure turned out to be a challenge, because the materials only say “rising”, “falling” and “stable”, but these terms are not defined. In other sources, not connected to the Zambretti algorithm the tendency is defined as the change of pressure with one (or two or three at most) units in hectopascals for the period of one to three hours. Armed with this knowledge I made the following: I save the sea-level pressure every 10 minutes in an array. When the array is filled I calculate the tendency as the difference between the average of the first three elements and the last three. The array is used as a queue, where the 11th value goes to the 10th position and the first one is discarded and so on.

The Zambretti algorithm returns a number, that have to be compared to a table with the text description of the forecast. Some computer implementations of the algorithm try to derive a formula that gives the number of the row in the description table based on the Zambretti number. I decided to use the approach, used by the developers of the Javascript implementation, where number intervals are used, defined with if-else if operators.

The mappings of the text descriptions to the images is given below:

Falling pressure	
Settled fine	Sun
Fine weather	Sun
Fine becoming less settled	Sun, cloud
Fairly fine showery later	Sun, cloud
Showery becoming more unsettled	cloud
Unsettled, rain later	Sun, cloud, rain
Rain at times, worse later	Sun, cloud, rain
Rain at times, becoming very unsettled	cloud, rain
Very unsettled, rain	cloud, rain

Settled pressure	
Settled fine	Sun
Fine weather	Sun
Fine possibly showers	Sun, cloud
Fairly fine showery later	Sun, cloud
Showery bright intervals	Sun, cloud, rain
Changeable, some rain	Sun, cloud, rain
Unsettled, rain at times	Cloud, rain
Very unsettled, rain	Cloud, rain
Very unsettled, rain	Cloud, rain
Stormy, much rain	Cloud, rain

Raising pressure	
Settled fine	Sun
Fine weather	Sun
Becoming fine	Sun, Cloud
fairly fine, improving	Sun, Cloud
fairly fine, possibly showers	Cloud

showery early, improving	Cloud, rain
Changeable mending	Cloud
rather unsettled Clearing later	Cloud
unsettled, probably improving	Cloud
unsettled, short fine intervals	Cloud
Very unsettled, finer at times	Cloud, rain
Stormy, possibly improving	Cloud, rain
Stormy, much rain	Cloud, rain

Plans for future development

- Calculating moon phases
- Showing the day of the week
- Adding buttons for setup
- Adding energy-independent memory for saving the collected data
- Adding external sensor for temperature and humidity
- Adding an image for storm
- Connecting via WiFi to mobile devices for setup and control
- Showing events – birthdays, appointments, official holidays, defined in a mobile app

Download:

<https://github.com/fandonov/weatherstation>

Връзки

<http://drkfs.net/zambretti.htm>

<http://keisan.casio.com/exec/system/1224575267#!>