Contents

1 Maintenance

NAS times are online for until around 14/9/2018.

To change IP adress, change it in the Arduino code two times (`myip` en `gw`), if needed also change the NAS ip adress:

```
ether.hisip[0]=192;
ether.hisip[1]=168;
ether.hisip[2]=178;
ether.hisip[3]=29;
```

and the dns (directly under it).

It may be needed to find the right dns for the NAS and the gw (gateway) address again, by doing a normal dhcp setup like

```
ether.dhcpSetup();
ether.printIp("IP:␣␣", ether.myip);
ether.printIp("GW:␣␣", ether.gwip);
ether.printIp("DNS:␣", ether.dnsip);
ether.printIp("SRV:␣", ether.hisip);
```

Do not change the dns for the NTP server, `dns[] = {195,121,1,34}`, otherwise the NTP dns lookup fails.

Also change it in the Android app two times (`ipString` and `host`), and in the desktop app.

The current ip address is `http://192.168.8.42`.

2 Setup of the project

2.1 Hardware used

We used an Arduino Nano ATmega328, bought at rs-components.

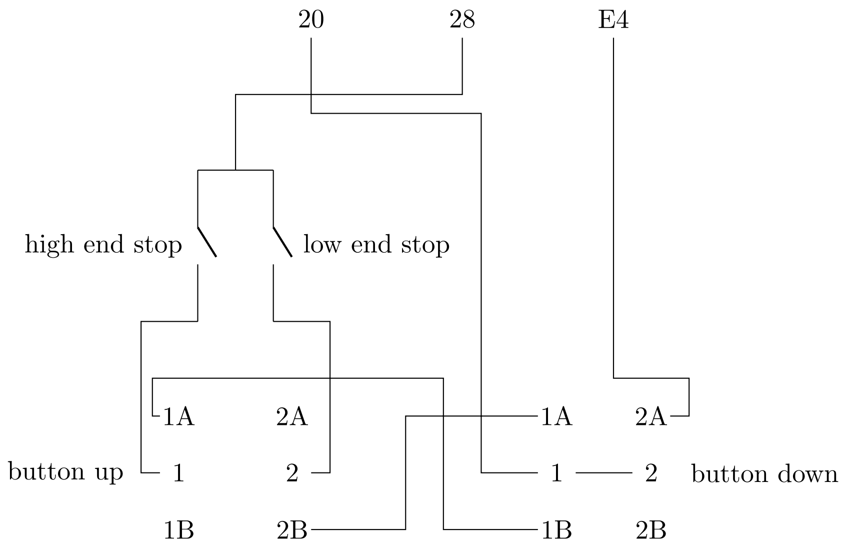
We used a ENC28j60 Ethernet shield, the version specifically for the nano. This seemed easier than a wifi shield because of this reason, and with a wifi shield it seemed we needed extra components and a circuit, and we didn't really understand it.

2.1.1 Components list

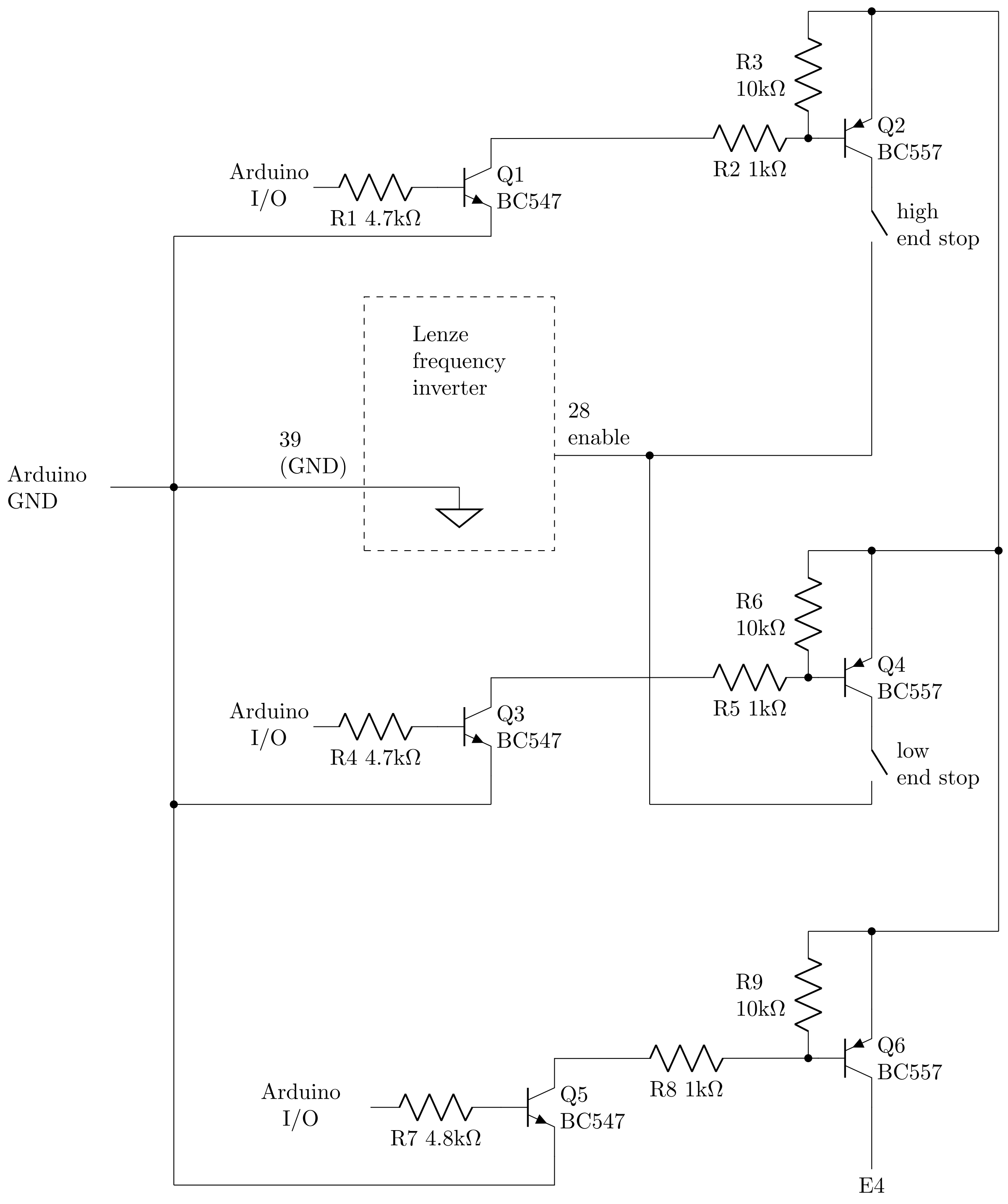
- BC547 NPN transistor
- BC557 PNP transistor
- 4.7k Ω resistor (yellow - violet - red - gold)
- 1k Ω resistor (brown - black - black - brown - brown - red)
- 10k Ω resistor (brown - black - orange - gold)

2.1.2 Button circuit

We implemented the begin- and endstops in the circuit for the buttons that move the panels up and down. When a button is NOT pushed, 1 and 1B, and 2 and 2B of that button are connected. When a button is pushed, connections are made between 1 and 1A, and 2 and 2A of that button.


20
28
E4
high end stop
low end stop
1A
2A
1A
2A
button up
1
2
1
2
button down
1B
2B
1B
2B

2.1.3 Arduino circuit



circuit	terminal (connection to)	to meter cupboard	Arduino
Low end stop	no	white/green	5
Ground	4	red	GND
E4	5	blue	4
High end stop	6	green	3
To potmeter:			
blue (GND)	7		
yellow/green (signal)	8	black	A7
brown (+5V)	9	yellow	5V

2.1.4 Potentiometer

The yellow-green wire from the potentiometer to the main control box is the signal wire, on connection 8 in that box. With the Arduino we measured 74 (analog in so scale 0-1024) in the highest position and 360 in the lowest position, with connection 9 to ground on the Arduino.

2.1.5 Motor control

We bought BC547 NPN transistors to control the 12V/20mA (docs, page 4-11) or 14V/40mA (measured) current of the EVF8202-E frequency inverter, and also base resistors otherwise the Arduino needs to give too much current to the transistor, and the transistor will be slow to turn off because of 'base charge storage'.

2.2 Software used

We decided on the PlatformIO IDE, (which uses python 2.7 and Clang for autocompletion) because it is a lot better than the standard Arduino IDE, and also seemed better than the Stino plugin for Sublime Text 3. A plugin for CLion also looked good but we didn't get that to work. PlatformIO worked fine but not always, so as a backup you can always use the Arduino IDE, but you need to make sure you have all the libraries in your `C:\Users\username\Documents\Arduino\libraries` folder.

3 Arduino Code

3.1 Internet/Ethernet connection

To connect the Arduino and the Ethernet shield to the internet, we used the EtherCard library. Because the ENC28j60 uses a different default CS pin (10 instead of 8), we had to add that in the code when making the connection. This is done by changing

```
if (ether.begin(sizeof Ethernet::buffer, mymac) == 0)
```

(with no pin specified, so the default pin is used) to

```
if (ether.begin(sizeof Ethernet::buffer, mymac, 10) == 0)
```

Note the third argument `10` added after `mymac`.

To get the current time using NTP, we adapted example code from the Arduino forum.

3.2 Solar Panel control

The calculation to find the goal value of the potmeter is within one voltage point accurate, which is because of the rounding to an integer voltage at the end. The calculation itself as implemented on the Arduino is 0 to 0.01 off, because of the rounding of the long value.

The offset can be seen with this Mathematica command

```
Plot[N[360 + ((x - 5)*100/(50 - 5))*(74 - 360)/100] -
N[360 + Floor[((x - 5)/(50 - 5))*100*(74 - 360)]/100], {x, 10,
10.01}]
```

which illustrates the difference between the exact solution and the Arduino code without rounding to the integer `expectedVoltage`, which was implemented as

```
float fraction = ( ( (float) ( (degrees - DEGREES_LOWEND) ) ) / (float) (
    DEGREES_HIGHEND - DEGREES_LOWEND) );
int expectedVoltage = POTMETER_LOWEND +
( (long) ( fraction*100 * (POTMETER_HIGHEND - POTMETER_LOWEND) ) ) / 100 ;
```

The idea is to keep setting the right pins high until the difference between the current voltage and the expected voltage is zero. If the potmeter happens to skip the value, no problem occurs because the Arduino keeps no memory of that and will try to reach the right value again. Tests showed that with sufficient sampling of the input the value may in a rare situation be skipped once but not more.

3.3 Storage of angles

Because the Arduino program space could by far not store all the unix times and angles for like a year, we decided to use a webserver on the Synology NAS as storage space. Every time the Arduino runs out of angles, it detects that either in the `loop()` or in `solarPanelAuto()`, and then requests new angles at http://192.168.2.7 with `requestNewTable()`. After sending such a request, the program needs to wait for a response before continuing, otherwise it continues with old angles. The 'waiting' is implemented with a global flag variable `responseReceived`, which is set true in the callback after the angle and date arrays are updated.

Currently the arrays are of size 10, and therefore the little PHP script on the NAS gives the next ten angles and dates when called. The number ten is chosen because of RAM limitations, if you want to change it, change the `tableLength` global variable, the number in the declarations of the arrays, the number of times the PHP script gives back, the `tableSize` which indicates how many bytes the NAS response will need, and is used in the ethernet buffer and when parsing. To calculate that size, given x angles, you'd do something like $11x$ for the dates plus $4x$ for the angles and a little more plus about 140 bytes/characters from the http header would be around 300 bytes for 10 angles. But tests point out almost 400 bytes are needed here, so be sure to test it well.

In the NAS, you can find the page with the file browser under `DiskStation/web/index.php`. By the way, on high performace, exporting the angles for ten times a day for two years, like `exportPeriod[DateObject[{2016, 9, 7}], DateObject[{2018, 9, 15}], 10]` took only ten minutes (Lenovo laptop on high performance).

A possible improvement is obviously to save the angles locally on for example an SD card, which would decrease dependencies on other systems.

3.4 Communication with the Android app

Communication goes by http requests, in the form of http://192.168.2.106/?panel=up. Currently implemented are `panel=up,panel=down,panel=stop,panel=auto,degrees=x,update`. To find what these are for and what they return, have a look at the code (communication file). When you request http://192.168.2.106/ you get a homepage with the current status of the solar panels (currently just the internal time of the Arduino). It is also possible to request to set the panel to x degrees by sending http://192.168.2.106/?degrees=xx where xx is an integer which needs to consist of two digits. To get an update of the current solar panel status, which currently only gives back the current angle and whether the panels are on auto or manual, request http://192.168.2.106/?update. You'll get back some html, with `[angle] ["auto" or "manual"]`.

On every action sent, the Arduino will send back an http response starting with `HTTP/1.0 200 OK`, and including something like `Panels going up.`.

3.5 Safety measures

In order to avoid relying on the hardware end stops, a few safety measures were introduced:

- A soft bound was introduced for the potmeter constants and degrees constants. We hope this catches varying potmeter bounds: sometimes the value of the low end may be a few points lower and then the code would try endlessly to get the solar panels below their hardware end stop.
- In the `solarPanelDown` and `solarPanelUp` methods, as close to the solar panel control as possible, we put an extra check if the reading of the analog pin does not go out of bounds. These bounds therefore include the soft bounds. This check intentionally does not rely on `readPotMeter()` (which samples readings for better accuracy), which reduces accuracy near the soft end stops but increases safety. Accuracy inbetween should not be influenced.
- The Arduino now has a timeout on moving the panels up or down, which means that when you do not send another up/down request within the timeout of x seconds the panels will stop automatically. The app and desktop version now try to send a request each $\frac{x}{2}$ seconds, but subject to change. Especially because of the in practice noticed bad connections this is a very practical alternative to keeping an open connection which would be even better but is a bit complicated with an Arduino.

4 Android App

4.1 User Manual

To move the solar panels up (or down), press the up (or down) button and hold it until the desired position/angle is reached. The current degree on the top right and the picture update while the button is held, so you know when the panels are at the desired position.

To set the solar panels in auto mode, press the auto checkbox.

To set the solar panels at a certain degree, move the slider until the x at the button which says "`SET ANGLE AT` $x°$" is the desired degree for the angle. Then press the button to set the angle of the panels at that degree. While the panels are moving, the degree on the top right and the picture update to be the current one. May you change your mind about the angle while the panels are moving, simply drag the slider to the desired angle. It's not needed to press the button again, when the panels are still moving.

To update the —TextView— containing the current angle, simply press the number. This will also update the position of the picture. This will also check the checkbox if the panels are in auto mode, and uncheck the checkbox if the panels are not in auto mode (if needed).

4.2 Communication with the Arduino

To communicate with the Arduino, the Android app sends http requests to the Arduino. For all the requests accepted by the Arduino, check section 3.4. Below a list containing what requests are sent when something is clicked can be found.

- **Up and down button.** `http://192.168.2.106/?panel=up` is sent at the moment the button is pressed, so the Arduino knows we want the panels to move up. After that, the request `http://192.168.2.106/?update` is sent with a set interval. This interval depends on the speed of the solar panels, and is about as much as it takes the panels to move 1 degree. This request is sent so often to be able to keep the degree (on the top right of the screen) and the picture up to date with the actual position of the panels. At the moment the button is released the request `http://192.168.2.10/?panel=stop` is sent to the Arduino, so it knows we want the panels to not move anymore. No `?update` requests are sent anymore.

- **Auto checkbox.** `http://192.168.2.106/?panel=auto` is sent when the checkbox is being checked, `http://192.168.2.106/?panel=manual` is sent when the panels go out of auto mode. Which is either when the user unchecks the checkbox, or when the user clicks the up/down button, or when the user clicks the button to set the panels at a certain angle.
- **Current degree text.** `http://192.168.2.106/?update` to update the current angle in the text, and the position of the picture to the actual position of the solar panels.
- **Set angle button.** This takes the current position of the slider/seekbar (`seekbar.getProgress()`) and uses that to send a request `http://192.168.2.106/?degrees=xx` to the Arduino so it knows at what angle to set the solar panels. Until the panels have reached the set position, the `?update` request is sent with the same interval as for the up and down buttons. The app knows when the panels have reached the set position by comparing the current position of the panels and the position of the slider before each request. If these are equal, it doesn't send an `?update` request anymore. This is why it's not needed to press the `SET` button again when changing the value of the slider while the panels are still moving, if that movement is caused by this button.

5 Calculations

5.1 Finding the angle between the sun and the line perpendicular to the solar panels

To find the angle α between the sun and the line perpendicular to the solar panel, we determined both lines in spherical coordinates (with the same distance to the origin) and then calculated the angle between the two.

For the sun this would be

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \cos\gamma_s \sin(\frac{\pi}{2} - \theta_s) \\ \sin\gamma_s \sin(\frac{\pi}{2} - \theta_s) \\ \cos(\frac{\pi}{2} - \theta_s) \end{pmatrix}$$

where γ_s is the azimuth, and θ_s the altitude of the sun.

For the line perpendicular to the solar panels this would be

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \cos\gamma_p \sin(\frac{\pi}{2} - \theta_p) \\ \sin\gamma_p \sin(\frac{\pi}{2} - \theta_p) \\ \cos(\frac{\pi}{2} - \theta_p) \end{pmatrix}$$

where γ_p is the azimuth (direction of the solar panels in respect to the South), and θ_p the altitude of the solar panels.

Then, to find the angle α between these two lines, we take the arccos of the dot product. This gives us

$$\alpha = \arccos(\cos(\gamma_p - \gamma_s)\cos\theta_s \sin\theta_p + \sin\theta_s \cos\theta_p)\,.$$

5.2 Mathematica calculations

The Mathematica package (imported with `<< SolArduino``, be sure to place it in your `%AppData%\Mathematica\Applications` folder) can calculate the optimal angle for a given day. To do that, it calculates for each angle between 0 and 90 the total of the insolation (power received by the sun) at each half hour of that day. Then it finds the angle for which that value is maximal. To find the insolation at a given hour, the function `angle` calculates the misalignment with the sun using the formula from the previous subsection, and then calculates the insolation using a formula from www.powerfromthesun.net, where parameters for urban haze compared a lot better with real life values (17/8) than clear day parameters.

It can therefore make graphs for optimal angles for a month, and averaging the values for a month, also for a year, and lots more as seen in the demonstration notebook. When plotting real data, for example days like 13/5, 19/7 and 17/8 are all cloudless days with the solar panels at around 25 degrees.

It is important to note that the functions `angle`, `directPower` and more do not take the hour of the day as input, but the index of the `sunPositions` table which contains the azimuth and altitude of the sun over the day. Therefore, *before you call functions which take an index as parameter you need to make sure you have* `sunPositions` *initialised* at the right day, done by calling `calculatesunPos[DateObject[{2016,7,18}]]` with whatever day you want in the table.