Technical University of Denmark

Report - An IoT solution for the electronics for a N2O Cooler - V 2.0

Course: Special course - Internet of things for N2O Cooler System

Date: February 21, 2018

Authors: Fadi Bunni - s154072

Advisors: Ying Yan

Table of Content

1 Introduction **1**

2 Overview **2**

3 Requirements **3**

4 State machine **3**

5 Data-sheet **3**

6 Interfaces **4**

7 Implementation and Tests **5**
7.1 Thermocouples 5
7.2 Pressure transducer 8
7.3 Load cell 9
7.4 Solenoid valve 9
7.5 Node-red all sensors and controllable units 11
7.6 State machine 13
7.7 GUI for data and control 19
7.8 Extending the WiFi capabilities 21

8 Discussion **25**

9 Conclusion **25**

10 Appendix A - Octo board thermocouples python code **26**

11 Appendix B - Pressure transduceres python code **27**

12 Appendix C - Load cell python code **27**

List of Figures

1 Overview of the Test bench for the N2O Cooler electronics 2
2 Node-red simple flow diagram 5
3 Node-red debug console for a simple flow diagram 5
4 Node-red flow-diagram of the Thermocouples 6
5 Node-red debug console for the thermocouples 7
6 Node-red flow-diagram of the pressure transducers 8
7 Node-red debug console for the pressure transducers 8
8 Node-red flow-diagram of the weight sensor 9
9 Node-red debug console for the weight sensor 9
10 Node-red flow-diagram of the solenoid valve 10
11 Node-red debug console for the solenoid valve 10
12 Node-red flow-diagram for everything 11
13 Node-red flow-diagram for everything 12
14 Node-red flow-diagram for the state machine 13
15 Node-red flow-diagram for the state machine zoomed in 13
16 Node-red setting up the switch function for the state machine 15
17 Node-red setting up the state machine node 16
18 Node-red debug for state machine 17
19 Node-red State machine switch for solenoid valve 18
20 Node-red GUI for weight sensor 19
21 Node-red GUI for solenoid valve 19
22 Node-red GUI for thermocouples 20
23 Node-red State GUI for pressure transducer 20
24 Soldering a IPX U.FL connector on the Raspberry pi 3. to extend it with an external WiFi antenna 21
25 WiFi wire adapter is connected to the IPX U.FL connector 21
26 WiFi without modification: show WiFi spots and their strength 22
27 WiFi without modification: show WiFi spots and their strength, extended information 23
28 WiFi with modification: No onboard antenna nor external 23
29 WiFi with modification: with external antenna 24

1 Introduction

Now a days technology is becoming an increasingly integrated part of many household products, especially products that can be connected and controlled through the internet. This process of *internetfication* of simple household products is giving the consumer more freedom and the seller a huge amount of data that can be analyzed and used to increase product satisfaction. A more mainstream name for this process is called Internet of Things or short IoT.

In this project description we will use the idea behind IoT and implement it as a solution to control and monitor a N2O cooler system. The controlling and monitoring is not only going to happen from a touch screen but also through a WiFi connection to a client host. Furthermore the cooler will have sensors that will collect data about itself. The data will be used to determine, through a state machine, what the cooler should do about its given state. Furthermore for safety reasons the data will continuously be analyzed by a human, to determine if the cooler needs to abort its given state, thus a manual system is also needed.

2 Overview

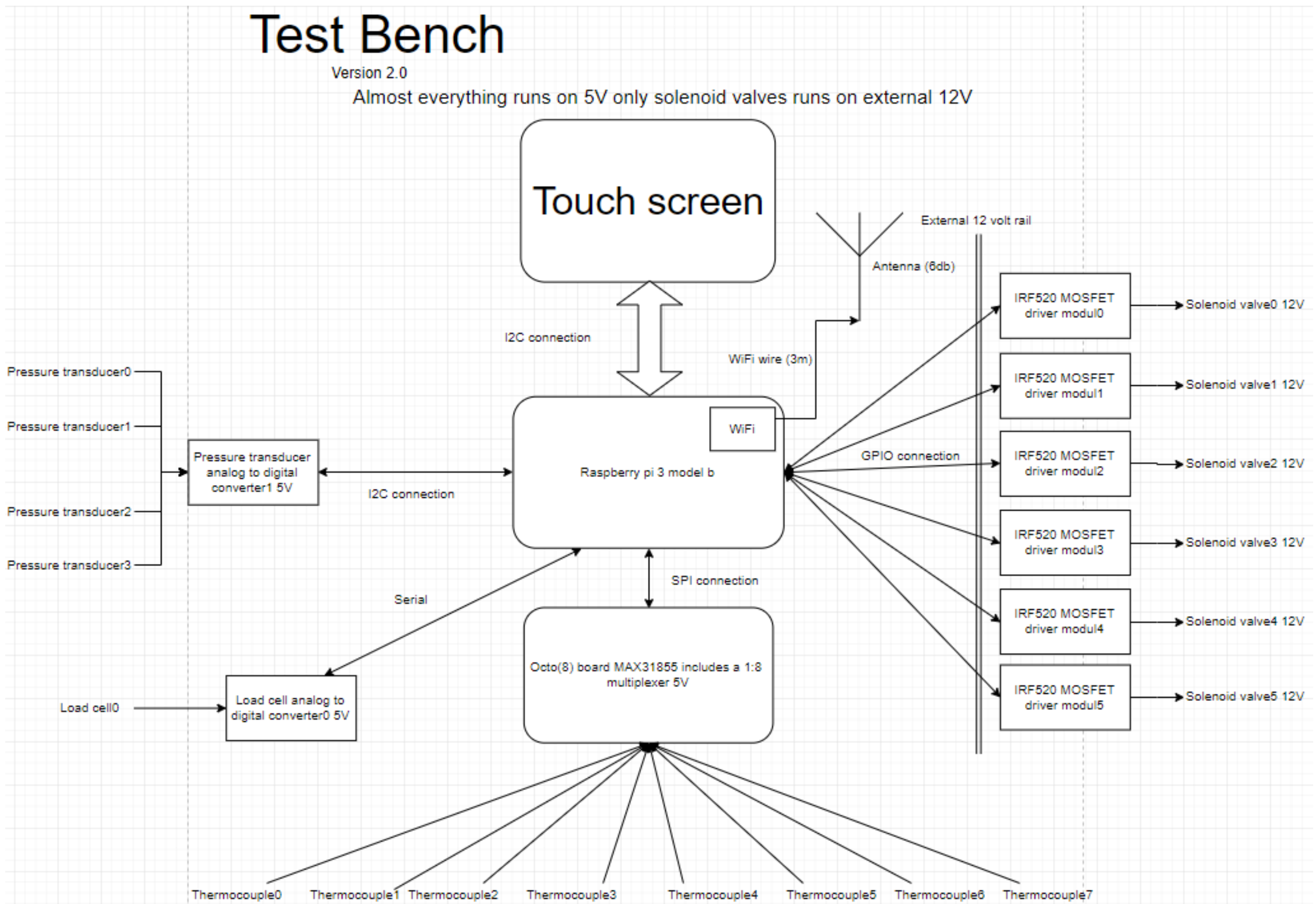

Figure 1: Overview of the Test bench for the N2O Cooler electronics

Figure 1. shows the overview of the test bench for the whole electronics system. There are 4 main parts the Raspberry Pi 3 model b, which is the microprocessor, the touch screen used for the GUI and controlling the system, The WiFi extension antenna, and the rest is the sensors and solenoid valves. There are 3 different sensors; in this case 1 load cell, 8 thermocouples and 4 pressure transducers. All parts run on a 5 volt rail beside the solenoid valves they run on an external 12 volt rail. The Antenna is extend by a 3 meter WiFi wire which is attached to a connector this is going to be manually soldered onto the raspberry pi board. Each sensor type has a specific chip controller, that is used to convert the signal from analog to digital, and send the data through either a SPI, I2C or serial connection. For the Thermocouples there is also a 1:8 multiplexer that reads the signal from all 8 thermocouples and send it to the on board microchip on the Octo (8) board. The touch screen has its own board that is going to help the Raspberry pi to render the images on the 7” screen, the connection between the two board is done by I2C.

3 Requirements

There are several requirements set for this N2O cooler project, we are only focusing on the electronics part. The goal is to try to fulfill as many of the requirements as possible

1. The N2O cooler needs to be controlled automatically and manually.
2. All the data read by the sensors and all the data produced by the automation has to be shown on a GUI or on a simple console.
3. The N2O cooler has to be able to use its sensors to monetize and control itself, by receiving the data from the sensors so that it can analyze them and act on its given state.
4. To control itself; a state machine is required.
5. To control it manually; it needs to have running GUI that shows the necessary information.
6. The console and GUI has to be visualized on an on-board touch display or on a website set up by the on-board controller.
7. All the sensors has to be controlled by just one board(microprocessor)
8. The electronics needs to be able to log all the data on a flash drive

4 State machine

For automation of the N2O cooler we need to look into designing a state machine, which job is basically to figure out what the N2O cooler should do about itself when it is on a given state and when a given sensor has triggered it to change to a new state. See section ”Tests” where the State machine is explained further. The state machine is of course only a software solution.

Before we can test we need to know what the requirements for the state machine are. We since we don’t know for sure how the N2O cooler is going to be we need to build a simple hypothetical example, and try to implement it. So the requirements are:

1. If the weight reaches a specific amount activate a given solenoid valve.
2. If a given thermocouple reaches to a specific temperature activate a given solenoid valve.
3. If a given pressure transducer reaches a specific pressure activate a given solenoid valve.

The above example is a simple state machine that will be tried to be implemented in this project

5 Data-sheet

For this project we will be using:

- A Raspberry pi 3 model b(microprocessor)
- Raspberry pi touch screen set
- Octo (8) MAX31855, used for converting analog to digital signal for 8 thermocouples
- ADS1015 used for converting analog signal to digital for the pressure transducers

- BSS138 Level shifter used for reducing the output power from sensors to protect the raspberry pi 3.
- HX711 Analog to digital converter (24bit) for load cell.
- Sensors: pressure transducers, thermocouples and load cell.
- Micro-SD 8gb
- MOSFET DRIVER for converting analog to digital data for the solenoid valves.
- WiFi antenna
- WiFi extension wire
- Solenoid valves used for controlling the flow of liquid nitrous oxide in the pipes.
- External batteries and power for 5V and 12V.

6 Interfaces

The primary interfaces will be:

- USB: The USB connection will be used for connecting to the Raspberry Pi and used for the data flow between the touch screen and raspberry pi. Also it can be used to power up 5V rails
- I2C: The I2C protocol will be used for connecting all the pressure transducers
- SPI The SPI protocol will be used for connecting Octo (8) board which has the MAX31855 chip that controls the thermocouples.
- Wifi: will be used to connect to the raspberry pi 3 and to a client computer where the data from the sensors will be sent to.
- Serial connection for controlling GPIO units like solenoid valve.

7 Implementation and Tests

For the testing we are going to look in to the individual sensors and controllable units, furthermore the GUI will be demonstrated. All necessary code will can be seen on the appendices.

We are going to use Node-Red for coding the individual sensors and controllable units. Node-Red is a flow-based programming language based on Node.js. Below on figure 2 and 3 an example of how the program works, can be seen. Two nodes are connected to each other via a wire, when the left node(injection) is pressed it will send a string containing "Hello World". The left node(debug) receives the string and prints it out on the debug console, seen on figure 3. Thus one can create many nodes, connect them to each other and send data forth and back between them. If you have the right node for a given sensor you can just connect it to a debug node and activate it, thus you can eliminate programming the sensor you just need to set up the right parameters for the node. Sometimes one will need to write his own function that filters, splits, and sends data to other nodes, so programming is still necessary in order to utilize the full capabilities of Node-Red.

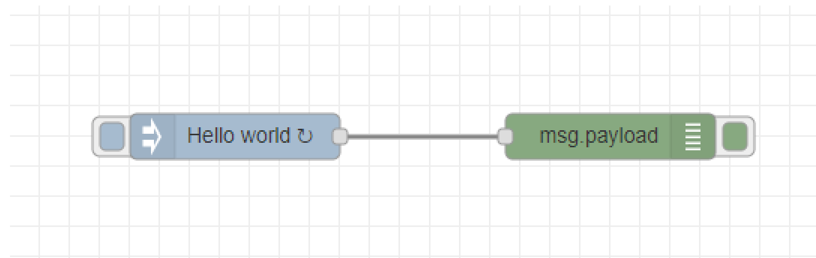

Figure 2: Node-red simple flow diagram

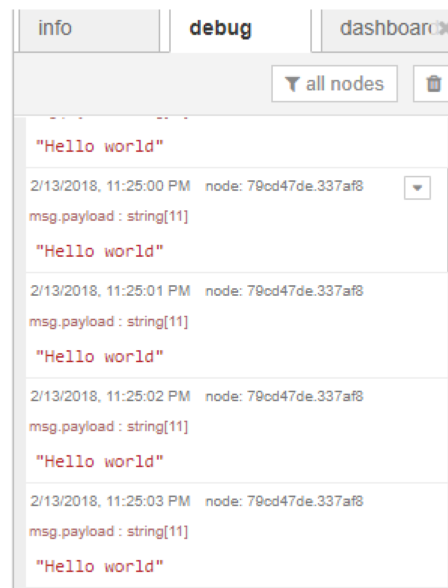

Figure 3: Node-red debug console for a simple flow diagram

7.1 Thermocouples

Below in figure 4 shows the flow diagram of the thermocouple. It looks to be rather complex but it is actually quite easy. Before we take a deeper look into it we will look into the python code for

the thermocouples for a split second, see *Appendix A - Octo board thermocouples python code.* At line 23 to 26 we have setup 3 GPIO these are used for the on board multiplexer on the Octo 8 board. Line 31 to 33 is the out GPIOs, where the variable "tc" fines which GPIO should be set to, in other words which thermocouple of the 8 should be read. So it cycles through them all and updates time is 125ms for each thermocouples, which is 125*8=1s meaning each thermocouples updates 1 time each seconds. Note that for all the sensors the information is printed as jsons object, it makes it easier to transfer data forth and back from python to Node-Red and then manipulate it.

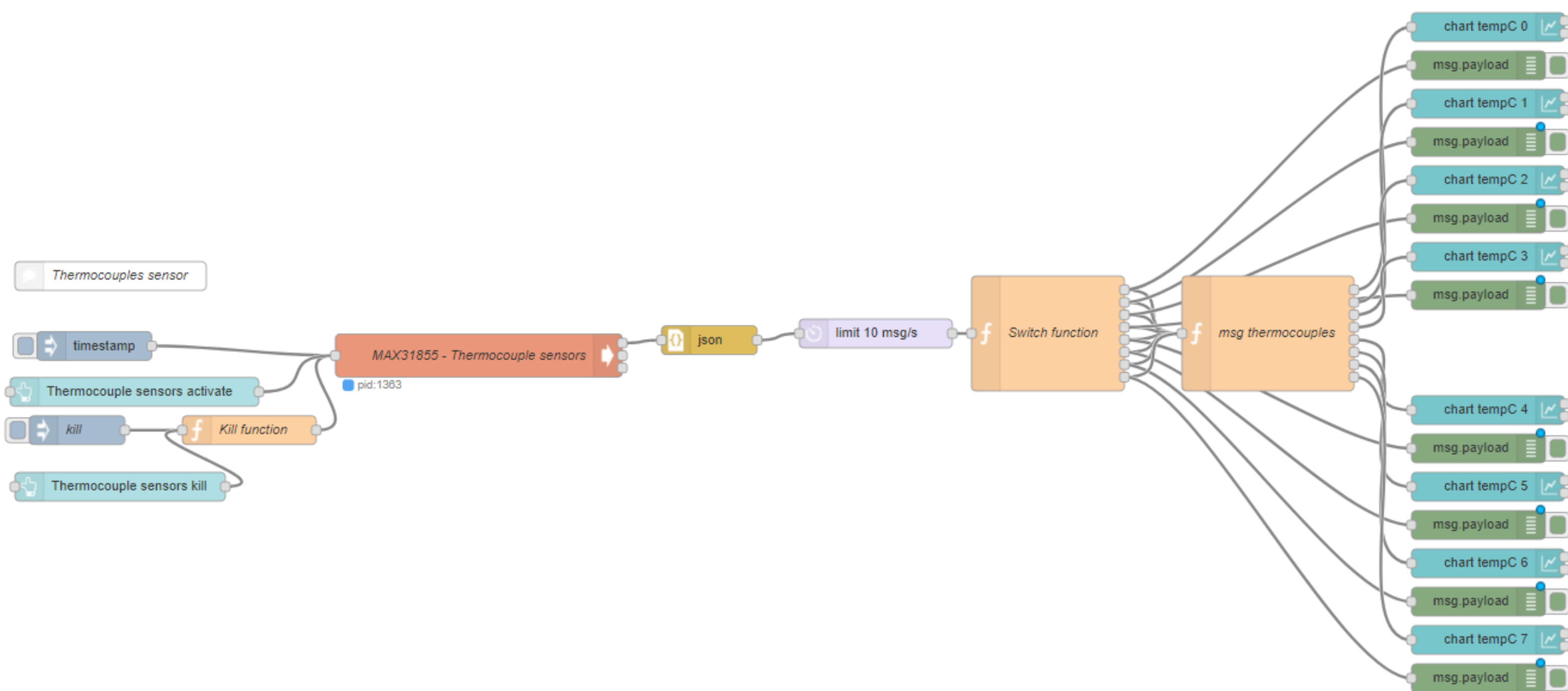

Figure 4: Node-red flow-diagram of the Thermocouples

So figure 4 has multiple nodes running. The first two important on the left, are the *timestamp* and *kill*. The first one activates and the other one kills the session(the pid number - see figure 4). The kill injector sends a payload to the kill function that in return changes the message payload to a kill payload that then get passed to the exec(red one) node. The exec node is used to run external scripts, in our case we are running a the python script shown in Appendix A. When the print function is executed inside the python script(see line 37 in Appendix A) the exec node listen for the print commands and return what ever is printed. As explained above we are using json objects, so we need to turn the jason string comming from the python script to a json object, that is what the next json node(yellow) is doing. We then have a limit message node, which job is to only allow 10 messages from the print function to be sent further into to flow diagram, so it is like a filter. The 10 message that get passed through each 10 seconds reaches to two function nodes which has been programmed manually, essentially their job is to filter the data so each thermocouple can be visualized individually on the debug node and on the GUI - The GUI is going to be explained later for all sensors, the GUI nodes are the baby blue color, the two on the left and 8 on the right.

Figure 5 shows the debug node output. Here we see the temperatures for each of the 8 ther-

mocouples. ”tc” represents the number of thermocouples starts from 0 to 7(see also figure 1), ”tempC” and ”internalC” are two different ways of measuring temperature. On the chapter about GUI we are going to show how the temperature increases and decreases it is easier to see it on the GUI.

Figure 5: Node-red debug console for the thermocouples

7.2 Pressure transducer

on figure 6 it can be seen the flow-diagram for the pressure transducer, note how the look of it resembles the thermocouples. basically what has been done, is that the same nodes has been reused and changed a bit to fit for the pressure transducers instead. That is the good thing about Node-Red easy reusability of node - which is actually code blocks. So what is happening on figure 6 is the same as for figure 4, instead of having 8 output debug nodes we only have 3 this time, since we only have 3 pressure transducers in use.

Figure 6: Node-red flow-diagram of the pressure transducers

Below on figure 7, it is seen what the 3 debug nodes outputs for the pressure transducers, approximately 1 bar, which is actually close to be valid. "i" is the id of the pressure transducers.

Figure 7: Node-red debug console for the pressure transducers

7.3 Load cell

Figure 8 shows the flow-diagram for the Load cell sensor, again the same flow-diagram as those above(now there is reasons to fall in love with Node-Red), its just the amount of debug nodes that has changed and the script to be executed in the exec node(red one).

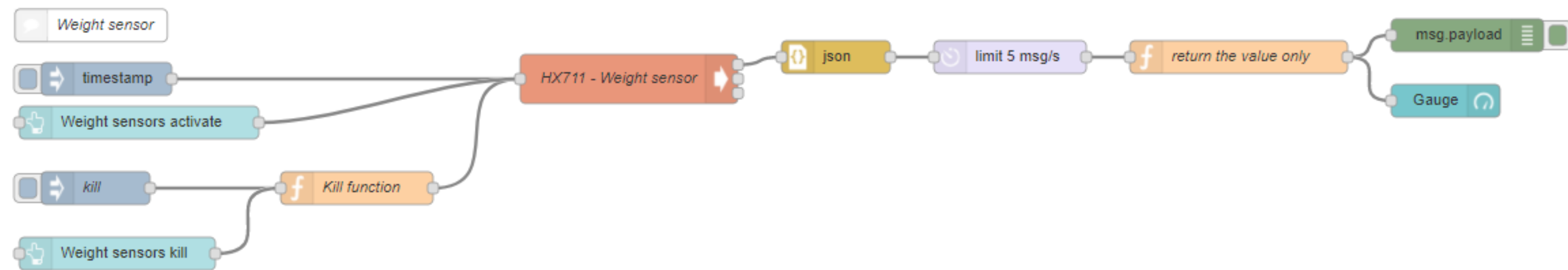

Figure 8: Node-red flow-diagram of the weight sensor

Figure 9 shows the output of the load cell sensor, right now it shows 0 since there is no load on it sensor. Look for the GUI chapter to see the sensors in action.

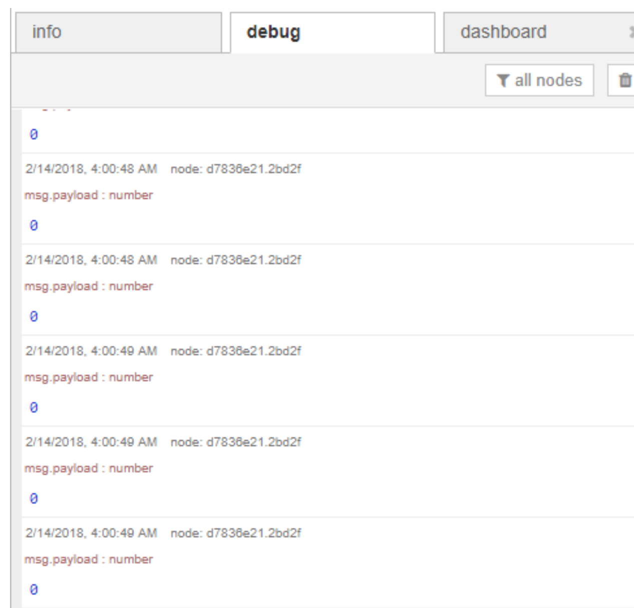

Figure 9: Node-red debug console for the weight sensor

7.4 Solenoid valve

On figure 10 we can see the flow-diagram for the solenoid valve. This flow-diagram is much simpler because we do not need to run an external script to activate the valve, it is purely running on the GPIO's. So when we inject "1" the valve opens when we inject "0" it closes, then that info is sent to the debug node(green) to be printed out.

Just for fun, a twitter node has been added, this node listens for any hashtag of "DonaldTrump" on Twitter. So if anyone hashtags DonaldTrump it will trigger the valve for 1 second, then shut it again. This goes to show again how easy Node-Red is to play around with.

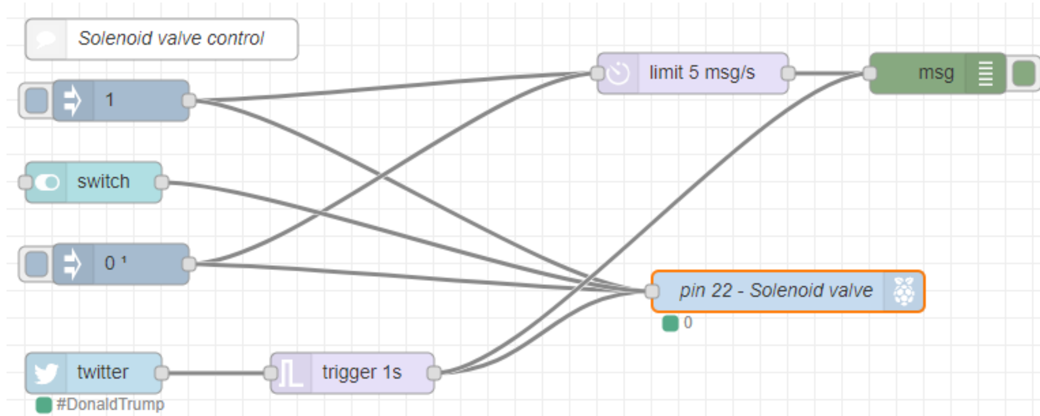

Figure 10: Node-red flow-diagram of the solenoid valve

Figure 11 shows the output of the debug nodes, the first 5 messages is injected manually, the last two comes from tweets of #DonaldTrump, it can be seen that the payload for these are also ”1” and ”0”.

info | debug | dashboard

all nodes

2/14/2018, 4:17:24 AM node: 24f0b44f.6b93fc
msg : Object
{ _msgid: "7c842a3a.5315d4", topic: "", payload: 1 }
2/14/2018, 4:17:26 AM node: 24f0b44f.6b93fc
msg : Object
{ _msgid: "bff7458.79565b8", topic: "", payload: 0 }
2/14/2018, 4:17:39 AM node: 24f0b44f.6b93fc
msg : Object
{ topic: "", payload: 0, _msgid: "f772be19.65dc1" }
2/14/2018, 4:17:42 AM node: 24f0b44f.6b93fc
msg : Object
{ _msgid: "1bb26f85.74395", topic: "", payload: 1 }
2/14/2018, 4:17:44 AM node: 24f0b44f.6b93fc
msg : Object
{ _msgid: "dbe14a8f.d73a38", topic: "", payload: 0 }
2/14/2018, 4:18:01 AM node: 24f0b44f.6b93fc
tweets/trump__toons : msg : Object
{ topic: "tweets/trump__toons", payload: "1", lang: "en", tweet: object, location: object … }
2/14/2018, 4:18:02 AM node: 24f0b44f.6b93fc
tweets/trump__toons : msg : Object
{ topic: "tweets/trump__toons", payload: "0", lang: "en", tweet: object, location: object … }

Figure 11: Node-red debug console for the solenoid valve

7.5 Node-red all sensors and controllable units

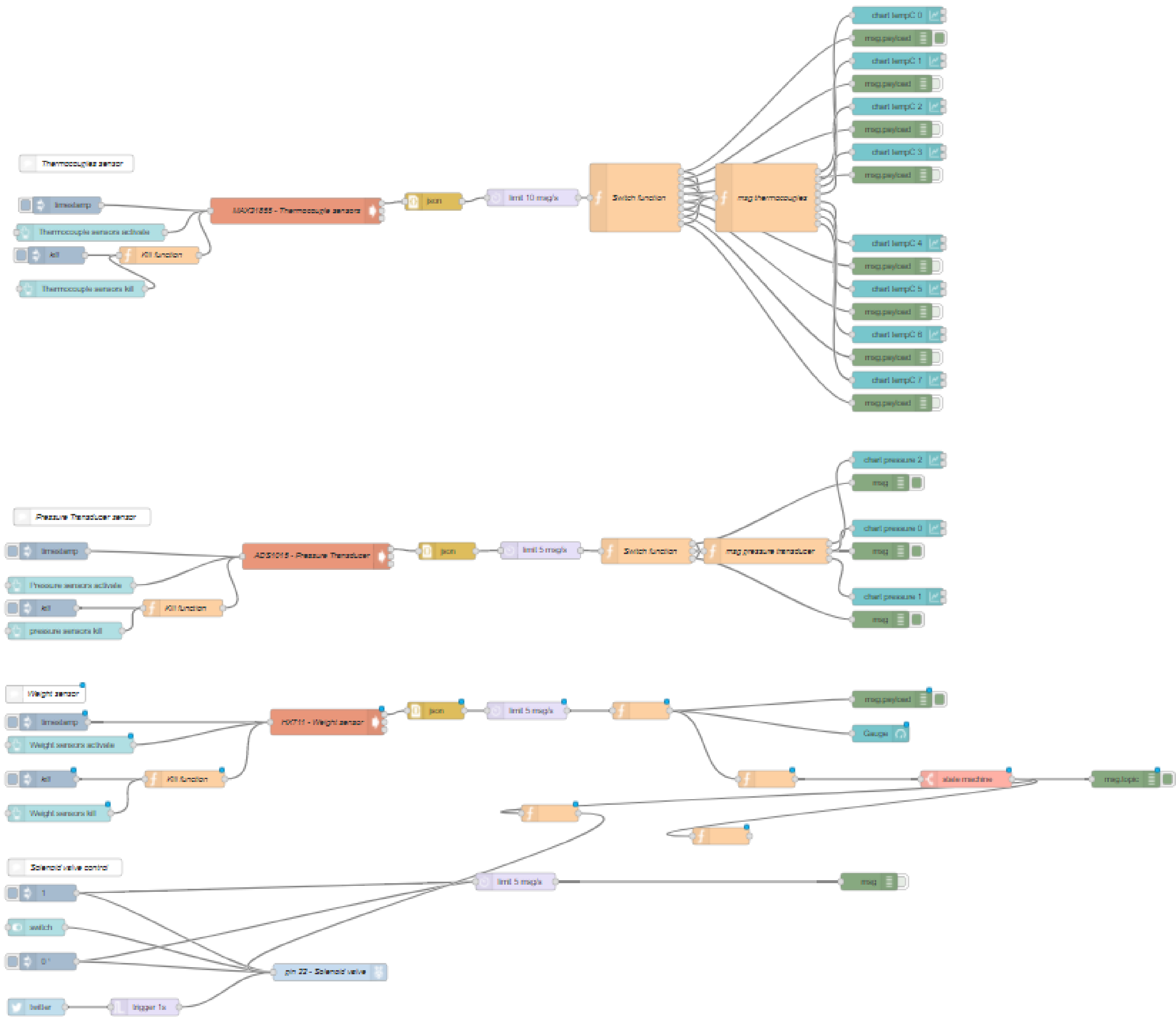

Figure 12: Node-red flow-diagram for everything

Figure 12 shows the overall flow-diagram of everything, in this case including the state machine(see next chapter). Basically you just deploy the node, and everything will be set and all the python scripts will be ready to be executed. That is how simple it is. The figure 13 below, shows the debug of all the sensors and controllable valves. As it can be seen all the sensors can run simultaneously without any problems.

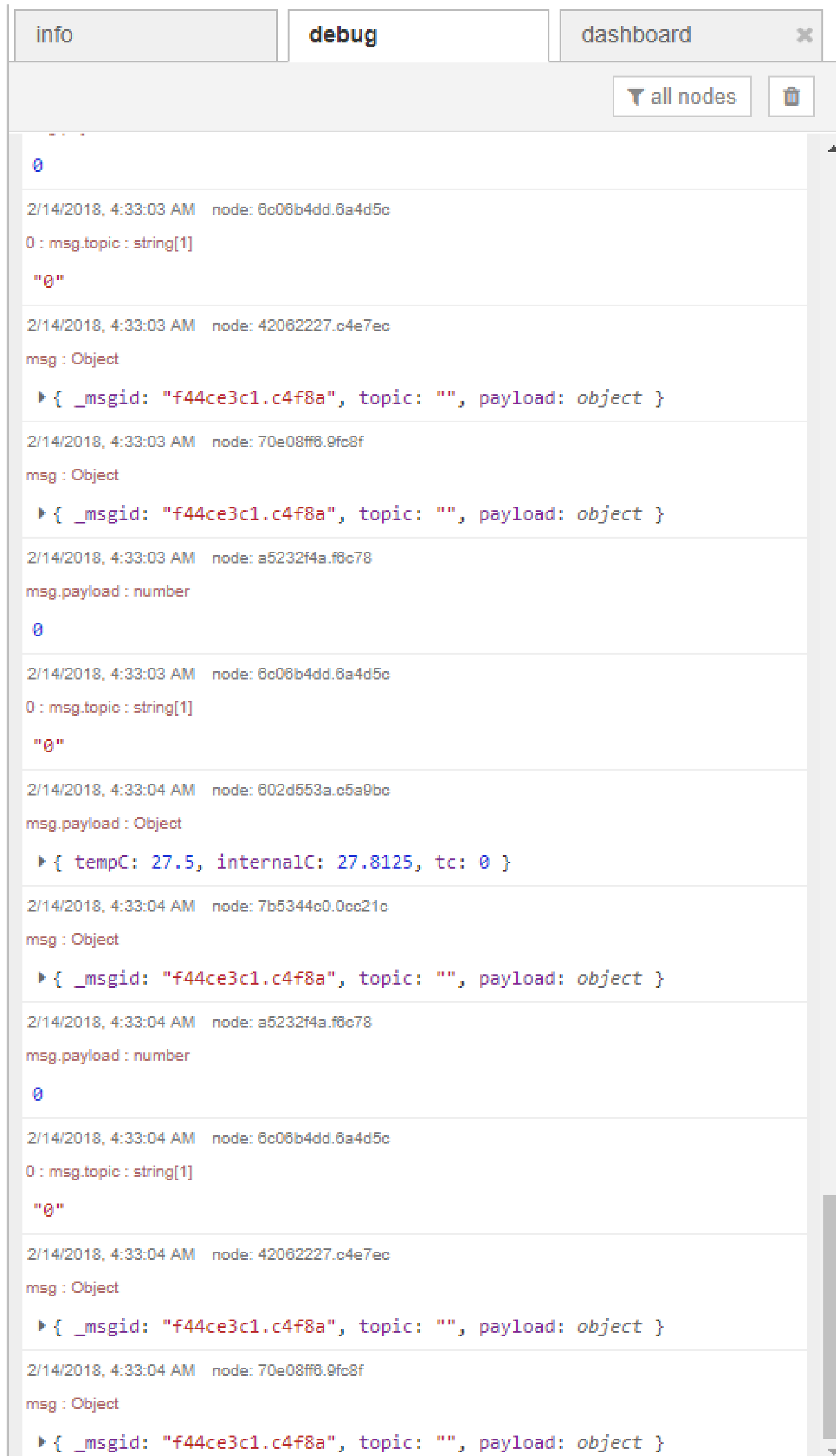

Figure 13: Node-red flow-diagram for everything

7.6 State machine

As explained before, a simple state machine has to be build to test out, how the sensors could be utilized to control the solenoid valves. See section *State machine* for the requirements.

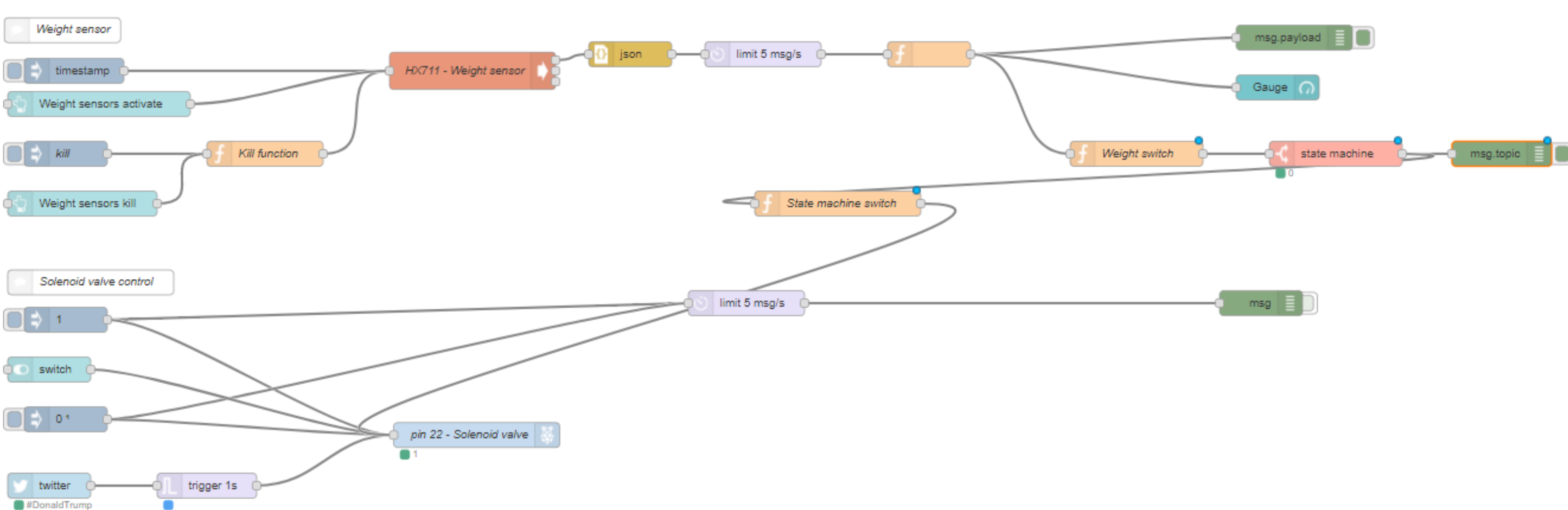

Figure 14: Node-red flow-diagram for the state machine

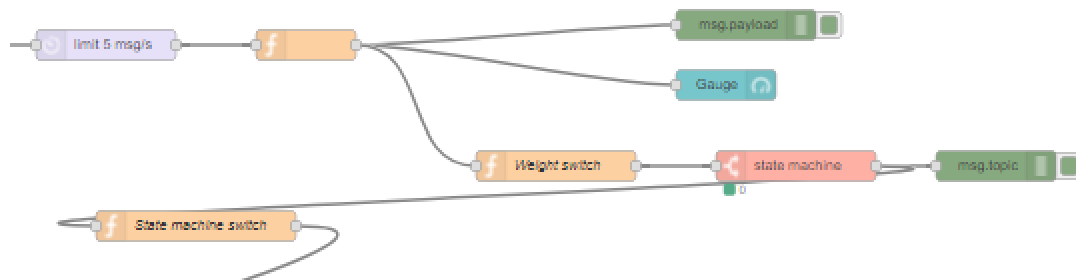

Figure 15: Node-red flow-diagram for the state machine zoomed in

Figure 14 and 15, shows the implementation of the state machine node, which is the pink color to the right(between the weight sensor and the solenoid valve controller). We can see that we have a Weight switch function node before the state machine node, this node is used to filter the data coming from the weight sensor. See figure 16. We can see the code shows, what should be sent to the state machine when the payload is 0 or in-between a specific number. Fx. if the weight reaches 250 to 500g the function node should return 500g as a string. Figure 17 shows the setup of the state machine. There are 4 states; 0,1,2 and 3. If the state machine receives a string reading 500g, it will trigger the first transformation, going from sate 0 til 1, if the string is 1000g it will trigger the transformation from stat 1 to 2, and so on.

Figure 18, shows the debug of the state machine(red) and the debug of the weight sensor(blue). It is possible to see that the weight in the beginning is 0. When we apply a force it goes up gradually to 30g then 383g, since we now are in-between 250 to 500 we will trigger first state transformation from 0 to 1, that is possible to be seen on figure 18 which show the number(as a string) ”1”. Then we increase the weight to 558g and the state trigger again to ”2”, and so

forth. Thus we can see that the state machine is working.

Figure 19 shows another function node, which is based on the output of the state machine to control the solenoid valve. basically what it says is if the state machine returns 3 activate the solenoid valve, when the state machine returns to state 0, deactivate the solenoid valve. In this way we can control the solenoid valve with a weight sensor.

unfortunately the test for controlling the solenoid valve with the thermocouples and pressure transducers was not made, therefore they are not included here, read more about it in the section *Discussion.*

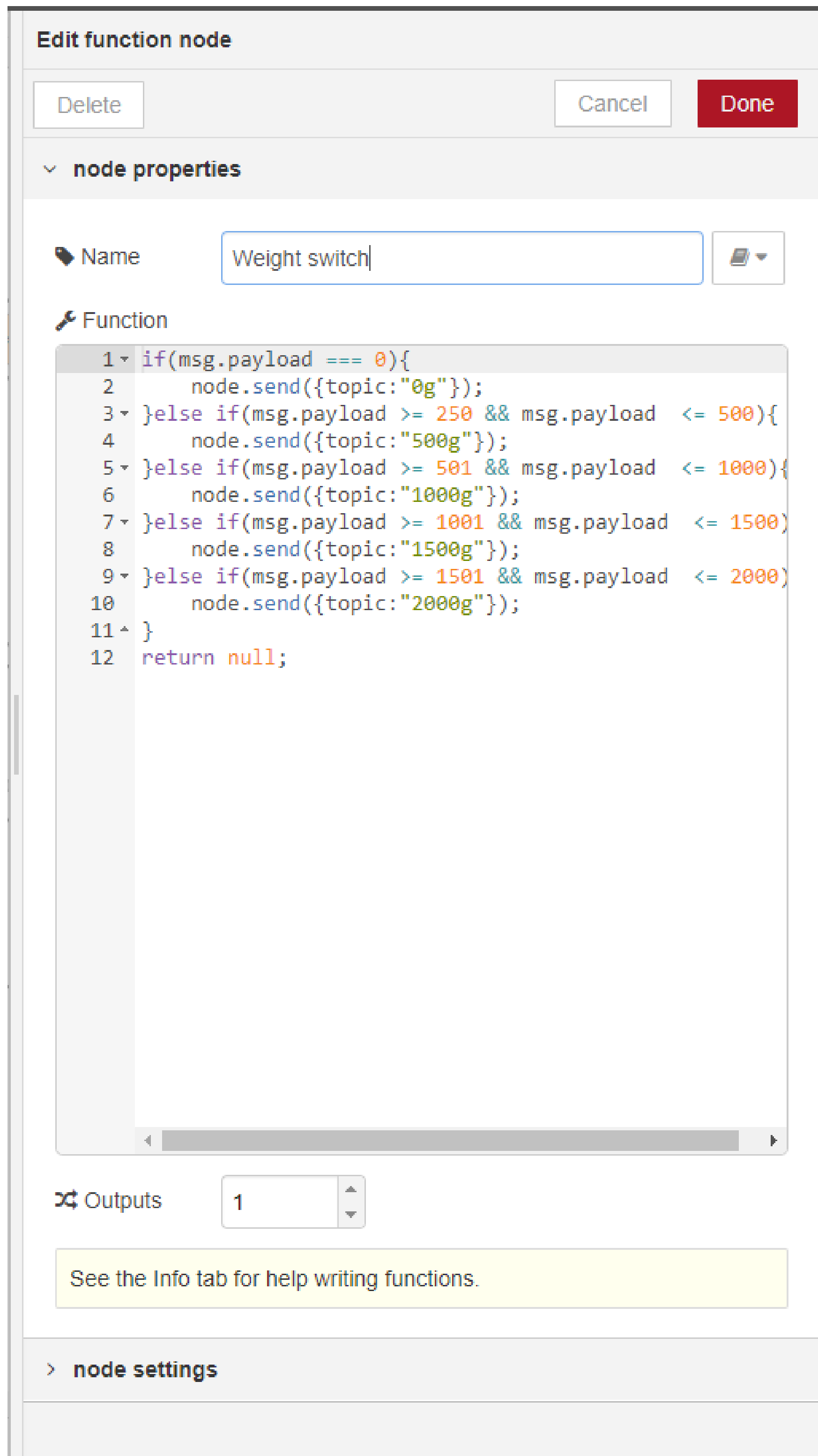

Figure 16: Node-red setting up the switch function for the state machine

Figure 17: Node-red setting up the state machine node

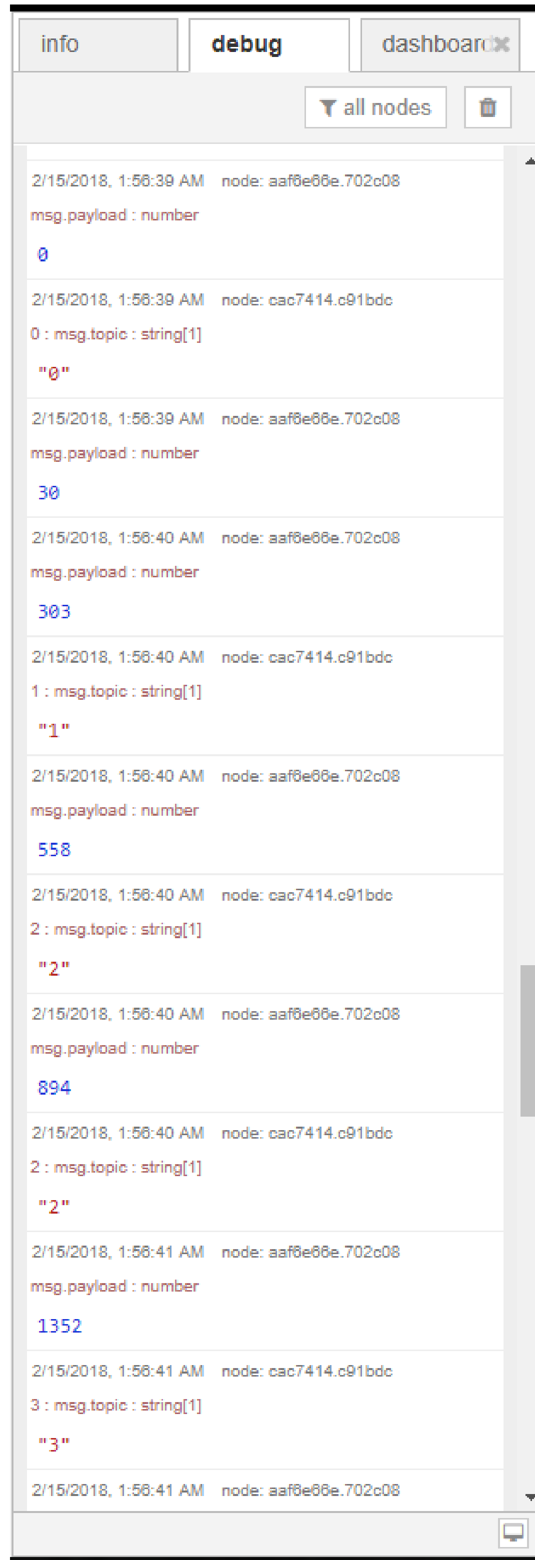

Figure 18: Node-red debug for state machine

Edit function node

Delete Cancel Done

node properties

Name: State machine switch

Function

```
if(msg.topic == "3"){
    node.send({payload:1});
}else if(msg.topic == "0"){
    node.send({payload:0});
}
return null;
```

Outputs: 1

See the Info tab for help writing functions.

node settings

Figure 19: Node-red State machine switch for solenoid valve

7.7 GUI for data and control

Below there are 4 figures 20, 21, 22 and 23. All are showing the GUI of Each sensor and the last solenoid valve. As it is possible to see we can start the data collection(figure 20,22 and 23) and we can kill it. At the same time on the most right, we can see the sensors moving. For the weight sensor figure 20, we can see a gauge showing the weight in grams. Figure 22, and 23 shows it as a graph going up and down. Figure 21 shows the solenoid valve, with this only it is obviously only possible to turn it on or off. The nodes for all of the GUI can be seen on figure 12, with the color of baby blue. On the upper left side is the button to switch between all the sensors.

Figure 20: Node-red GUI for weight sensor

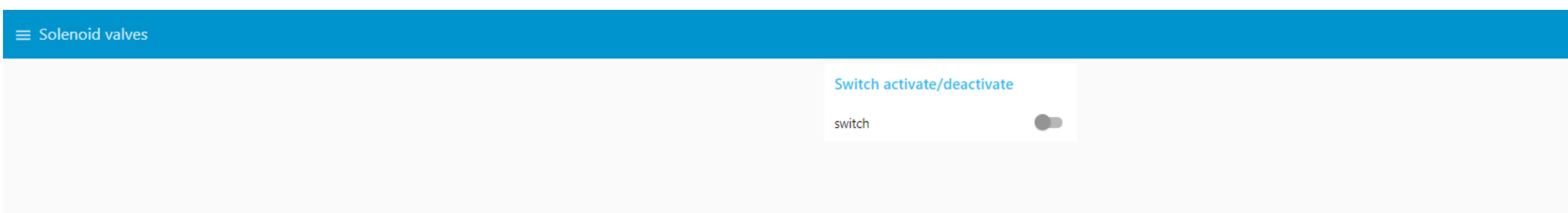

Figure 21: Node-red GUI for solenoid valve

Figure 22: Node-red GUI for thermocouples

Figure 23: Node-red State GUI for pressure transducer

7.8 Extending the WiFi capabilities

One of the goals of this project was to try to extend the WiFi capabilities of the Raspberry pi 3 model b, with an external WiFi antenna, since the N2O cooler is going to be placed inside a 20 feet metal container, the container is going to act as a Faraday cage, therefore it is important to extend the Pi with an antenna that can be placed outside the container.

First of all we have to physically modify the Raspberry pi 3, in order to be able to attach an external antenna. Figure 24 shows exactly that procedure, first of all two 0 ohm resistors had to be removed(too small to show on the pictures). Then a IPX U.FL connector was soldered, this connector can be seen on the picture, it is hard not to avoid it since the soldering looks bad - see figure 24. Figure 25 shows the adapter used to connect to any simple WiFi external connector than can be found.

Figure 24: Soldering a IPX U.FL connector on the Raspberry pi 3. to extend it with an external WiFi antenna

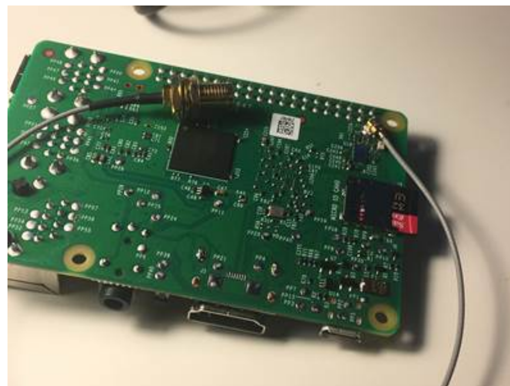

Figure 25: WiFi wire adapter is connected to the IPX U.FL connector

Figure 26 shows, before the modification, the available WiFi spots and their strength in dBm(the higher the better), and other information. Fami 1 is -48 dBm which is only 1 meter away, while Fami2 is 6 meters away through a wall shows -81.

```
pi@raspberrypi: /root
lqScan windowqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk
xFami1 A4:2B:B0:DE:8E:B4  89%, -48 dBm, ch   8, 2447 MHz ESS
xFami2 84:C9:B2:6A:B7:C2  41%, -81 dBm, ch  12, 2467 MHz ESS
```

Figure 26: WiFi without modification: show WiFi spots and their strength

Figure 27 shows link quality and signal level -47dBm and different sort of information. Figure 28 is with the antenna modification, but it is not working, clearly it is seen the strength is reduced down to -84dBm, since it is a logarithmic scale that is actually a lot. Also the link quality is bad. This is still Fami1 only 1 meter away from router. the reason for this is because the on-board antenna has been turned off and there is no external antenna attached to the Raspberry pi yet.

Figure 28 shows after solving the problem by adding the external antenna, the connection got much better. This is still fami1 connected only 1 meter away from the router.

It is clearly possible to see an improvement compared to the on-board antenna. So we went from -48 to -27(sometimes -25dBm). If dBm is converted to miliwatts, we get -48dBm=0.000016mW and -27dBm=0.001995. This means that the signal has been inproved by a factor of 125, which is really really good!

```
lqInterfaceqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk
xwlan0 (IEEE 802.11), phy 0, reg: DK (DFS-ETSI), SSID: Famil                                             x
tqLevelsqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqu
x                                                                                                        x
xlink quality: 90%  (63/70)                                                                              x
x===========================================================================================             x
x                                                                                                        x
x                                                                                                        x
xsignal level: -47 dBm (0.02 uW)                                                                         x
x===========================================================                                             x
x                                                                                                        x
tqStatisticsqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqu
xRX: 40,581 (4.10 MiB)                                                                                   x
xTX: 77,347 (9.78 MiB)                                                                                   x
tqInfoqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqu
xmode: Managed, connected to: A4:2B:B0:DE:8E:B4, time: 16:58m, inactive: 0.0s                            x
xfreq: 2447 MHz, channel: 8 (width: 20 MHz)                                                              x
xrx rate: 72.2 MBit/s, tx rate: 72.2 MBit/s                                                              x
xstation flags: WME TDLS, preamble: short, slot: short                                                   x
xpower mgt: on,  tx-power: 31 dBm (1258.93 mW)                                                           x
xretry: short limit 7,  rts/cts: off,  frag: off                                                         x
xencryption: off (no key set)                                                                            x
tqNetworkqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqu
xwlan0 (UP RUNNING BROADCAST MULTICAST)                                                                  x
xmac: B8:27:EB:78:93:54, qlen: 1000                                                                      x
xip: 192.168.1.105/24                                                                                    x
mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj
```

Figure 27: WiFi without modification: show WiFi spots and their strength, extended information

```
lqInterfaceqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk
xwlan0 (IEEE 802.11, WPA/WPA2),  ESSID: "Famil"                                                          x
tqLevelsqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqu
x                                                                                                        x
xlink quality: 37%  (26/70)                                                                              x
x=====================================                                                                   x
x                                                                                                        x
xsignal level: -84 dBm (3.98 pW)                                                                         x
x================                                                                                        x
x                                                                                                        x
x                                                                                                        x
tqStatisticsqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqu
xRX: 920 (58.39 KiB), invalid: 0 nwid, 0 crypt, 0 frag, 0 misc                                           x
xTX: 1,631 (346.51 KiB), mac retries: 90, missed beacons: 0                                              x
tqInfoqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqu
xmode: Managed,  access point: A4:2B:B0:DE:8E:B4                                                         x
xfreq: 2.447 GHz, channel: 8,  bitrate: 14.4 Mbit/s                                                      x
xpower mgt: on,  tx-power: 31 dBm (1258.93 mW)                                                           x
xretry: short limit 7,  rts/cts: off,  frag: off                                                         x
xencryption: n/a (requires CAP_NET_ADMIN permissions)                                                    x
tqNetworkqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqu
xwlan0 (UP RUNNING BROADCAST MULTICAST)                                                                  x
xmac: B8:27:EB:49:AC:C9, qlen: 1000                                                                      x
xip: 192.168.1.112/24                                                                                    x
mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj



F1info  F2lhist F3scan  F4      F5      F6      F7prefs F8help  F9about F10quit
```

Figure 28: WiFi with modification: No onboard antenna nor external

```
pi@raspberrypi: ~
lqInterfaceqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk
xwlan0 (IEEE 802.11, WPA/WPA2),  ESSID: "Famil"                              x
tqLevelsqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqu
x                                                                            x
xlink quality: 100%  (70/70)                                                 x
x============================================================================x
x                                                                            x
xsignal level: -27 dBm (2.00 uW)                                             x
x===================================================                         x
x                                                                            x
x                                                                            x
tqStatisticsqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqu
xRX: 3,955 (227.90 KiB), invalid: 0 nwid, 0 crypt, 0 frag, 0 misc            x
xTX: 7,033 (1.35 MiB), mac retries: 24, missed beacons: 0                    x
tqInfoqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqu
xmode: Managed,  access point: A4:2B:B0:DE:8E:B4                             x
xfreq: 2.447 GHz, channel: 8,  bitrate: 28.8 Mbit/s                          x
xpower mgt: on,  tx-power: 31 dBm (1258.93 mW)                               x
xretry: short limit 7,  rts/cts: off,  frag: off                             x
xencryption: n/a (requires CAP_NET_ADMIN permissions)                        x
tqNetworkqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqu
xmac: B8:27:EB:49:AC:C9, ip: 192.168.1.112/24                                x
mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj
F1info  F2lhist F3scan  F4      F5      F6      F7prefs F8help  F9about F10quit
```

Figure 29: WiFi with modification: with external antenna

8 Discussion

For this project it is possible to say that it we have achieved most of the goals, or the so called requirements set in the beginning of this report. The N2O cooler can be controlled automatically and manually, by simultaneously using either a state machine or human interference's trough the GUI. All the data is shown on the GUI or the Debug console, this was an important requirement since we need to be able to motorize the situation of the N2O. We managed to build a simple state machine to control the solenoid valve with the weight sensor, this could easily have been extended to also include the pressure transducers and thermocouples, but the project went over due. The state machine was a success since it actually could control itself and decide what to do with the solenoid valve on a given input. The GUI was also a success since it showed all the data and was able to control the sensors, this was done through a web application running on WiFi meaning all who had the IP could access it and monitor the situation. Sadly the final product did not include a successful logging system. This was neglected since it is a matter of adding a node that saves all the data on sql-lite, it could have been done within 2 min of work, so it was not a challenge in it self, therefore other harder requirements where in focus to be solved.

9 Conclusion

It can be concluded that nearly all of the requirements set, in the beginning of the report, was achieved successfully. We can say that it is possible to build a system running on a microchip and not necessary a microcontroller to monitorize a given system, in our case a N2O cooler. So there is no problem for the microcontroller to handle multiple sensors at the same time, in fact, it might be the best option, since it is easier to just use Node-Red to connect everything together without doing a lot of coding, and put the focus on making the sensors working with their own python code, this helps and saves a lot of time. Furthermore we can conclude that we successfully implemented a GUI, that can monitor all the sensors, and to control the solenoid valve. Lastly we can conclude that modifying the on-board WiFi on the Raspberry pi was very successful, the general strength of the WiFi increased by a factor of roughly 125.

Thus this project as been success and a good starting point to figure out how to further develop the system, in order for it to be robust and capable of never failing.

10 Appendix A - Octo board thermocouples python code

```
#Provided by a helpful customer - thanks!

#Here is the code written in python tested and working
   with the device on raspberry pi. You can adjust the
   sleep time as you want but you cannot go any lower than
    .125 ms from my experience.

import time
import json
import Adafruit_GPIO.SPI as SPI
import Adafruit_MAX31855.MAX31855 as MAX31855
import RPi.GPIO as GPIO

# Define a function to convert celsius to fahrenheit.
def c_to_f(c):
 return c * 9.0 / 5.0 + 32.0

# Raspberry Pi software SPI configuration(GPIO)
CLK = 11
CS = 8
DO = 9
sensor = MAX31855.MAX31855(CLK, CS, DO)
T0 = 17
T1 = 27
T2 = 22
GPIO.setmode(GPIO.BCM)
GPIO.setup(T0, GPIO.OUT)
GPIO.setup(T1, GPIO.OUT)
GPIO.setup(T2, GPIO.OUT)
tc = 0
# Loop printing measurements every second.
print('Press Ctrl-C to quit.')
while True:
 GPIO.output(T0, tc & 1<<0)
 GPIO.output(T1, tc & 1<<1)
 GPIO.output(T2, tc & 1<<2)
 time.sleep(0.125)
 tempC = sensor.readTempC()
 internalC = sensor.readInternalC()
 print json.dumps({"tc":tc, "tempC":tempC, "internalC":
    internalC})
 tc = tc+1
 if tc == 8:
  tc=0
```

Listing 1: Python example

11 Appendix B - Pressure transduceres python code

```
from time import sleep
import Adafruit_ADS1x15
import json

adc = Adafruit_ADS1x15.ADS1015(address=0x48, busnum=1)

# Gain = 2/3 for reading voltages from 0 to 6.144V.
# See table 3 in ADS1115 datasheet
GAIN = 2/3

# Main loop.
while 1:
    values = [0]*3

    for i in range(3):

        values[i] = adc.read_adc(i, gain=GAIN)
        # Ratio of 15 bit value to max volts determines
           volts
        volts = values[i] / 2047.0 * 6.144
        # Tests shows linear relationship between psi &
           voltage:
        psi = 50.0 * volts - 10
        # Bar conversion
        bar = psi * 0.0689475729

        print json.dumps({"i":i, "bar":bar})
        sleep(0.250)
```

Listing 2: Python example

12 Appendix C - Load cell python code

```
import RPi.GPIO as GPIO
import time
import sys
import json
from hx711 import HX711

def cleanAndExit():
    print "Cleaning..."
    GPIO.cleanup()
    print "Bye!"
    sys.exit()

hx = HX711(5, 6)
```

```
14
15 # I've found out that, for some reason, the order of the
     bytes is not always the same between versions of python
     , numpy and the hx711 itself.
16 # Still need to figure out why does it change.
17 # If you're experiencing super random values, change these
      values to MSB or LSB until to get more stable values.
18 # There is some code below to debug and log the order of
     the bits and the bytes.
19 # The first parameter is the order in which the bytes are
     used to build the "long" value.
20 # The second paramter is the order of the bits inside each
      byte.
21 # According to the HX711 Datasheet, the second parameter
     is MSB so you shouldn't need to modify it.
22 hx.set_reading_format("LSB", "MSB")
23
24 # HOW TO CALCULATE THE REFFERENCE UNIT
25 # To set the reference unit to 1. Put 1kg on your sensor
     or anything you have and know exactly how much it
     weights.
26 # In this case, 92 is 1 gram because, with 1 as a
     reference unit I got numbers near 0 without any weight
27 # and I got numbers around 184000 when I added 2kg. So,
     according to the rule of thirds:
28 # If 2000 grams is 184000 then 1000 grams is 184000 / 2000
      = 92.
29 #hx.set_reference_unit(113)
30 hx.set_reference_unit(-207)
31
32 hx.reset()
33 hx.tare()
34
35 while True:
36     try:
37   time.sleep(0.175)
38         # These three lines are usefull to debug wether to
               use MSB or LSB in the reading formats
39         # for the first parameter of "hx.
              set_reading_format("LSB", "MSB")".
40         # Comment the two lines "val = hx.get_weight(5)"
              and "print val" and uncomment the three lines
              to see what it prints.
41         # np_arr8_string = hx.get_np_arr8_string()
42         # binary_string = hx.get_binary_string()
43         # print binary_string + " " + np_arr8_string
44
45   # Prints the weight. Comment if you're debbuging the MSB
        and LSB issue.
46
```

```
47     val = max(0, int(hx.get_weight(5)))
48         print json.dumps({"val":val})
49 
50         #hx.power_down()
51         #hx.power_up()
52     except (KeyboardInterrupt, SystemExit):
53         cleanAndExit()
```

Listing 3: Python example