Kimma's Vending Machine controlled by Vemio-2 and Raspberry Pi

September of 2019

Table of Contents

1 Introduction

This document was written as a way to document the work done throughout 2 weeks, rewiring the machine with VEMIO-2, and building the control software based on a raspberry pi 3b+.
VEMIO-2 can drive a matrix of 8×12 motors, although in this document we will reference a 6×10 configuration, and has some more features, which are explained in detail in the section below. For more specifications, please consider the technical documentation of the board (linksavailable in section 5).

2 Hardware

2.1 VEMIO-2 Overview

The operating principle of the machine is based on the VEMIO-2 board, represented in the figure below, which has direct control over the outputs (motors), can read digital inputs and even a temperature sensor needed to control the air conditioner present inside the machine. This board also has terminals for connecting a common 2 relay board

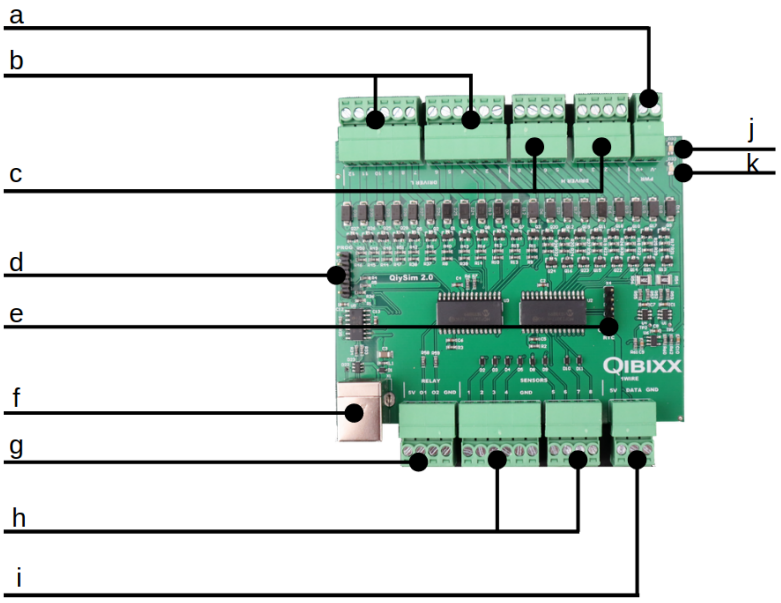


Figure 1: VEMIO-2

Explaining the previous picture, in regards to a practical implementation, we can see:

a) Outputs power connection (12-24V) - This is the power that will be passed to the motors (VCC) so, one must choose the power source accordingly;

b) Low Drive Outputs (outputs 1 to 12) - These must be wired to the ground terminals (GND) of the motors and will select the row to activate the desired (odd/even) motor. Therefore, the active state of this pins is low driven and the inactive state is high impedance;

c) High Drive Outputs (outputs 25 to 32) - These terminals must be wired to the positive terminals (Vcc) of the motors and will select the desired (odd/even) motor to activate, accordingly to the Low Drive selection. Therefore, the active state is with low impedance connection to the supply voltage and the inactive state is disconnected;

d) Factory programming connector - not used;

e) RTC connector - not used;

f) USB input - used for communication with the VMC, which is, in this case, the raspberry pi;

g) 2 relay board connector (outputs 13 and 14)- Digital ttl outputs. These pins should be used to trigger the relays for the cold machine and for the fan;

h) Digital inputs - These, which in inactive state are high, are used to get feedback from the motors home position and from the door signal.

i) DS18B20 connector - Used to connect one-wire temperature sensor;

j) Green LED indicator (output 16) - Can be programmed to get an easy and instant feedback;

k) Red LED indicator (output 15) - Can be programmed to get an easy and instant feedback.

This I/O board, accepts the following ASCII commands:

- o<x>,1 - Activates an output, where x is the output number;
- o<x>,0 - Deactivates an output, where x is the output number;
- a - reads current sensor (a1 - start continuous reporting; a2 - stop continuous reporting);

- i - reads digital inputs (i1 - start continuous reporting; i2 - stop continuous reporting);
- T - reads one-wire temperature sensor.

All commands must be followed by a line feed ("\n").

Simplifying things a little bit, if the connections are properly done, one can think like a matrix and rotate each motor, calling it by cartesian coordinates. The following table shows the commands that must be sent, in order to activate each motor of the 6 by 10 matrix of the machine.

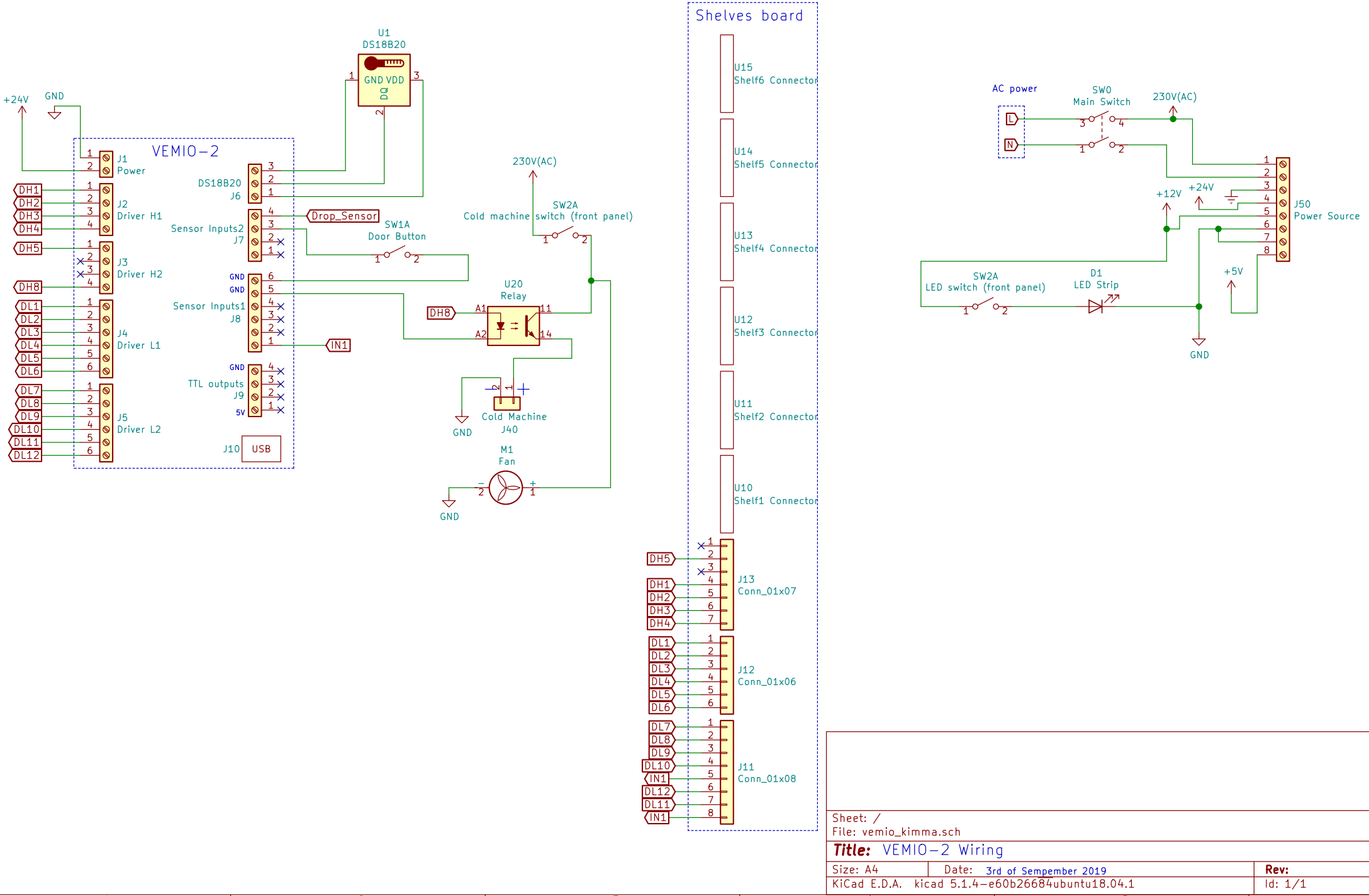
KIMMA Motors Matrix											
	Column (x axis)										
Row (y axis)	(x,y)	Motor 1	Motor 2	Motor 3	Motor 4	Motor 5	Motor 6	Motor 7	Motor 8	Motor 9	Motor 10
	Shelf 6	O12,1(0) O25,1(0)	O11,1(0) O25,1(0)	O12,1(0) O26,1(0)	O11,1(0) O26,1(0)	O12,1(0) O27,1(0)	O11,1(0) O27,1(0)	O12,1(0) O28,1(0)	O11,1(0) O28,1(0)	O12,1(0) O29,1(0)	O11,1(0) O29,1(0)
	Shelf 5	O10,1(0) O25,1(0)	O9,1(0) O25,1(0)	O10,1(0) O26,1(0)	O9,1(0) O26,1(0)	O10,1(0) O27,1(0)	O9,1(0) O27,1(0)	O10,1(0) O28,1(0)	O9,1(0) O28,1(0)	O10,1(0) O29,1(0)	O9,1(0) O29,1(0)
	Shelf 4	O8,1(0) O25,1(0)	O7,1(0) O25,1(0)	O8,1(0) O26,1(0)	O7,1(0) O26,1(0)	O8,1(0) O27,1(0)	O7,1(0) O27,1(0)	O8,1(0) O28,1(0)	O7,1(0) O28,1(0)	O8,1(0) O29,1(0)	O7,1(0) O29,1(0)
	Shelf 3	O6,1(0) O25,1(0)	O5,1(0) O25,1(0)	O6,1(0) O26,1(0)	O5,1(0) O26,1(0)	O6,1(0) O27,1(0)	O5,1(0) O27,1(0)	O6,1(0) O28,1(0)	O5,1(0) O28,1(0)	O6,1(0) O29,1(0)	O5,1(0) O29,1(0)
	Shelf 2	O4,1(0) O25,1(0)	O3,1(0) O25,1(0)	O4,1(0) O26,1(0)	O3,1(0) O26,1(0)	O4,1(0) O27,1(0)	O3,1(0) O27,1(0)	O4,1(0) O28,1(0)	O3,1(0) O28,1(0)	O4,1(0) O29,1(0)	O3,1(0) O29,1(0)
	Shelf 1	O2,1(0) O25,1(0)	O1,1(0) O25,1(0)	O2,1(0) O26,1(0)	O1,1(0) O26,1(0)	O2,1(0) O27,1(0)	O1,1(0) O27,1(0)	O2,1(0) O28,1(0)	O1,1(0) O28,1(0)	O2,1(0) O29,1(0)	O1,1(0) O29,1(0)
Bottom of the machine											

Table 1: Motor matrix

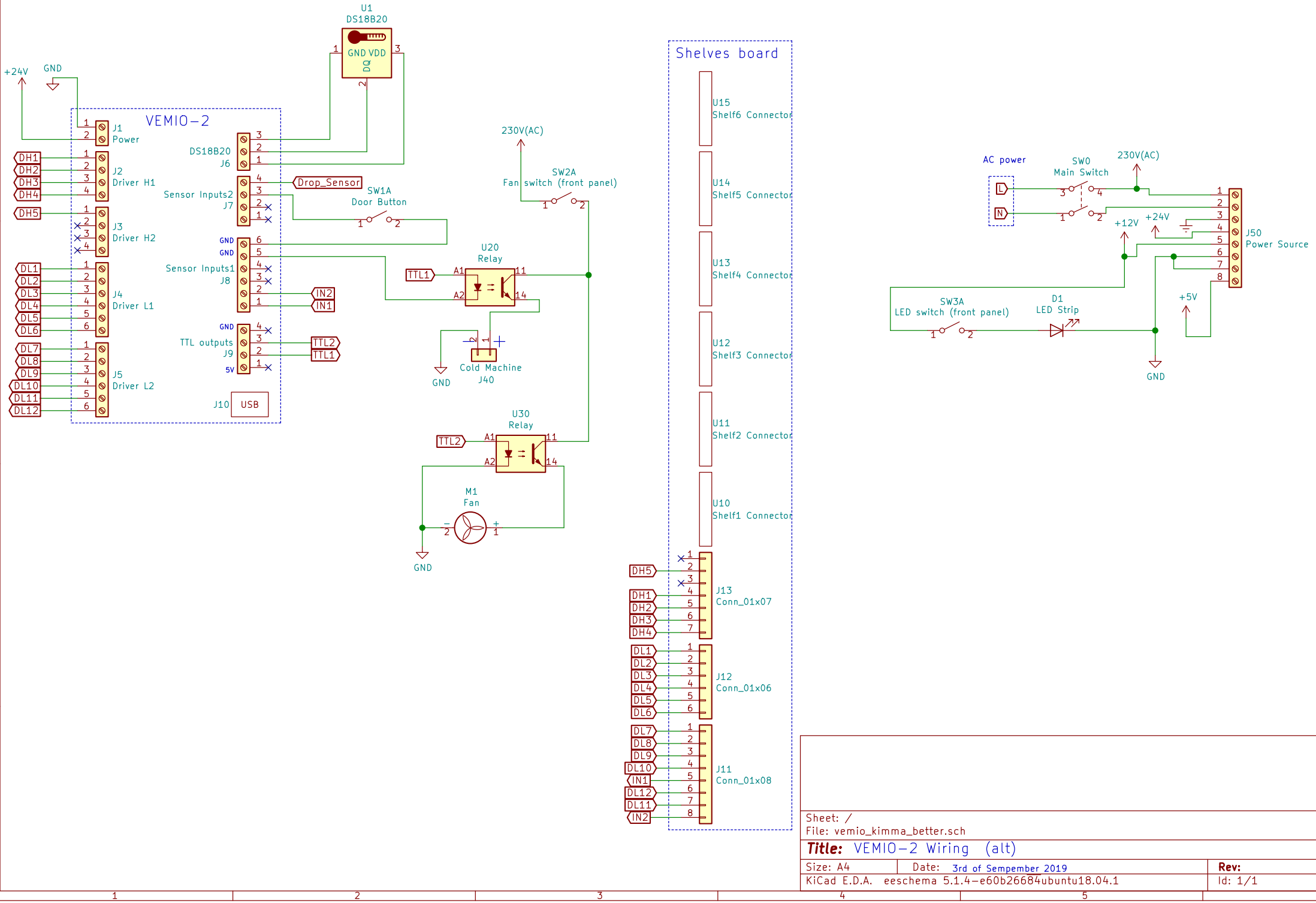
2.2 Wiring and Connections

For a better understanding of how everything is connected, please take a look at the schematics below. Please note that there can be many possible ways to wire the peripherals such as relays, cold machine, fan, etc. Therefore, 2 schematics are presented in this document. The first one, which was reproduced entirely, and a second one which is more conventional. Also some more functionalities can be added, depending on the application.

Please consider also that, since there was not an easy way to communicate with the RS-485 drop sensor, a small hack was made, pulling a wire from the sensor, which is normally high and, on detecting an object, it will go to 0V, thus allowing to read it with the digital inputs of VEMIO-2 and simplifying this part of the project.



Sheet: /		
File: vemio_kimma.sch		
Title: VEMIO−2 Wiring		
Size: A4	Date: 3rd of Sempember 2019	**Rev:**
KiCad E.D.A. kicad 5.1.4−e60b26684ubuntu18.04.1		Id: 1/1



Sheet: /
File: vemio_kimma_better.sch

Title: VEMIO-2 Wiring (alt)

Size: A4	Date: 3rd of Sempember 2019	Rev:
KiCad E.D.A. eeschema 5.1.4-e60b26684ubuntu18.04.1		Id: 1/1

3 Software

The controller software is divided in 3 distinct parts which, in the main thread, communicate back and forth, based on the user inputs:

1) **kimma_daemon.lua** - accepts TCP commands and communicates with VEMIO-2 thus interfacing with the low level, activating the I/O's, reading serial messages from VEMIO and proccessing them asynchronously. The script is listening on port 31421 and only accepts commands from localhost, by default. It accepts the following TCP commands:

   - VEND,<x>,<y>,<(optional)vendTimeout:seconds>- starts the vending process for item in the <x>,<y>coordinates;
   - T,<temperature>- sets the temperature for the machine in Celsius (default is 10^{o}C);

   However, it is possible to run the daemon with some parameters (default):

   - timeout-vend (10) - timeout for vending approval by cashless device;
   - tcp-client-timeout (3000) - timeout for clients to send VEND;
   - tcp-allow-remote-hosts (false) - to allow tcp connections outside 127.0.0.1 set this to true;
   - tcp-port (31421) - TCP server port;
   - debug - enable debug output

   To use these specifications just invoke them with the run command, like in the example below:

```
lua kimma_daemon.lua tcp-allow-remote-hosts=true debug=
    true
```

   Summarizing this part, there are 2 main threads. The vending (main) thread and the temperature one. Since the vending process will be triggered conventionally from the user interface, it will not yet be explained in this topic but further below. Either way, the temperature thread is independent of the UI so, it follows a simple flowchart explaining that one:

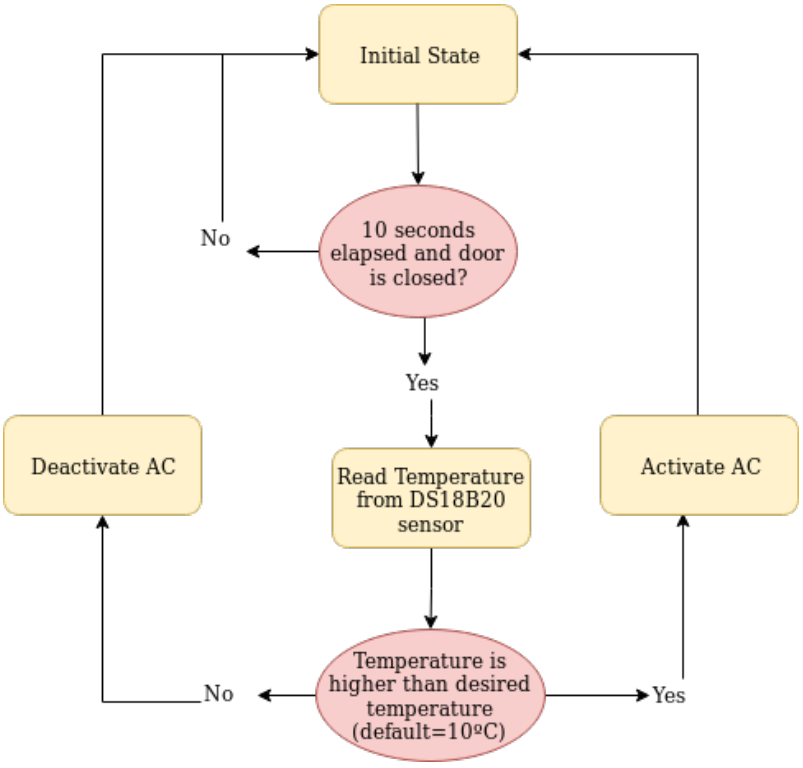


Figure 2: Temperature thread flowchart

2) **process_donation.c** - this script, acts as a CGI (Common Gateway Interface) between the WEB UI and the LUA script, therefore, converting GET requests in the TCP commands that the daemon is expecting to receive.
Since the user will only want to get some product out of the machine, the only possible request that the CGI will process is the VEND command, with the parameters to choose the motor to activate. Thus, one can send a GET request to activate the motor placed in coordinates 4,4 like the following:

```
http://<ip adress>/proces_donation.bin?x=4&y=4
```

After receiving a request it will automatically call the underlying cashless master daemon. And respond with http header:

```
Content-Type: text/plain;charset=utf-8
```

and following by the body:

```
status:<status code>
message:<message received from cashless master daemon>
```

Please note that the second line (message) is only returned if status=0.
Possible status codes (please note that those are not http status codes):

- 0 - success;
- 20 - argument x is not present;
- 21 - argument y is not present
- 4 - could not establish link to daemon
- 5 - could not resolve host name
- 6 - communication link failure

In case of status code 0 (success), the CGI will send the appropriate TCP command (VEND,<x >,<y >) to the LUA daemon, which will start the vending process and then, it will also print it's response.

3) **React app** - This is the top layer which creates the UI in the browser, interfaces with the CGI, sending GET request, and controls the dash payment. For having a better idea of what was implemented, a simple flowchart was made which follows in the figure 3.

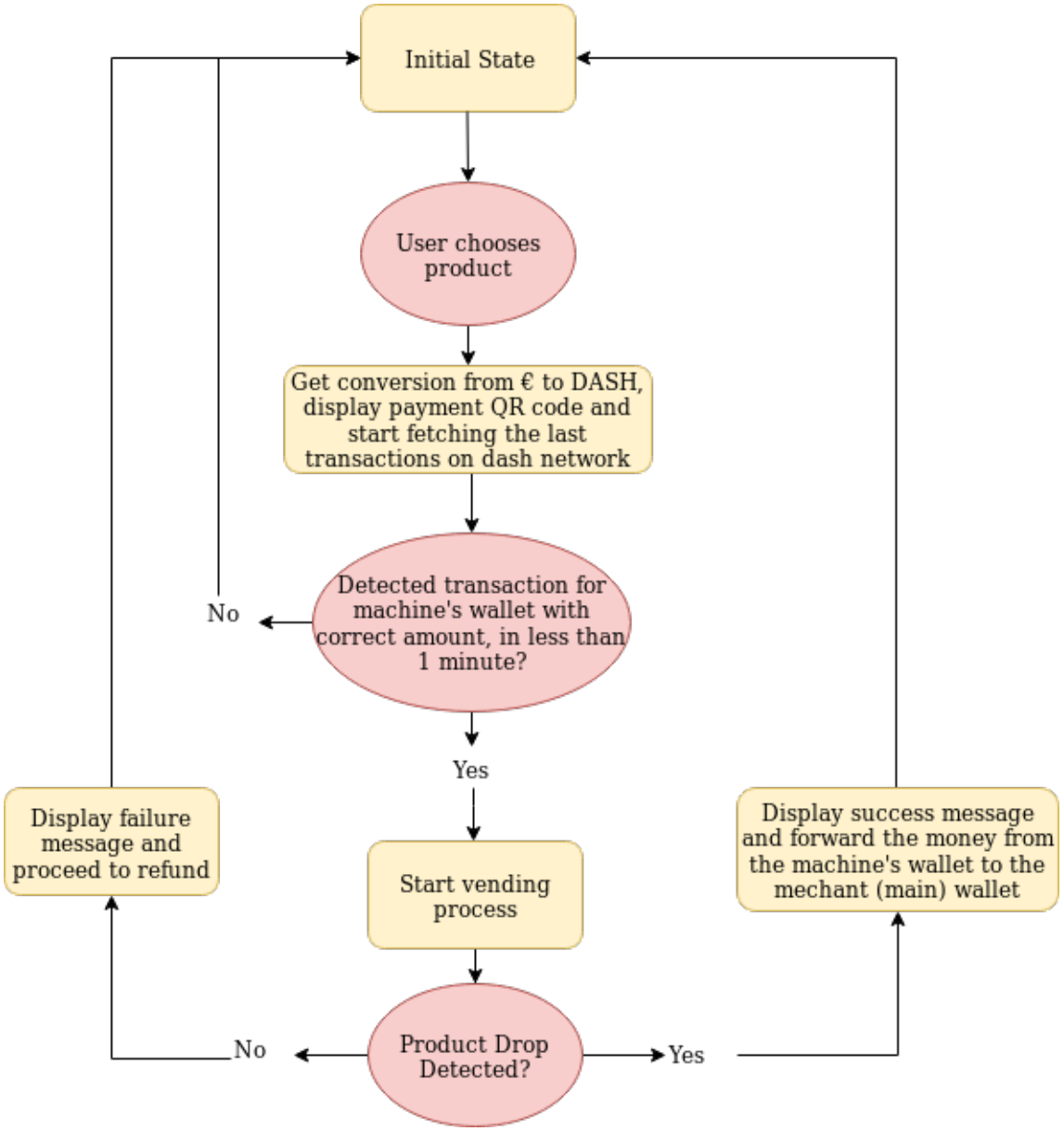


Figure 3: React app flowchart

4 Conclusion

Again, there are many possible ways to implement this logic with other programming languages, frameworks, interfaces, etc.
We found that this would be a good way to provide some layers of abstraction and to keep things more encapsulated.

5 Useful Links

- VEMIO-2 User Guide - https://qiba.pt/wp-content/uploads/2019/09/VEMIO-2-USER-GUIDE.pdf
- VEMIO-2 Product Brief - https://qiba.pt/wp-content/uploads/2019/09/VEMIO-2.pdf
- QIBA home page - https://qiba.pt/#pll_switcher
- Contact Us - https://qiba.pt/contact-us/