



Smart Laundry Management



by simonvdhende

Dandywash is a smart laundry management system, oriented towards people who have little time to spend on trite household tasks like doing laundry. We've all been there, just tossing our dirty clothes in the basket, hoping to find motivation to sort through the mess later. However, no one ever finds it. Until we really need some piece of clothing and can't find it anywhere. That's just the start. Then comes the sorting, filling and tracking. Doing this simple and repetitive task takes up way too much attention and focus. That is exactly why I started this project. Dandywash eliminates all of these tiresome activities. You no longer have to spend another second sorting, tracking or measuring your loads. While maintaining full control. Find out more, and how you can achieve the same productive result, by reading through this article.

Supplies:

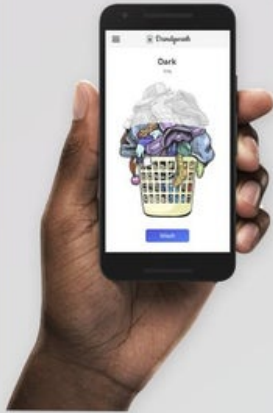
I created a detailed bill of material in Excel, which you can view [here](#).

This contains all essential bits and pieces you need, and where to get them.

On top of those, I would like to list some additional items that will come in very handy when making this project yourself, but are not obligated.

- Since you'll need some long jumper wires and those aren't really a thing, I suggest you buy both female - female cables as male - male cables. I also bought female - male but those aren't really necessary. This way, you can create longer cables by chaining them together. This eliminates time consuming soldering work.
- I have also added a lot of safety resistors in the circuit. Feel free to take those out if you feel extra confident. If you are running low on resistors I recommend you pick up [this kit](#), it's very convenient to always have the resistors you need, labelled clearly.

1MCT - Project 1
Simon Van Den Hende



Dandywash

Smart laundry management. No longer spend any time sorting, tracking or measuring your loads. All while maintaining full control. Optimized for people who have little time or energy to spend on trite household tasks.





<https://vimeo.com/429283066>

Step 1: Preliminary

Booting the Raspberry Pi

In order to run a whole IOT chain from the Raspberry Pi, we need to initialize the device. This can be done by downloading the [provided image](#), and burning it onto a micro SD card (16GB). This can be done using Win32DiskImager or any other software really. Make sure your SD card is completely empty and formatted before burning the image. [This video](#) explains the whole process step by step. Note that you do not need to use the raspbian image but the provided image in stead.

When you are done writing the SD card, you can remove it and insert it in the Pi. Make sure the Pi is not connected to the power yet!

When the SD card is inserted, connect the Pi to your laptop using an ethernet cable. Only then, when it is already in your control, give it power. The Pi will boot in a couple of seconds.

You can monitor this by going in the command prompt and typing

```
> ping 169.254.10.1 -t
```

When you **get a reply** rather than a 'Host Unreachable', your Pi has successfully booted. This means we can interact with it. Exit the infinite loop of pinging by pressing Ctrl+C. Now you can enter the Pi by typing

```
> ssh pi@169.254.10.1
```

this will prompt you for the password, which is the default **raspberry**.

When booting for the first time, it is generally good practice to run both

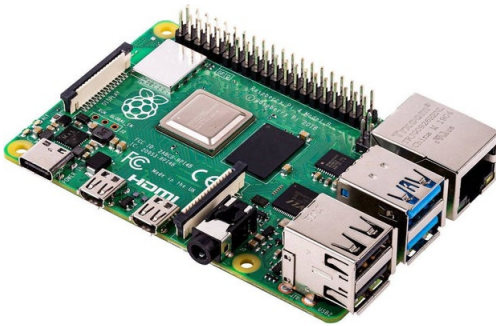
```
$ sudo apt-get update  
$ sudo apt-get upgrade
```

This will ensure all packages are updated and on the latest version.

MariaDB and Apache2 will already be installed. So we don't have to worry about those. We do, however, have to set up some other things in order to get everything functioning the way we want.

However, you should reboot first, to make sure everything is ready for the next step.

```
$ sudo reboot
```



```
C:\Users\Simon>ssh pi@169.254.10.1  
pi@169.254.10.1's password:  
Linux raspberrypi 4.19.75-v7l+ #1270 SMP Tue Sep 24 18:51:41 BST 2019 armv7l  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Tue May 26 09:41:40 2020 from 169.254.86.139  
pi@raspberrypi:~ $
```

Step 2: Setting Up the Database

We will set up the database using your laptop / desktop, not the Pi. Open up MySQL Workbench ([download guide](#)) and add a new connection.

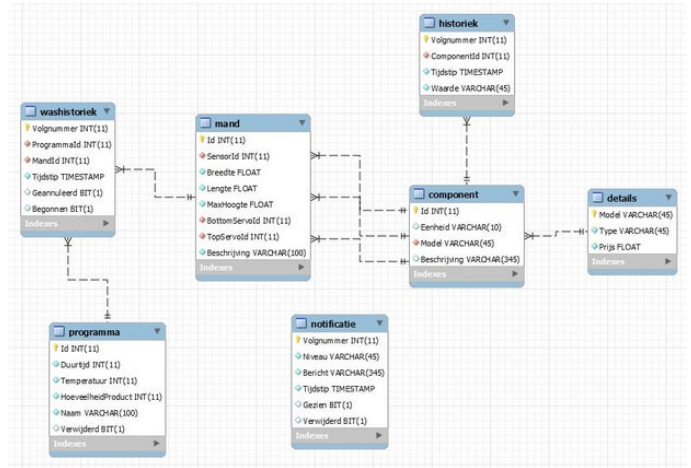
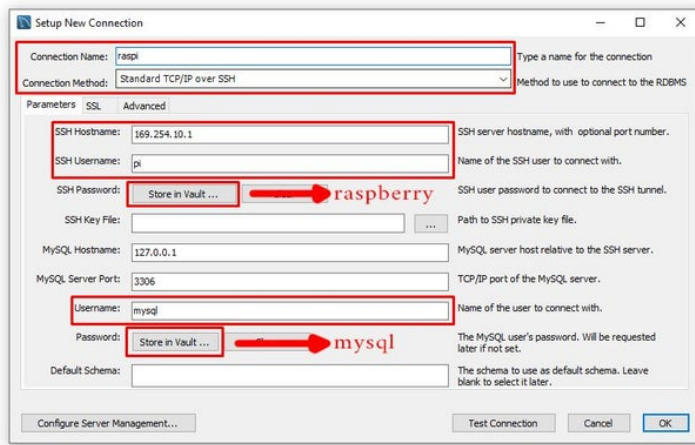
Afterwards, you will be prompted with a configuration window. Mine is filled in the way yours should be. Pay close attention to the marked fields. The arrows point to the passwords you have to store in the vault. These are just defaults and can be changed to your likings.

When all info is entered, click on **Test Connection**, ignore the warning, and hopefully see the succes window. If you don't, some field(s) are wrong. You can

proceed by clicking **Ok** on the window with all the input fields.

The connection should now be visible in the starting window. Click on it to try connecting. The password should be entered automatically since we stored it in the vault.

The last step is to import the database. You can download the dump [here](#). [This video](#) explains how to open and run a .sql file. Make sure you are connected to the Raspberry Pi, and not the local instance on your laptop!



Step 3: Setting Up the Git Repository

Working with a git repo is pretty much necessary here. Especially if you want to easily switch between your pc and raspi. Git should already be installed on the device, so you can just git clone whatever repo you want to whatever folder you want. However, since we are using apache, we need to put our Frontend code (html, css, javascript) in the /var/www/html folder. I don't want to put the whole repo here, and I definitely don't want a separate repo.

This can be solved by creating a symbiotic link, which is essentially the same as a shortcut in windows. It can easily be set up by typing the following command in the raspi terminal (after cloning the repo!)

```
$ git clone <a href="https://github.com/howest-mct/1920-1mct-project1-VanDenHendeSimon.git" rel="nofollow"> https://github.com/howest-mct/1920-1mct-project1-...</a>
```

Creating a symbiotic link has the following structure

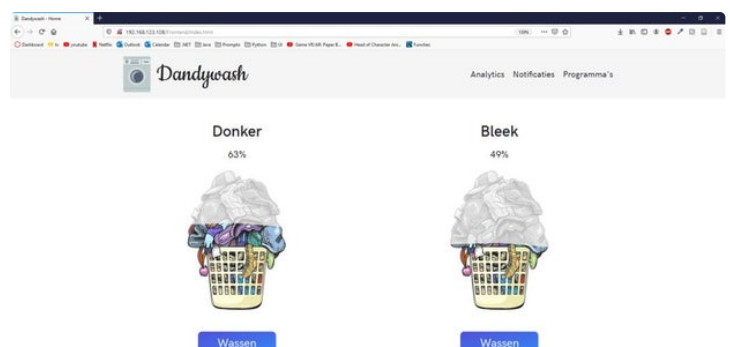
```
$ ln -s /path/to/dir /path/to/symlink
```

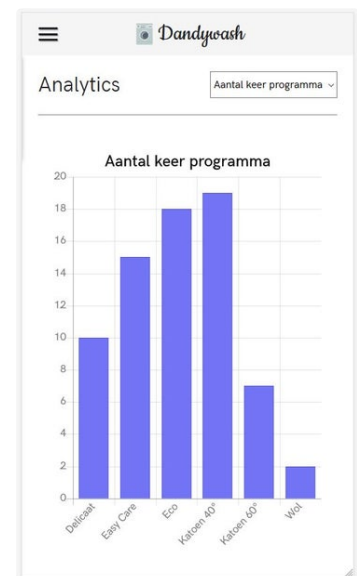
Applied to this use case, the command should look something like this

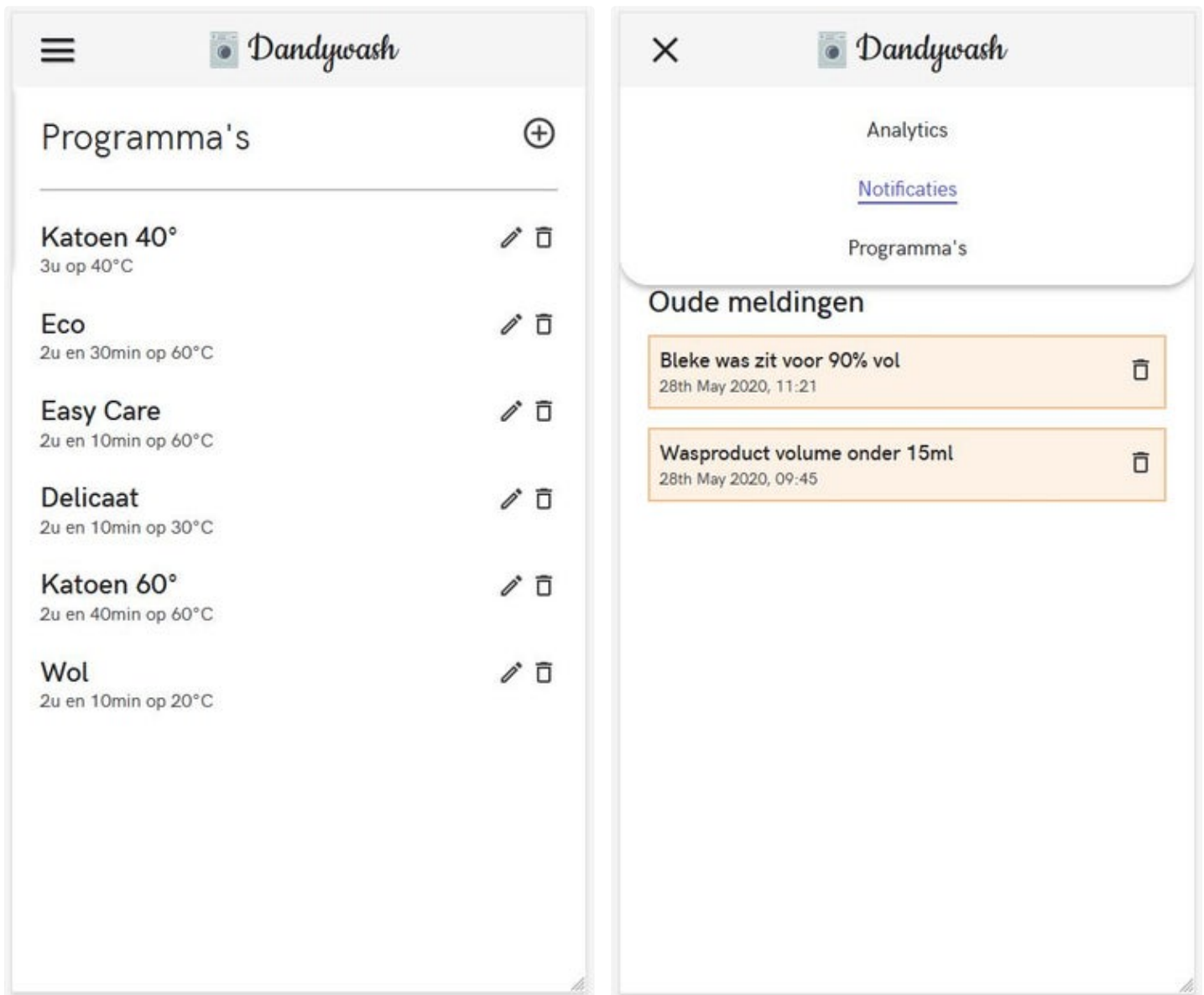
```
$ ln -s ~/home/pi/project1/git-repo/ /var/www/html
```

Now, if all went well, you can browse to <http://169.254.10.1/Frontend> should see the index.html from the git repo.

In this folder you will find the complete responsive frontend code. Including HTML5, CSS and JavaScript.







Step 4: Backend

For this project, we will be using Flask in combination with Socketio. This allows us to set up a flexible webserver with routing and websockets. This Flask app will also interact with the Database in order to perform CRUD actions. The best thing about this whole stack, is that it takes very little time and effort to set up. Firstly, make sure the following third party Python packages are installed. These should be included in the image, but by running the following commands you can make sure / update to newer versions.

```
$ pip3 install mysql-connector-python
$ pip3 install flask-socketio
$ pip3 install flask-cors
$ pip3 install gevent
$ pip3 install gevent-websocket
```

You should now be able to run the app.py script without any issues. It could be that you get an attributeError saying type object 'Database' has no attribute 'cursor'. This is caused by a mistake in the config.py file. Make sure the username password, and name of the database are correct and have access to the database we just imported. This

is especially noteworthy incase you changed the default username and password in MySQL.



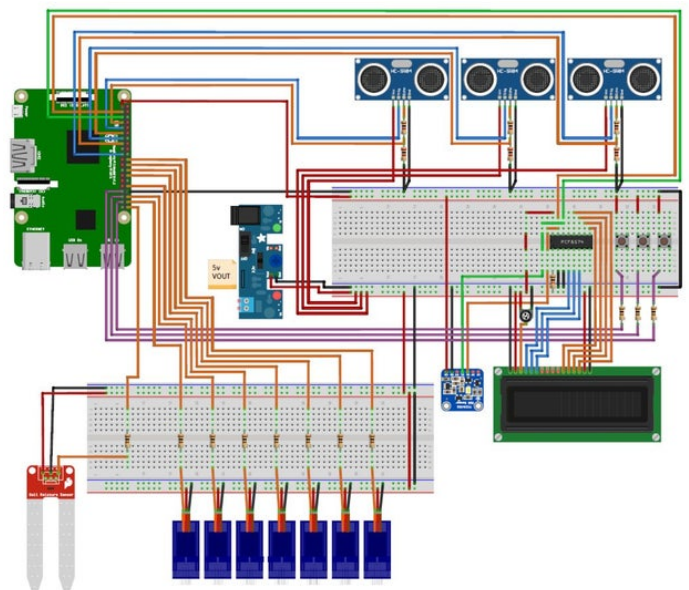
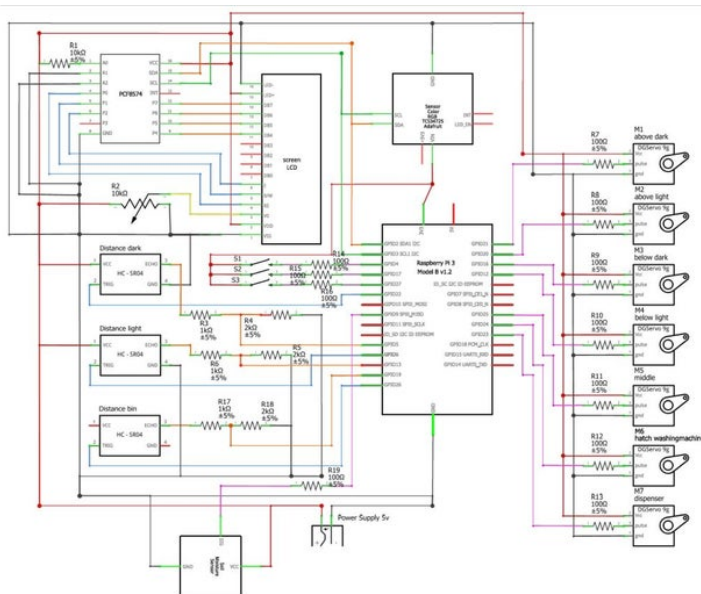
Step 5: Circuit

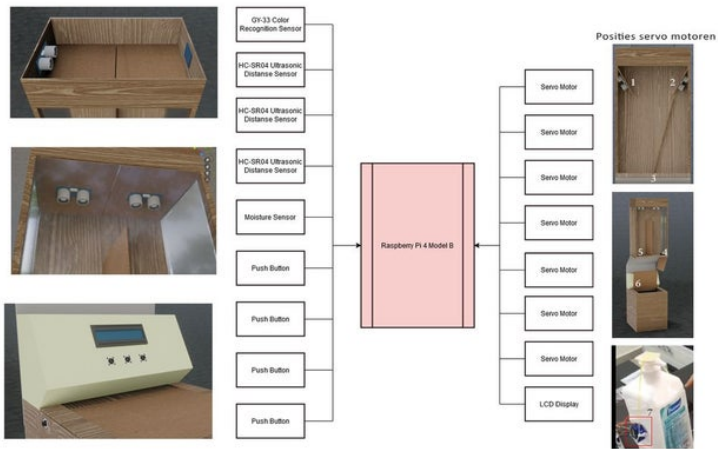
I can't really say much about the circuit. You'll just have to build this and run the test scripts in the git repo. I created a testing script for each sensor and actuator in the circuit, so you can test each part / component individually.



It could be that you will need to change the pin numbers in the code. I have also added a lot of safety resistors in the circuit. Feel free to take those out if you feel extra confident. If you are running low on resistors I recommend you pick up [this kit](#), it's very convenient to always have the resistors you need, labelled clearly.

If the circuit scares you at all, please do not get discouraged. Try to break it down in sections. Build the buttons out first, make sure it works, and then move on to the next sensor. This is something you can't just build in 1 go, unless you're astonishingly talented.

Finally, note that the Raspberry Pi is not suitable for any serious software PWM. Linux is not a real-time operating system. This means you will have slight jittering in the servo motors. GPIO pin 18 does support hardware pwm, but we need more than just the 1 pin.





	https://www.instructabl...	Download
	https://www.instructabl...	Download

Step 6: Case

I had a whole design planned out in my head, which could not be realised because of the current pandemic. Of course this is a situation that demands flexibility from everyone, and that is exactly how I reacted. I still have the original 3D scene that I made, and I will share this [here](#) as well, if you would like to build the case that way. However, for the rest of this article, I will discuss how the case was alternatively built.

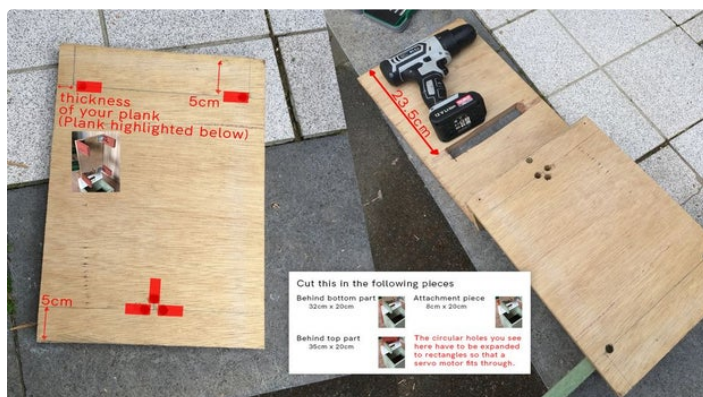
The main inconvenience was the abs plate I was going to use to mount the top portion to the bottom portion. This was the perfect material. Aesthetically pleasing and very practical. This could, however, not be realised, so I had to find an alternative. Since I couldn't think of another material of the same strength that could be bent in the same way, I decided to replace it by a wooden lookalike. This made the rounded curves impossible, but actually created another flat surface that could be used to store items like laundry products or clothespins. I ended up using it to store a second breadboard, making my circuiting life a lot easier for this prototype.

Note the rectangular hole that was drilled in the back. This allows for cables to be routed to the Raspberry Pi.

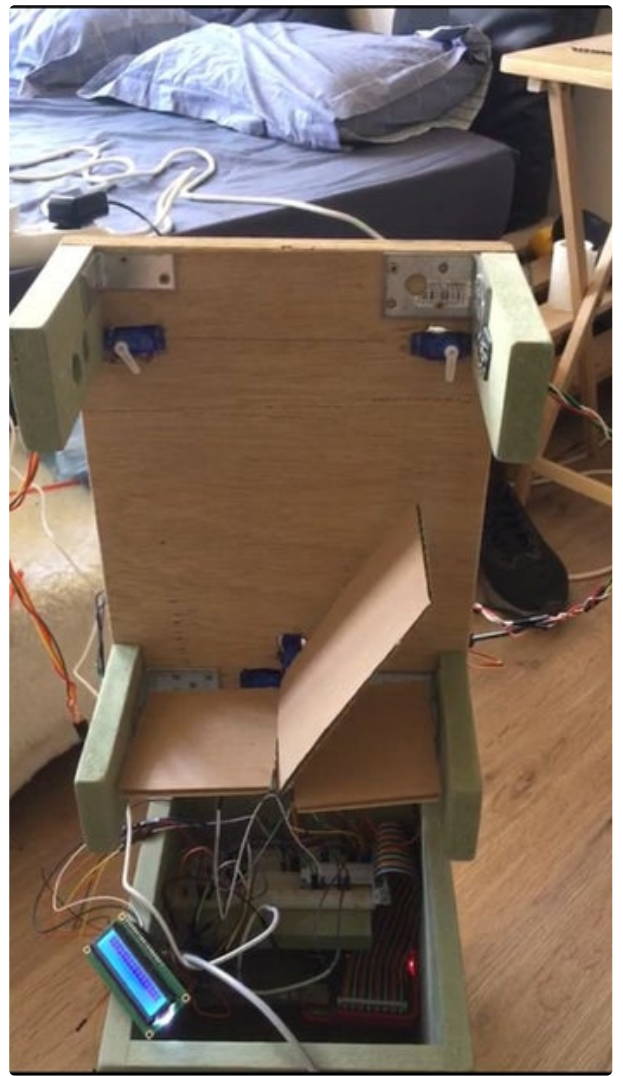
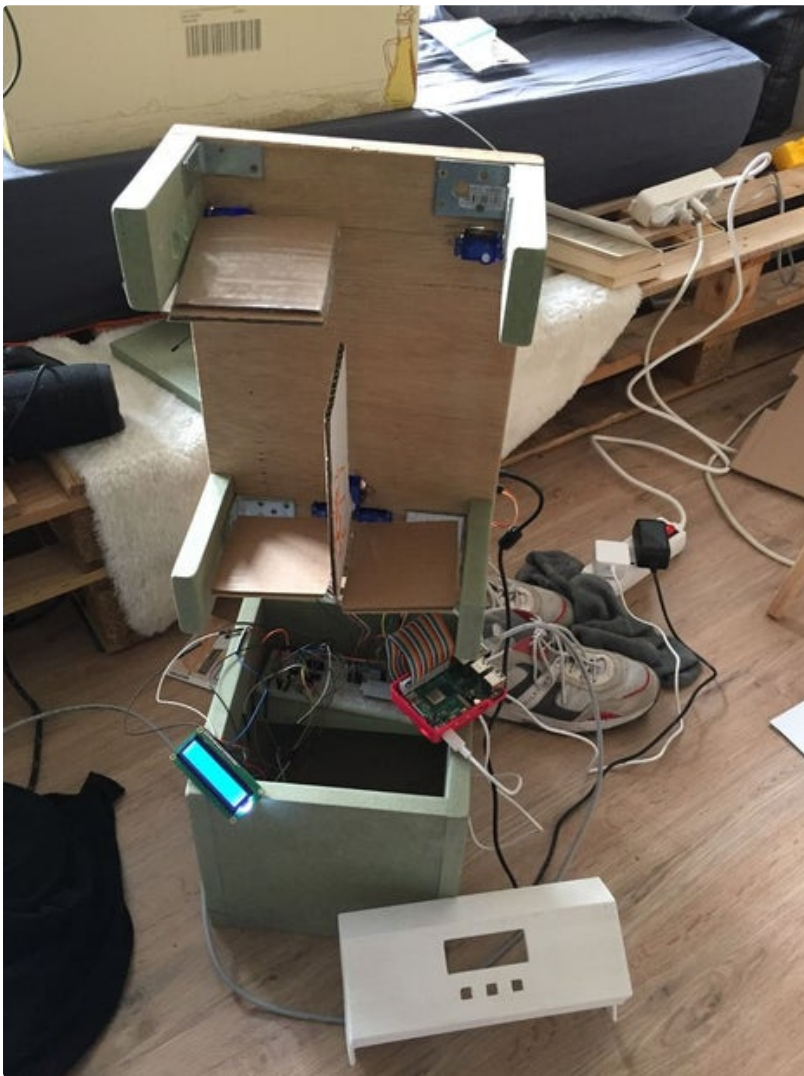
For the planks I paid a visit to my local diy store. They always have some scrap wood laying around and are willing to cut it into pieces for a small price. I paid a total of €5 in total. Huge shoutout to Louis from Hubo Wevelgem to make this possible. Afterwards it was just a matter of drilling holes and screwing everything in place. A detailed overview of where to cut and where to drill can be found [here](#).

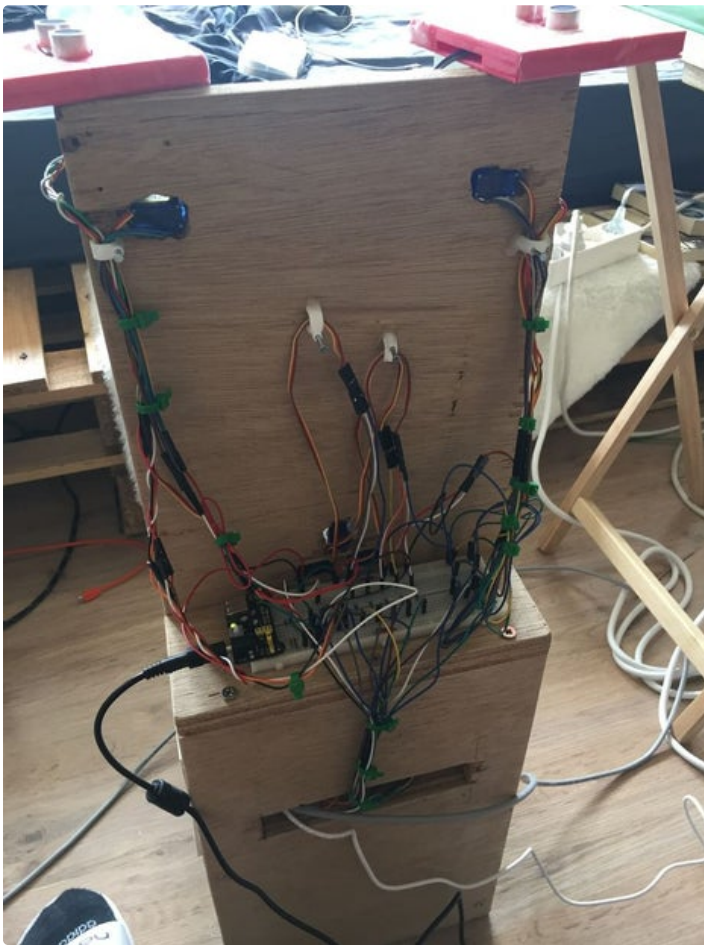
For the 3D Printed bits, I had to rely on the people around me, since school could not provide this service anymore due to the pandemic. Through a friend of a friend I got in contact with someone who was just starting to build his [3D Printing business](#). He was generous enough to print [my main piece](#). The quality was rather churlish due to a misconfiguration of the printer. I bought a [primer spray](#) and gave it 3 coatings, restoring the overall look.

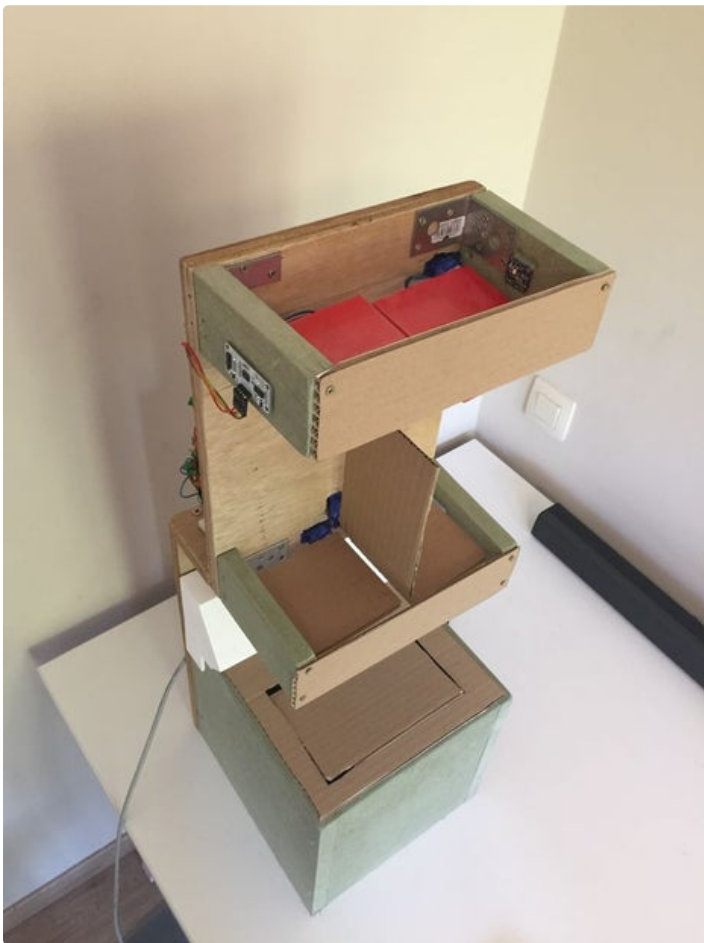
The [distance sensor holders](#) were done by another friend. He also printed the [hatches](#) that were attached to the servo motors. At first I tried this with cardboard, but they wouldn't stick very well. Note that if you 3D Print these bits, you need `bottom_hatch.stl` **twice**, as well as `distanceSensorHolder.stl`. `main_piece.stl` and `middle_hatch.stl` only need to be printed once.

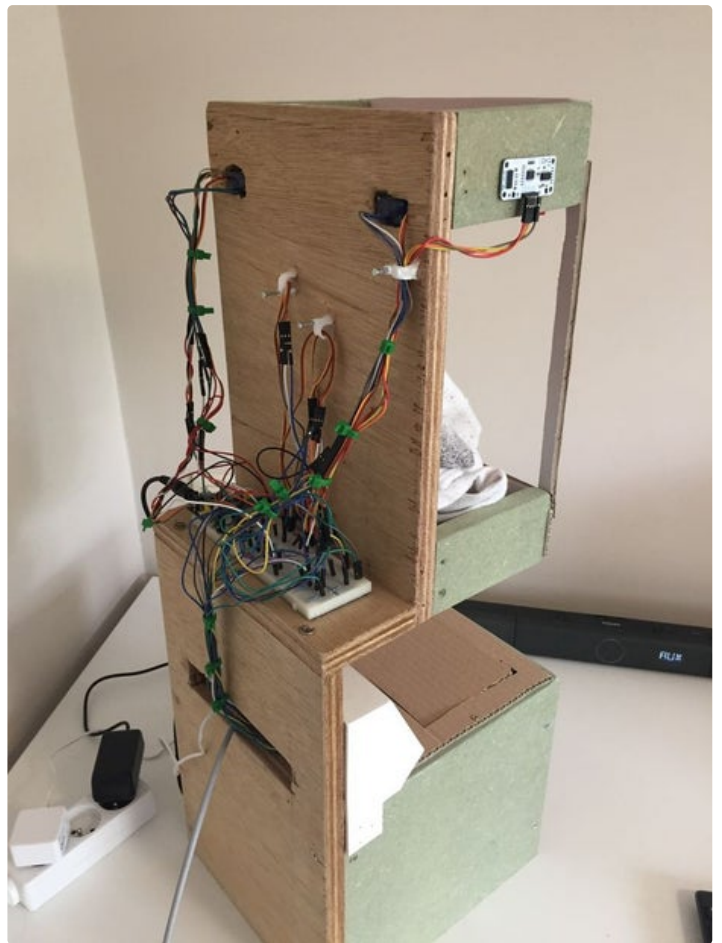














Step 7: Questions?

If any part is not totally clear to you just yet, don't hesitate to reach out and allow me to help you.

Feel free to make contact via email on simonvdhende@outlook.com