Coink Documentation

Release 0.0.1

The Coink contributors.

Aug 01, 2018

Contents

CHAPTER 1

What is Coink?

Coink is an IoT piggy bank.

In the 21st Century piggy banks can be very easily connected to the cloud and become part of the rapidly growing Internet of Things.

Having automatic reports of your savings online is not only very convenient but can also make saving fun for kids! With Coink you can have real-time graphics with your cumulative savings as well as the recent daily savings to track your progress.

Adults may use it as well, not only for fun, but to help them quit smoking, for example, keeping track of the money they save each day. Or in bars and restaurants, to keep track of the tips they get each day.

Of course the magnetic coin detection system can be used elsewhere: in vending machines, arcade game machines… The cloud-connected piggy bank is just an application.

CHAPTER 2

How can I build it?

2.1 Setup

2.1.1 Board

We use an ESP8266-based board, in particular a NodeMCU development kit[3].

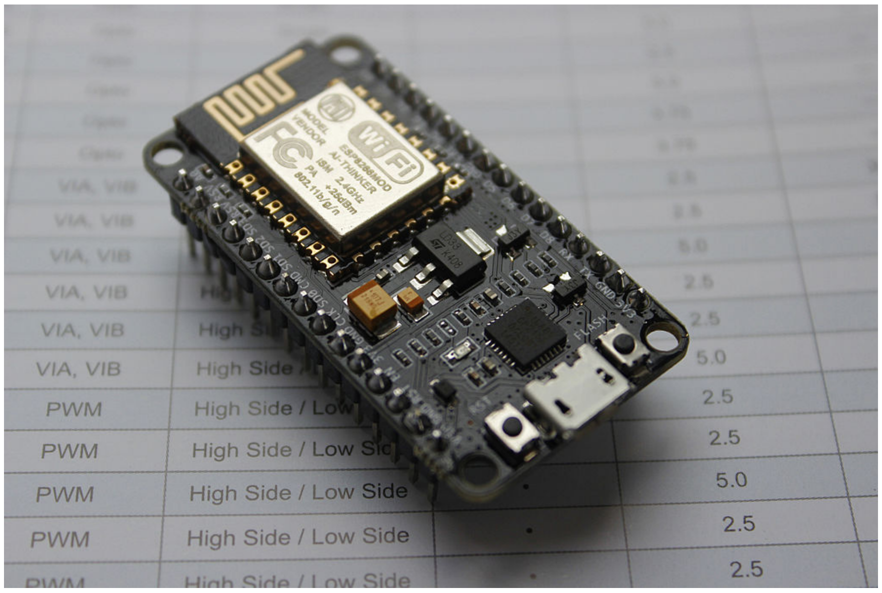


Fig. 1: NodeMCU development kit.

It is a very cheap, but powerful board.

[3] https://github.com/nodemcu/nodemcu-devkit-v1.0

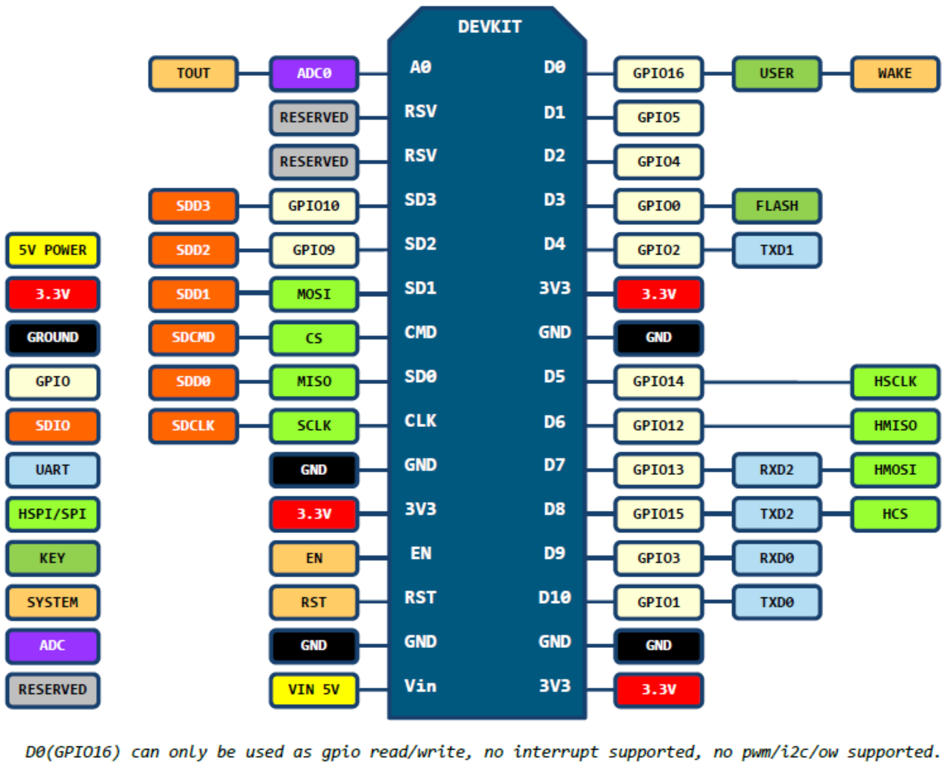


Fig. 2: NodeMCU development kit pinout.

2.1.2 MicroPython

We use the Python programming language or, more accurately, the MicroPython[2] programming language. Not only it is a beautiful language, but also implements an HTTP stack that will ease the process of communicating with a remote server in any IoT project.

In order to start using MicroPython, we need to load the MicroPython firmware onto the ESP8266 board first.

Flashing the firmware

We need to make sure we have the required permissions to flash the device. Either:

```
sudo usermod -a -G dialout $USER
```

Or, to avoid the need to log out of your session:

```
sudo echo 'ATTRS{idVendor}=="1a86", ATTRS{idProduct}=="7523", MODE="0666"'> /etc/udev/
↪rules.d/99-nodemcu.rules
sudo udevadm control --reload-rules
```

Note: You might need to adjust the `idVendor` and `idProduct` depending on what you see with `dmesg` when connecting the device.

We use esptool[1] to reflash the device, which is very easy to install with:

```
pip install --user esptool
```

Then, reconnect the device and erase its flash memory with:

```
esptool.py --port /dev/ttyUSB0 erase_flash
```

Download the latest MicroPython firmware for ESP8266 boards and flash it with:

```
esptool.py --port /dev/ttyUSB0 --baud 460800 write_flash --flash_size=detect 0␣
↪esp8266-20180511-v1.9.4.bin
```

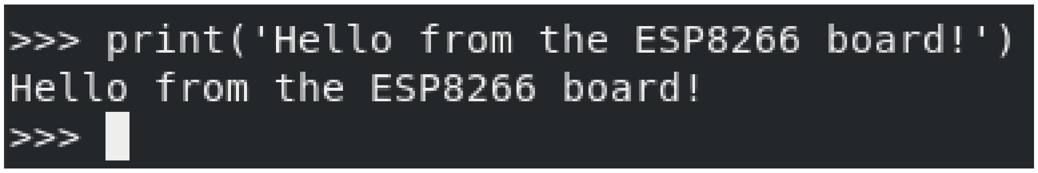
Testing the REPL

Once the MicroPython firmware is on the device we can access the REPL (Python prompt) over serial with a 115000 baudrate:

```
screen /dev/ttyUSB0 115200
```

If everything went well we should see a `>>>` (if not, try to press "enter"). You may now try to execute a simple `print()` call, which will be executed in the board itself:

[2] https://docs.micropython.org/
[1] https://github.com/espressif/esptool/



```
>>> print('Hello from the ESP8266 board!')
Hello from the ESP8266 board!
>>>
```

Fig. 3: MicroPython REPL.

2.1.3 Sensor

The ESP8266 board is connected to the sensor through an I2C bus, in particular, using D1 (GPIO5) for the clock and D2 (GPIO4) for the data transmission. The sensor is powered with the ESP8266 board directly, using its 3V3 and GND pins.

Once the sensor is connected to the board we can check the I2C connection by simply:

```
from machine import I2C
from machine import Pin


i2c = I2C(scl=Pin(5), sda=Pin(4), freq=400000)
devices = i2c.scan()
print(devices)
```

Which should print a non-empty list.

2.1.4 References

2.2 Build

2.2.1 Ramp

Coink is actually very easy to build. First we need the ramp to insert the coins. The ramp is a 3D design that can be very easily printed with any FDM 3D printer (or any other more-advance 3D printer if you have access to it).

The 3D design is very simple but, in turn, we need to do some work before using it. First, we need to glue some plastic sheets to cover the inner parts of the ramp. This way we avoid the coin bumping into any disturbances around the sensor board, where the ramp is not continuous:

Then we need to cover the outter part of the ramp by glueing another plastic sheet. We will glue our magnet (in orange) there, just in front of the sensor board. Any magnet will do, we just need to make sure it is not too powerful, or the coins will get stuck in the ramp!

2.2.2 Box

The piggy bank can be made with a small wooden box. Wood is easy to work with, which is important as we will need to create a hole for the ramp in the box:

Once we have the hole, we can very easily glue the ramp to the box with some epoxy to make sure it does not move.

Even if obvious, we should make sure the magnet and the magnetic sensor board both fit inside the box before glueing the ramp!

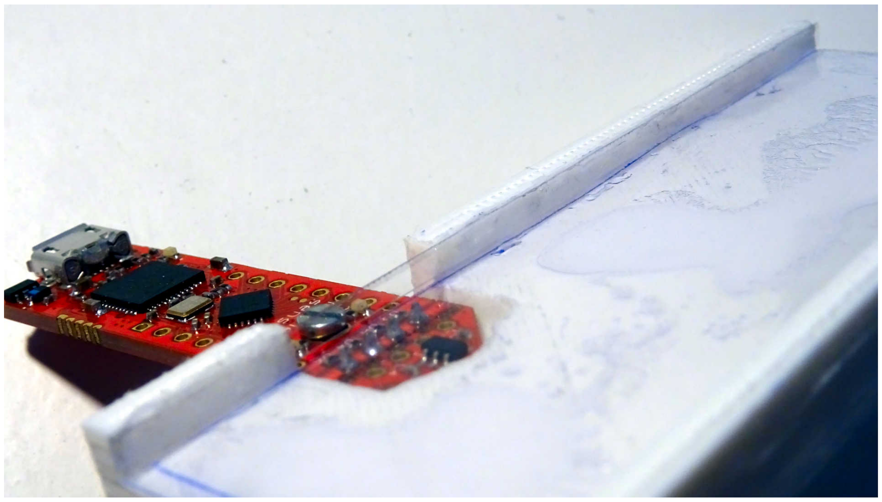


Fig. 4: Plastic sheets covering the inner parts of the ramp.

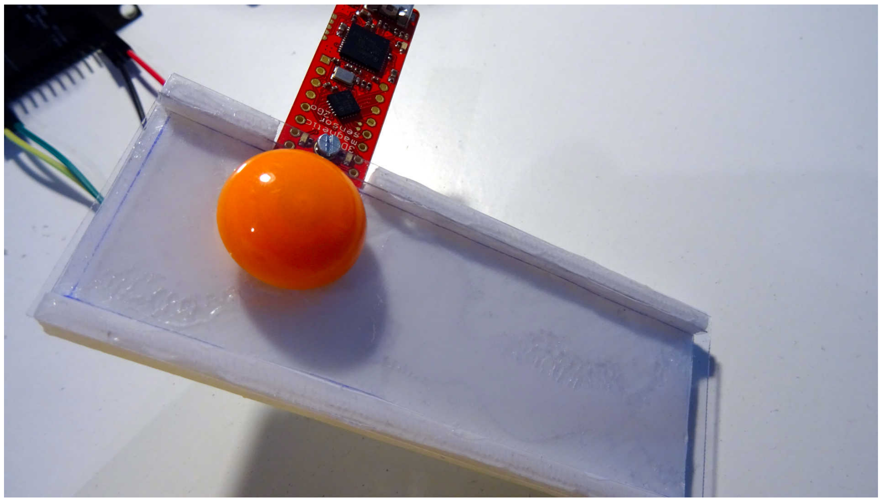


Fig. 5: Plastic sheet covering the outter part of the ramp and magnet attached to it.

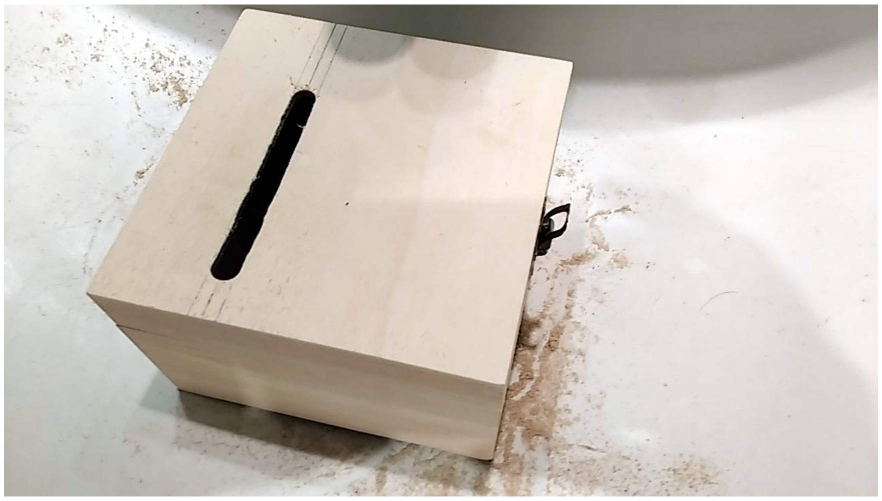


Fig. 6: Wooden box with a hole for the ramp.

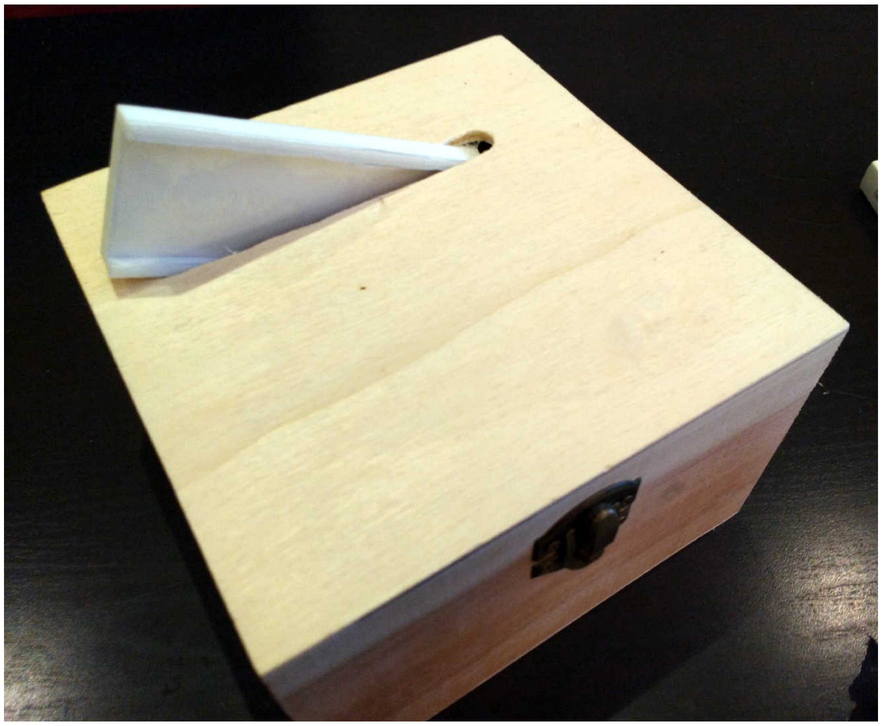


Fig. 7: Ramp glued to the box.

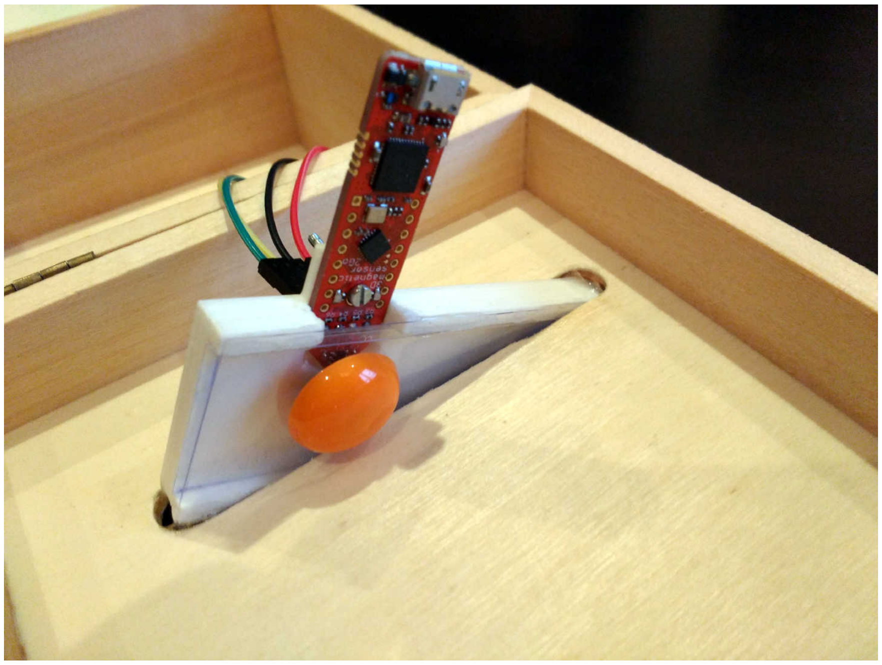


Fig. 8: Detail of the ramp inside the box.

2.2.3 Wiring

Wiring the magnetic sensor board and the NodeMCU boils down to connect only 4 wires:

NodeMCU pin	Magnetic sensor pin
D1 (GPIO5)	SDA (P2.10)
D2 (GPIO4)	SCL (P2.11)
3V3	3V3 (P1.0)
GND	GND

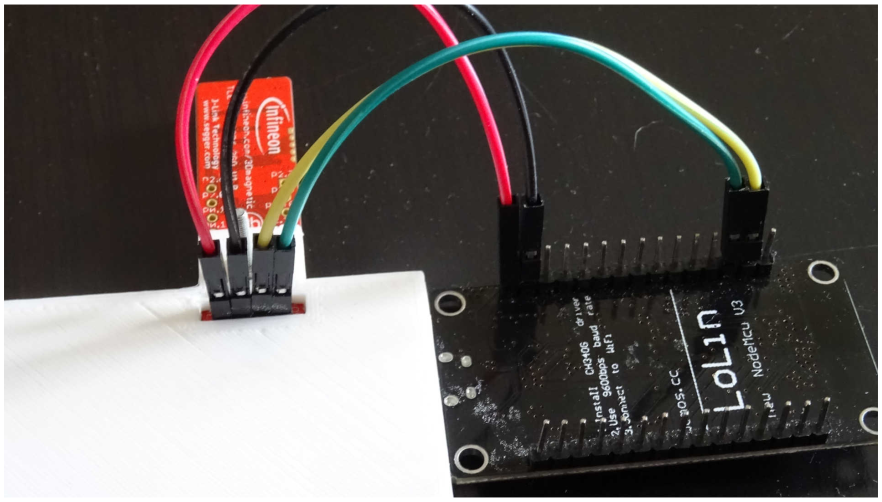


Fig. 9: NodeMCU and magnetic sensor wiring.

2.3 Data

2.3.1 Visualization

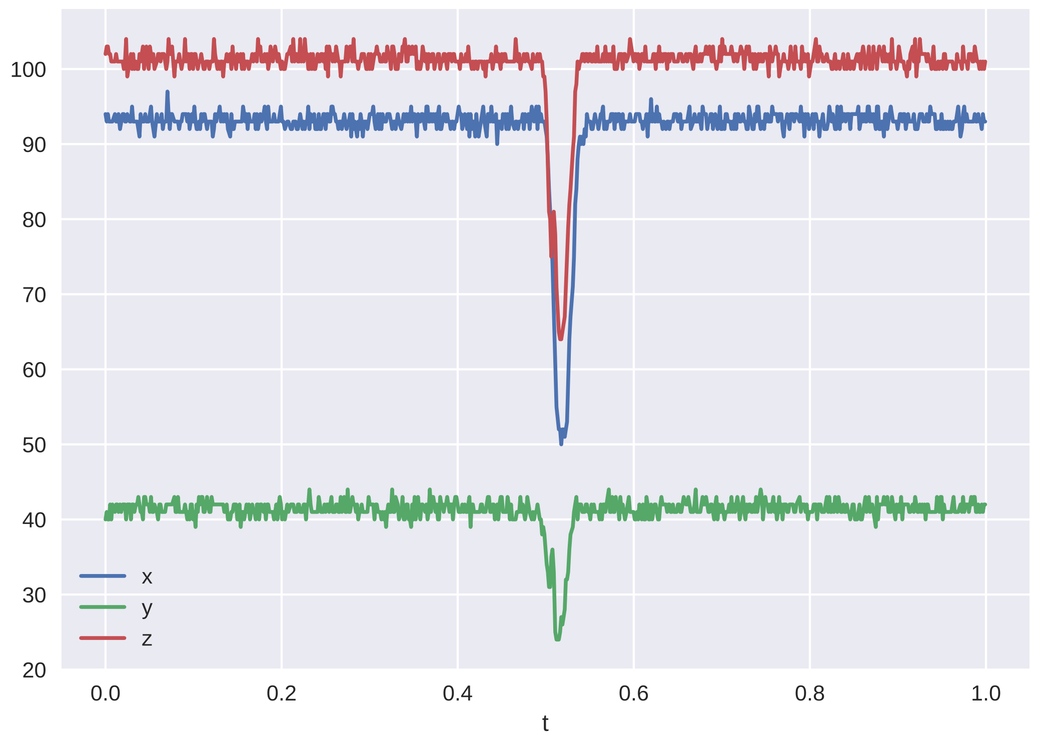
If we visualize the sensor readings during one second, when a coin is passing by, we note that there is little time (around 50 milliseconds) to get useful data:

Each of the axis has a different iddle-state reading, so we can normalize the readings by first dividing by the average iddle-state reading and then substracting 1 to have an iddle-state reading of 0 in all axis and also to have data curves that represent relative variations of the magnetic field instead of absolute.

After doing this normalization we can visualize our region of interest for both 1 € and 2 € coins:

2.3.2 Differences

Having a look at the previous curves we might notice some differences between the 1 € and 2 € curves. Among them we might notice a small difference in the time span. Although that makes sense (the 2 € coin has a bigger diameter than the 1 € coin) and we would expect it to alter the magnetic field for a longer period, this is assuming the speed is the same for both coins. That, of course, is not very reliable, as we could simply insert coins with higher speed when pushing them down the ramp. Therefore, we will ignore differences in the x-axis (time) and focus on the y-axis (magnetic field relative variations).



In order to be able to compare curves and try to differenciate among different coins, we did some simple feature engineering to try to extract some curve features that are relevant and easy to compare/match against known profiles:

min_[axis] Represents the absolute minimum value, for the given axis.

d0_[axis] Represents the maximum positive increase since the last rolling minimum from the iddle state and until the absolute minimum is reached, for the given axis.

l0_[axis] Represents the low value where `d0_[axis]` started, for the given axis.

Although the differences for those engineered features might seem obvious for the curves shown above, we want to make sure that the differences are significant on average for all the data we gathered.

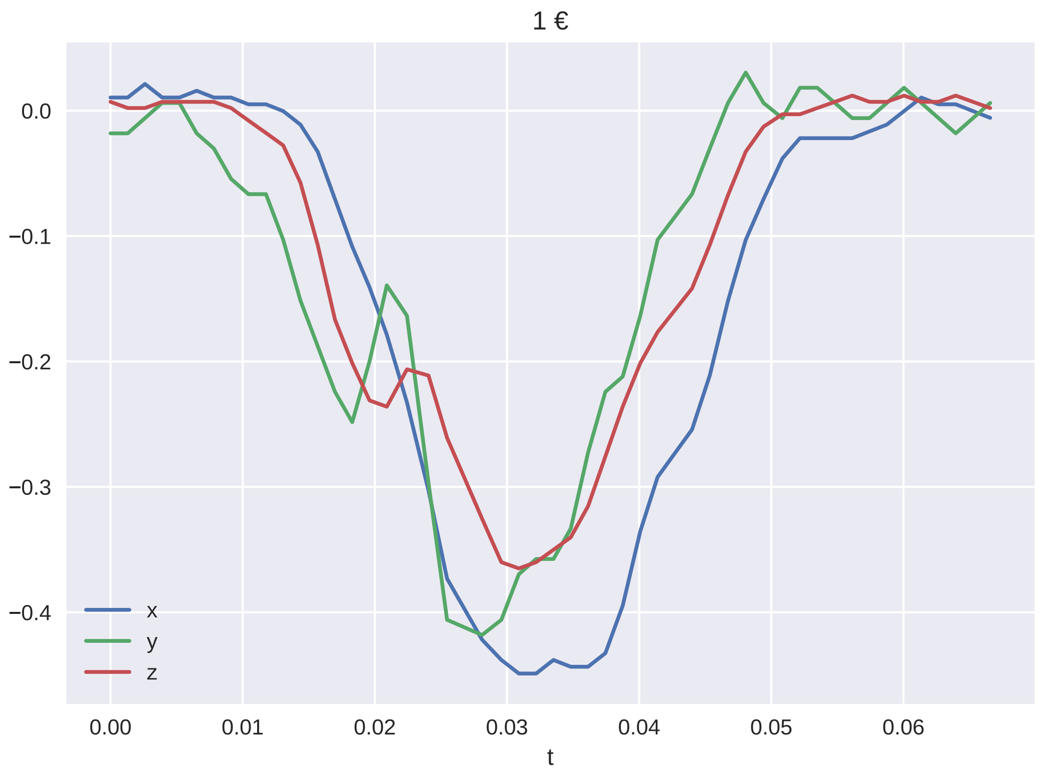
To visualize these differences we can use box plots for all `min`, `l0` and `d0` features, for all axis as well:

We can see there are many significant differences between the two coins. Although we could use only the most significant among them it is better to simply use all features. Matching against the average plus some expected deviation we can not only differenciate between 1 € and 2 € coins, but also detect unexpected (potentially fake coins) readings.

2.4 Cloud integration

2.4.1 Thingspeak

Integrating Coink with an external cloud service like Thingspeak is very straight forward thanks to their public RESTful API. We just need to write a couple of lines of code to process and visualize the data appropriately.



The following code snippet plots the daily savings in the last couple of days:

```
[data, time] = thingSpeakRead(516536, 'Fields', [1], 'NumDays', 7);
data = timetable(time, data)
data = retime(data, 'daily', 'sum')
bar(data.time, data.Variables, 'FaceColor', [1 .5 0])
```

The following code snippet plots the total accumulated savings:

```
[data, time] = thingSpeakRead(516536, 'Fields', [1], 'NumDays', 365);
curve = cumsum(data);
area(time, curve, 'FaceColor', [1 .5 0]);
title(strcat('Total savings: ', num2str(sum(data), '%.2f'), ' €'));
ylabel('€');
```

Note that the code above assumes that the Thingspeak channel was created to ingest a single field, which corresponds to the coin that was detected on insertion. The code then sums by day or calculates the total cumulated sum.

2.5 Indices and tables

- genindex
- search

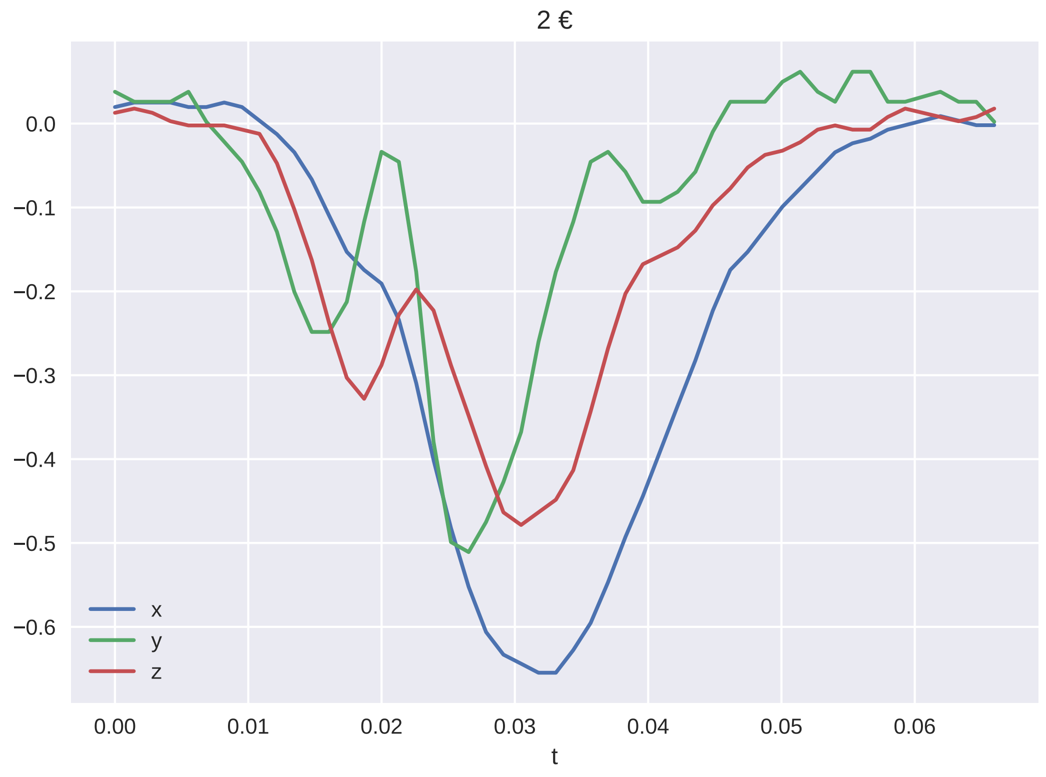
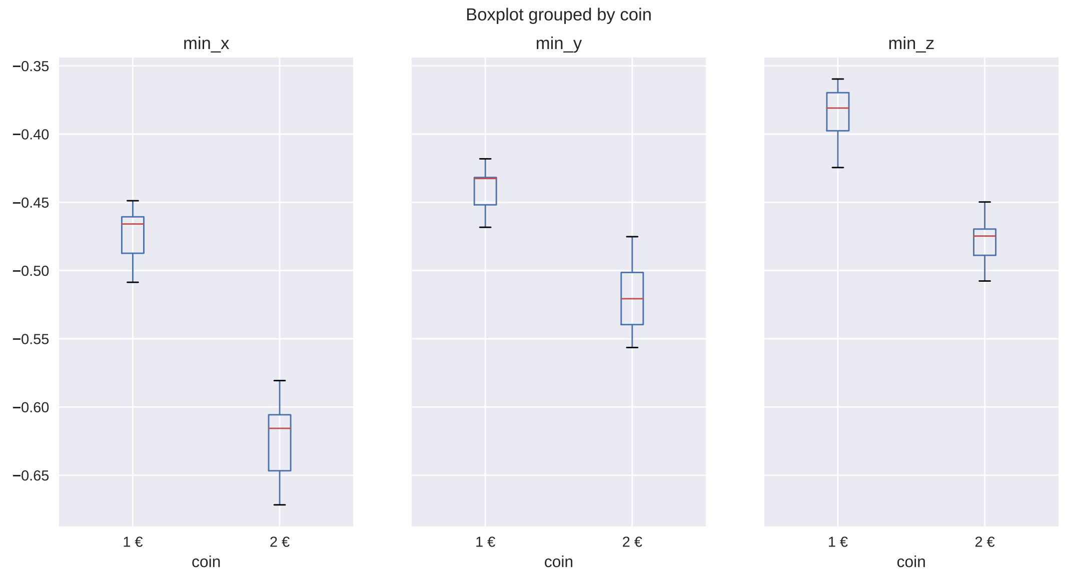
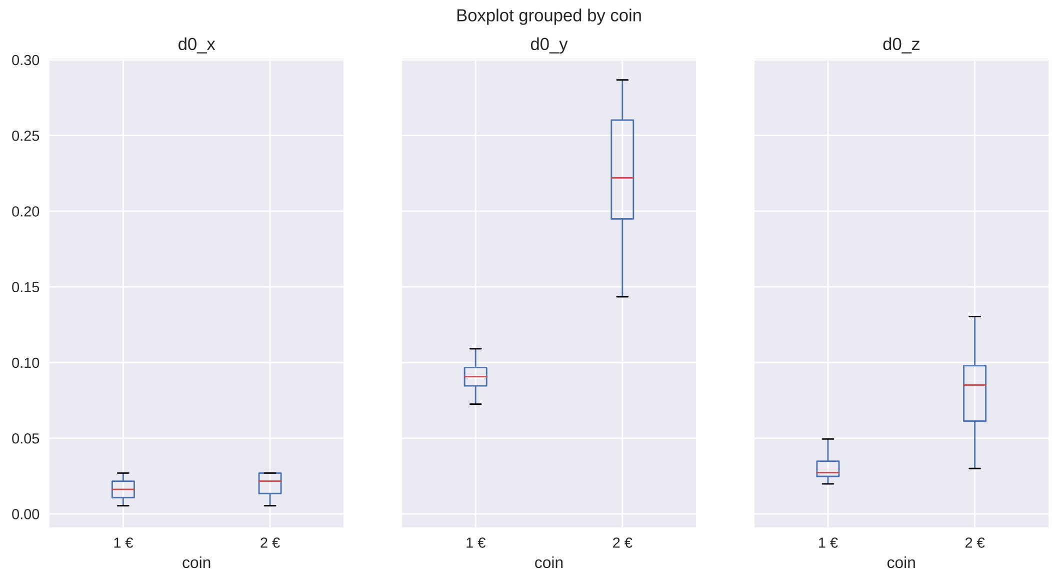
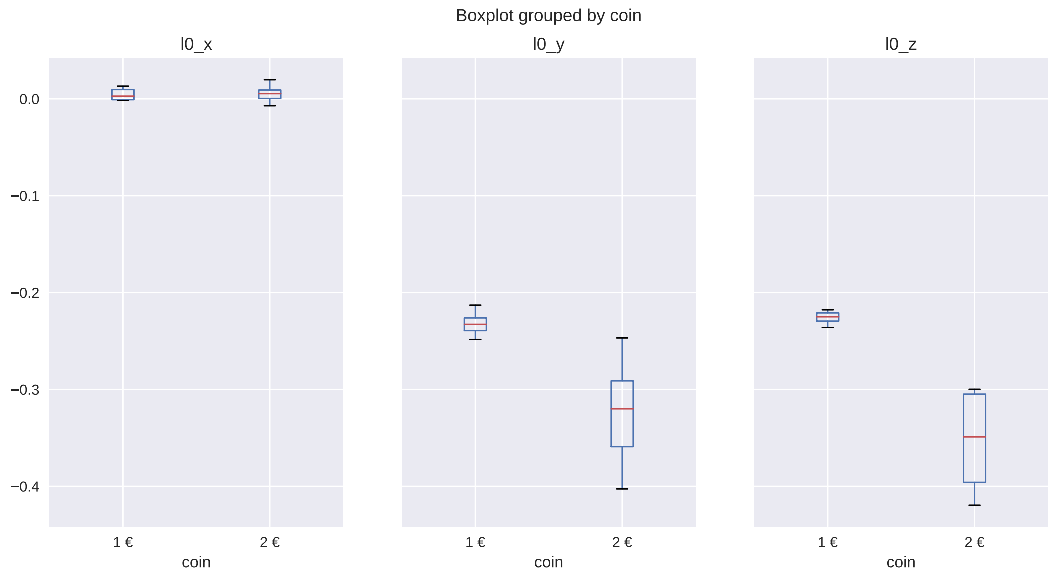
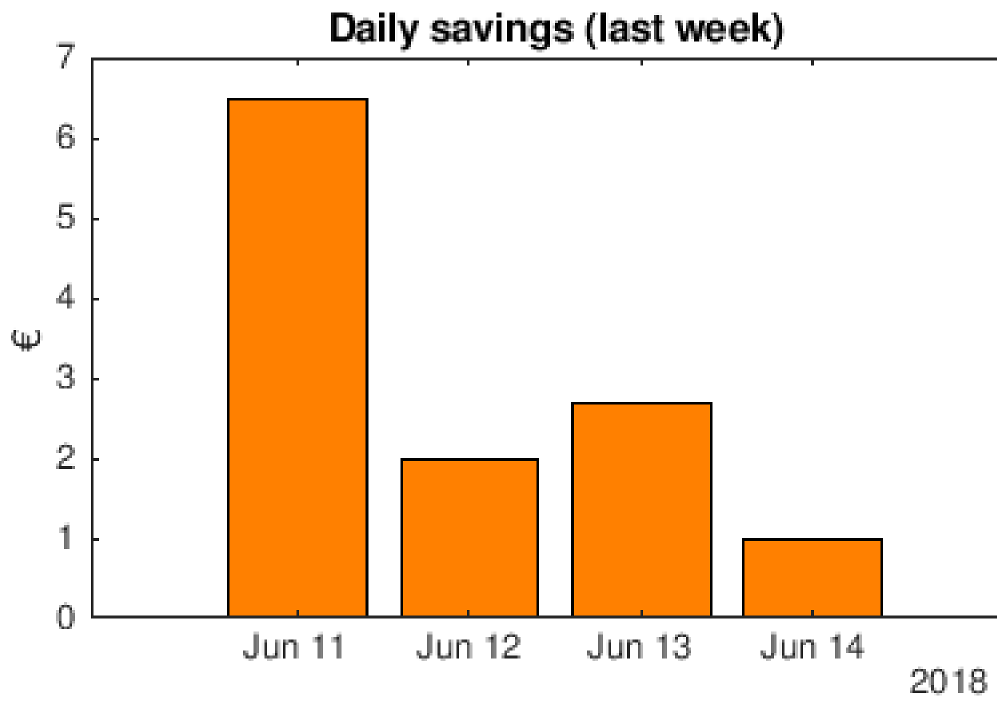


Fig. 10: Engineered features.

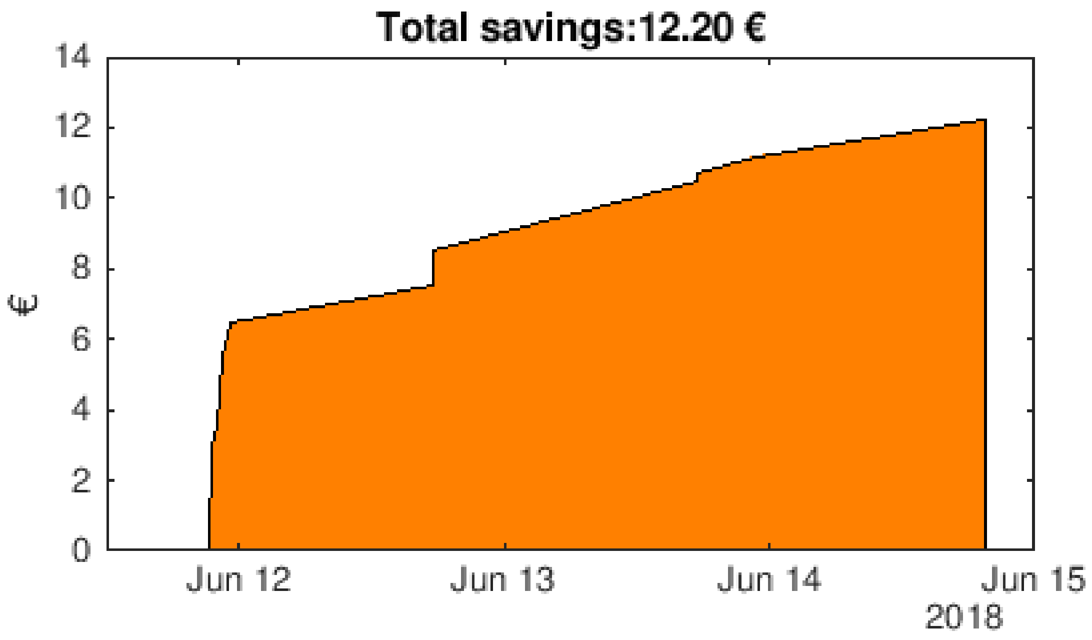



Boxplot grouped by coin
d0_x
d0_y
d0_z
0.30
0.25
0.20
0.15
0.10
0.05
0.00
1 €
2 €
coin


Boxplot grouped by coin
l0_x
l0_y
l0_z
0.0
−0.1
−0.2
−0.3
−0.4
1 €
2 €
coin



Daily savings (last week)
€
0
1
2
3
4
5
6
7
Jun 11
Jun 12
Jun 13
Jun 14
2018



Total savings:12.20 €
€
0
2
4
6
8
10
12
14
Jun 12
Jun 13
Jun 14
Jun 15
2018

Index