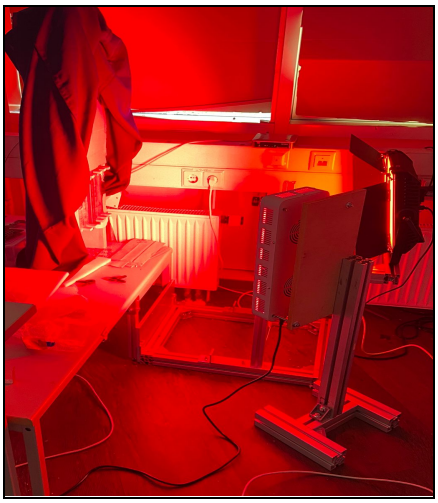
# Study of a Beehive Capturing System

1. **Motivation for the experiment**:

Honey bees (Apis mellifera) are highly social insects known for their vital role in pollination and honey production. They organize themselves into complex colonies with a queen, worker bees, and drones. Worker bees gather nectar from flowers, converting it into honey through enzymatic reactions and dehydration. Honey serves as their primary food source during winter. Bees also play an essential part in pollinating various plants, thereby aiding global food production [ref: honeybee]. The paper "Chronic within-hive video recordings detect altered nursing behaviour and retarded larval development of neonicotinoid treated honey bees" (2020) presents a novel approach to investigate the impact of insecticides on honey bee brood care and larval growth. The authors utilized video cameras to monitor the activities of bees in hives that were exposed to neonicotinoids. Neonicotinoids are a group of five insecticides (four of which are banned in Germany) that get absorbed into plant pollen and nectar, and ultimately ingested by the bee. They discovered that the use of neonicotinoids diminished nursing activity among bees and impeded larval growth. Additionally, there were adverse impacts on colony size and survival. These findings have prompted further investigation into bee behavior, utilizing a variety of sensors. The sensors utilized in our system provide numerous advantages when observing and scrutinizing a bee colony. They can measure several parameters significant to the bee's productivity and health, including weight, humidity, temperature, and bee activity. The data can be collected automatically and continuously without the need for human intervention or disturbing the bees. Moreover, the data is wirelessly transferred to the archiving platform, thereby reducing the need for cables and wires that could interfere with the hive or the environment. The camera setup supplies high-resolution data, facilitating detailed and accurate analysis and visualization of the bee colony. Studying honey bee behavior and health using cameras and sensors is a valuable approach in biological research and was a major motivation for us to develop this system. Sensors were deployed in and around the system to monitor temperature and humidity. The attached weight sensor helped to monitor the hive weight. This provided insights into swarming behavior, and changes in hive population. Together our system enables highly detailed monitoring of the hive parameters. Studying this data should be a valuable approach to understanding honey bee ecology, health, and their interaction with the environment. While observing the bees in our experiment though, we encountered a problem that ultimately led to an early end for the observation.

# 2. System Design

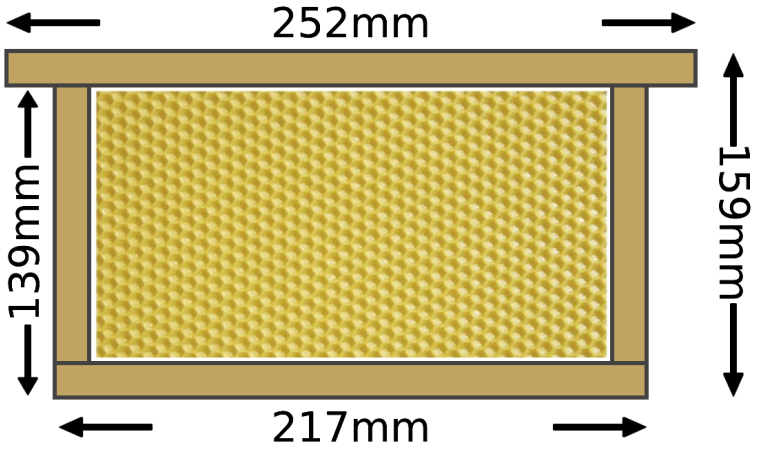


**Old system:** The system was developed to record the behavior of bees in a hive using a high-performance camera and a versatile LED light. The BASLER acA2040 camera can capture images with a 3-megapixel resolution and record videos at a frame rate of 36 frames per second. The MR300W LED light was adjustable in light intensity and wavelength. To prevent disruption to the bees, a red or infrared light source was utilized, as these frequencies are not visible to bees. Therefore the system enabled the observation and analysis of bee activity and interactions in a natural environment. The images and videos were directly recorded by a nearby computer for further processing and storage. The energy consumption was a disadvantage of this system as it was necessary to capture images and videos continuously. Furthermore, monitoring the bees solely via camera resulted in incomplete insights. Thus, a significant overhaul was necessary for the system to yield richer data.

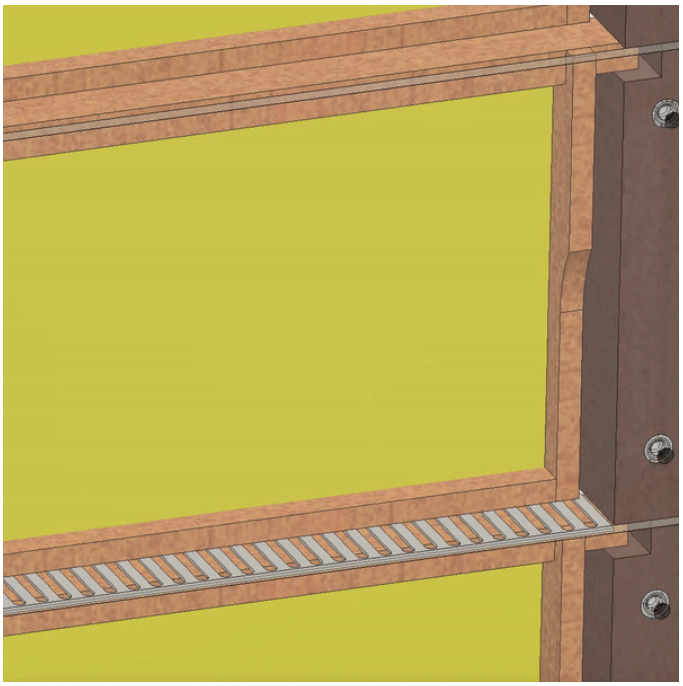
**Improving the system:**

The new system stems from a combination of hardware changes and software upgrades, which, in combination, resulted in a better and more efficient system.
The first decision we made was that the old hive design needed an upgrade. After the duration of the last experiment, the polycarbonate windows were cemented to the wooden frame of the hive by the bees with propolis, a resinous substance they use to seal cracks for better thermal insulation. The old design featured one big window that was supposed to be slid up to access the inside. With the huge surface area glued together by the bees, the window did not move an inch, even when applying serious leverage. Another problem was the inadequate insulation. Bees are persnickety regarding hive temperature, as they need 34.5 - 35.5 °C to keep their brood well-developed. Therefore a well-insulated hive can help alleviate some work, as they need to invest less work into heating or cooling.

After some brainstorming, we concluded that we should use a modular design going forward. Not only did this enable easier manufacturing and faster implementation of changes, but we also got rid of the large, single-viewing window. Instead, we developed small modules that could be stacked on each other. Each module includes a front and back window that can be completely removed by loosening a screw in every corner. To make the hive structurally sound, the modules have connecting dowels and a clamping mechanism to hold everything together. As a sizing reference, we use the widely established “Standard-Mini-Plus-System”. This is a hive format derived from the normal-sized “Dadant-Format”, but exactly half its size. Because of its small form factor and easy handling, it is often used for queen breeding.

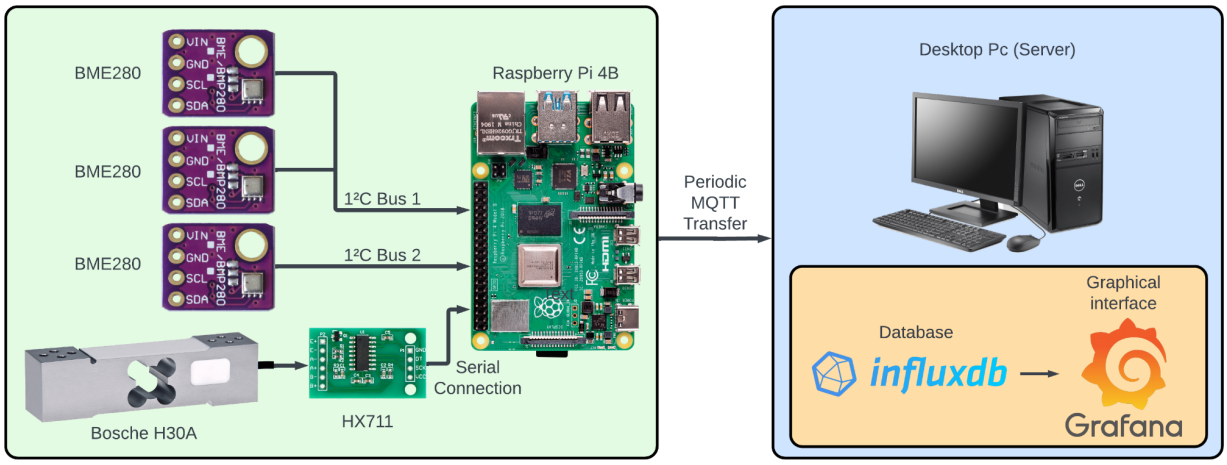



For our mini hive configuration three modules, each housing one Mini-Plus frame, are stacked on top of each other. Every frame can be removed, either completely with the module or by sliding it out to either side.



Another big advantage of the frame format being widely available is that we can move the bees to a new home in winter. We planned this to be a completely off-the-shelf hive located outside because past experiments showed that the bee colonies were simply not able to survive the winter conditions in our lab. One beekeeper suggested that the bees would need the cold outside temperatures, which are not achievable indoors. Another theory suggests that the insulation we used earlier is not sufficiently aiding in maintaining a stable 35°C temperature, which could have led to considerably more energy expenditure or the bees might not have been able to produce enough heat to reach that temperature altogether.

Because the old insulation just consisted of taped-on cardboard, we designed a cover-style encasement out of Polystyrene, a cheap and harder version of styrofoam. This enclosure is a simple box with an open bottom to be put over the whole hive from the top. It is fitted with a circular cutout in the top (to allow for a small sugar-water bottle to be installed for feeding) and a 15 cm by 25 cm viewing window in the lower front to study the brood. While the top opening can be neglected, the front entrance, directly next to a hive window, lets a lot of heat escape and in turn is a few degrees colder than the better-insulated, upper modules. For normal worker activity, this was not a problem and the bees happily went in that section, but the queen didn't lay any eggs in the outer row, touching the Polycarbonate. Our first supposition was that the exposed window was too cold. To heat up this area a bit, we installed an infrared floodlight (originally used for illuminating a property in combination with a night vision camera), because that was the only "invisible" heat source we had at hand at that moment. This worked out great and the little heat it emitted was, at a short distance, exactly the right amount to heat up the viewing window to the same temperature as the better-insulated windows.




Our first change to the measuring setup was made to the capturing unit. The task of capturing sensor data was transferred to a Raspberry Pi 4B. It has more than enough processing power to capture data and relay it to a central server, while also being quite energy efficient. Another selling point is the easily accessible SPI and I2C interfaces, which can directly communicate with most sensors available. The second change happened to our main measuring instrument, the camera. We swapped the Basler for a Raspberry Pi High Quality Camera. The main improvement is that this camera easily connects to our new measurement foundation via a 15-pin ribbon cable at the Pis CSI (Camera Serial In) port. In addition, it boasts a 12.3-megapixel sensor, which is quite a bit more than our last camera's 3-megapixel. Another great perk is that we can reuse the expensive Japanese 16mm lens from the older system, as it uses the same C-mount. The 16mm lens in this setup provides a perfect focal distance to capture the whole viewing window at 1.1 meters distance.

To get a better insight into the hive we equipped the underside of the hive with a Bosche H30A load cell. This load cell in combination with a HX711 A/D-converter provides an accurate reading of the current weight and consequently enables us to draw conclusions about the bee population and honey production. In addition, three BME280s, which can measure temperature, humidity, and air pressure, were spread out around the hive, one in each module.

Software side, we have a Python script running on the Raspberry Pi, which regularly samples sensor data. Every few hours we send the collected data to our storage server using MQTT, which is a lightweight and reliable communication protocol for IoT devices.

On our storage server, we are running a Python script, which receives the MQTT messages and inputs them into InfluxDb, our database. After that, we can visualize the data with Grafana.

We chose InfluxDb, as it is a time-series database that can be used to store data in a structured and efficient way. Time-series databases are optimized for storing and querying data that have timestamps, such as sensor data. InfluxDb allows fast insertion and retrieval of data, as well as aggregation and filtering operations. Grafana is a data visualization tool that is used to display data in graphs and dashboards. Grafana connects to InfluxDb and queries the data using the InfluxQL language. Grafana provides various options for creating and customizing graphs, such as line charts, bar charts, pie charts, gauges, tables, and maps. Grafana also supports alerts and notifications, which certain conditions or thresholds in the data can trigger.

Moreover, the system also uses Wake on LAN to save power on the server side. Wake on LAN enables a computer to be turned on or woken up by a network message. Because of this, our server can stay in sleep mode for most time, which reduces its power usage greatly. The server can be woken up by a wake-on LAN packet (which is addressed to the MAC address of the server's network interface card) sent by the Raspberry Pi. Afterward, the server resumes its normal operation and receives the data from the client (Raspberry Pi). After complete data transfer, the server shuts down.

**Connecting the load cell**

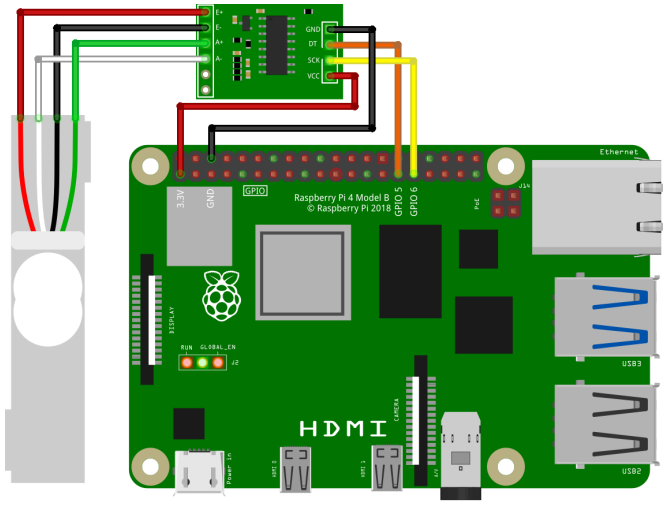
Wiring up the H30A load cell to the A/D converter is straightforward, although color coding can be different on other load cells.

Red to E+ (Input)
Black to E- (Input)
Green to A+ (Output)
White to A- (Output)

The next step would be to wire the HX711 to the Raspberry Pi. For that, we used JST SM connectors to make replacing parts as easy as possible.

VCC to free 3.3V pin
GND to free GND pin
DT to pin 29 (GPIO 5) or any other free GPIO pin
SCK to pin 31 (GPIO 6) or any other free GPIO pin

In the following figure the last two steps are visualized:




To test if everything is working correctly we can use this library. Just clone (or download and unpack) the repository, and run

```
python setup.py install
```

This will install the required GPIO libraries. If you have connected the HX711 to the GPIO ports 5 and 6 you don't have to change anything, if not, you have to adjust the code.

**Calibrating the load cell**

In the Python script, you find the line “hx.set_reference_unit(referenceUnit)”. The readings of the loadcell don't have a Unit. As we most likely want to have grams displayed we need to figure out the ratio of scale units to grams. To do this follow the next steps:

1. Measure the offset:
   Run “hx.read_median(5)” once without anything on the scale. The reading you get is the offset, as you were expecting a 0.
2. Set the offset:
   Run “hx.set_offset(your_offset)”. Now your scale is tared.
3. Calculate the reference unit:
   Place an object of known weight (eg. 1 KG dumbbell) on the scale and run “hx.get_value(5)”. This will output the measured weight of the object but without a unit. To get grams you need to calculate the ratio “R” by dividing them. R=hx711_value/weight_of_object
4. Set the reference unit
   To set the reference unit run “hx.set_reference_unit(R)”. Running “hx.get_weight(5)” should give you the weight in grams now.

**Improving HX711 readings:**

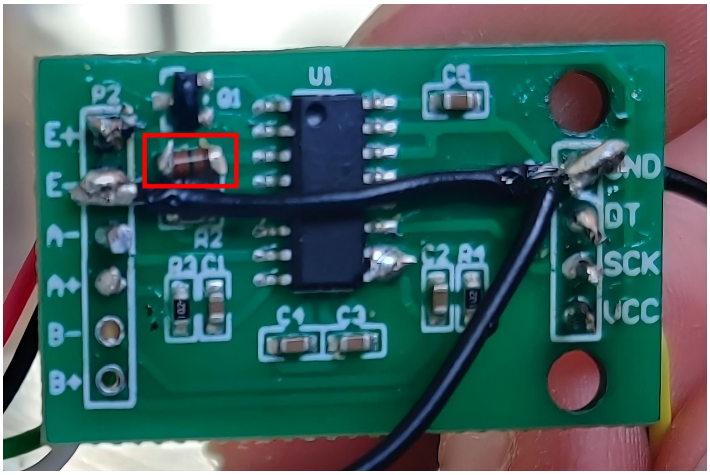
There are many HX711 boards on the market - some are good, and some are quite a bit worse. The grade of the board depends on how the PCB is designed AND equipped, as some manufacturers leave out populating some components in order to save costs. Good HX711 boards feature two channels to connect two load cells: A and B. On some cheap boards, the B channel isn’t equipped and simply does not work. But even with fully equipped boards, the B channel seems to produce quite a bit more noise than the A channel, which is why we recommend against using it altogether. If you want to use more than one load cell in your design, simply attach one HX711 (channel A) to each load cell, as they are cheap and readily available.

*HX711 board with missing B-Channel*

The second thing to watch out for is the resistance between the board's E- and GND pins. In the reference diagram, provided by the manufacturer of the HX711, there should be a direct

connection between these two, but a very low resistance is also fine. If you measure a higher resistance you will have to solder that connection in, as a missing connection here might lead to measuring inaccuracies, drifting over time, and/or a higher temperature dependency.

Even tho the HX711s datasheet lists 3.3V and 5V as suitable voltages, the most common boards you can buy online shouldn't be run at 3.3V, as they use a voltage divider to create a 4.3V excitation voltage, which is recommended for stable load cell readings when the input voltage is 5V. As the Raspberry Pi's GPIO pins only tolerate 3.3V we need to modify the HX711 Boards to make them work well with 3.3V, by lowering the excitation voltage to 3.1V (at least 0.1V lower than the input voltage of 3.3V as per the HX711 datasheet). You can find the voltage divider on the board labeled R1 and R2. To get the appropriate Voltage we either need to replace R1 (normally 20k Ohm) with a 10k Ohm resistor or solder a 22k Ohm resistor in parallel with R1. This gets us to around 2.9V excitement voltage, which is perfect.



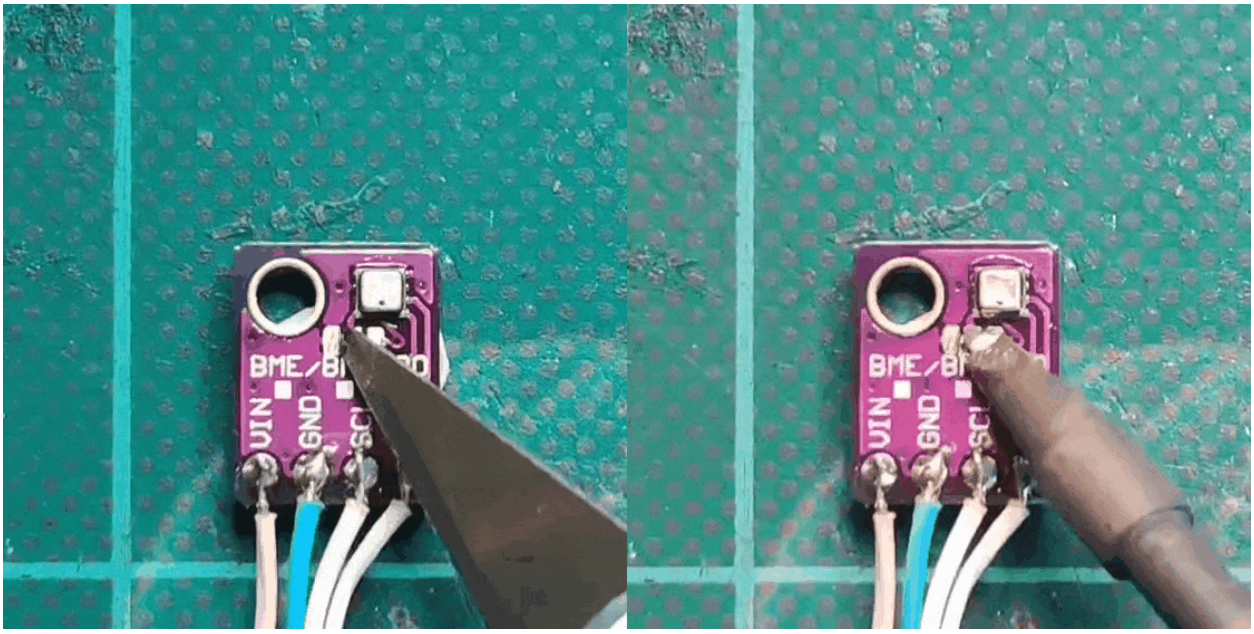

The above picture shows the 22 Ohm resistor (soldered in parallel to R1) in the red box and the black wire connecting E- and GND.

**BME280**

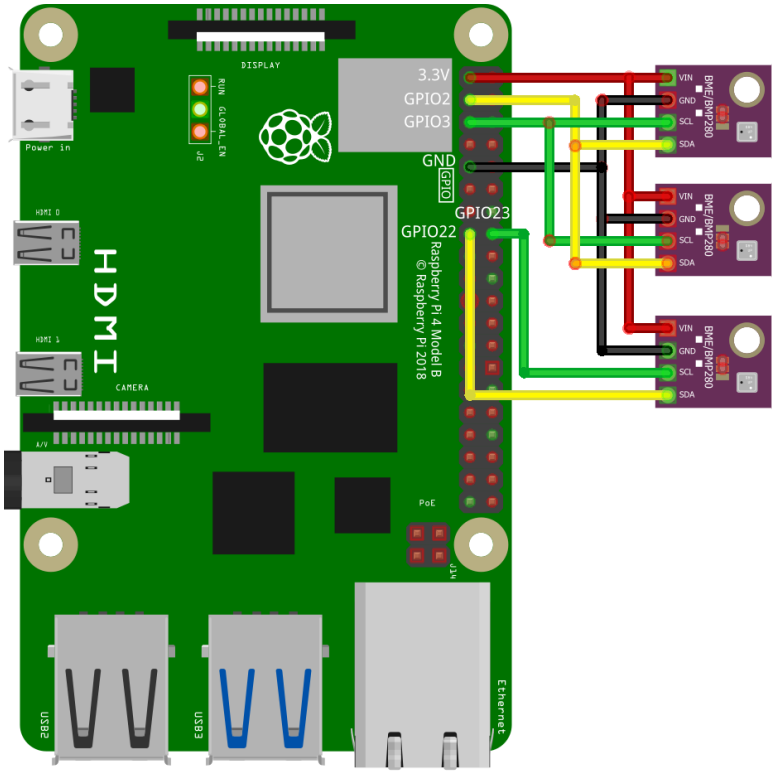
The sensors we use to measure temperature etc. use I²C as their communication protocol. That is nice, as it only uses four shared wires (GND, VCC, Clock, Data) for every device on the bus, which makes it very easy to use. Also addressing is very easy, as each device has a built-in address. This means that you can just send a request to address "**0x76**" for example and you will receive the temperature of that node. This is very fine if you only use one of each sensor, as different sensor types tend to have different addresses programmed. As the 7-bit addresses only allow for 112 unique addresses, not every device manufactured can have a distinct address. The BME boards we use allow you to choose from two possible addresses (0x76 and 0x77) by cutting and bridging some solder pads. As we want to use three temperature sensors we also need to create a second I²C bus on the Raspberry Pi, which we will show you later.

**Connecting the BMEs**

To fit the three sensors on two buses we need to modify one of the BMEs and assign it the address “0x77”. This is easily done by cutting the small trace connecting pad 1 and 2, followed by bridging the pads 2 and 3 with a little blob of solder.



The following diagram shows you how to connect the sensors to the Pi in detail:




**Enabling a second I²C port**

The first step would be to install I²C-Tools by running the following commands on your Pi:

$ sudo apt update
$ sudo apt upgrade
$ sudo apt install i2c-tools

After that, we can use `i2cdetect -l` to list all available I²C interfaces, which, in our case, until now, should only return the interface `i2c-1`. After that run `i2cdetect -y 1` to list all connected devices to port 1. If you wired everything right the output should look something like this:

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: —– -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 77
```

Next add the line "`dtparam=i2c_vc=on`" at the end of the config file at `/boot/config.txt,` which you can open with
sudo nano /boot/config.txt

After this reboot (sudo reboot) you should see the second I²C interface when you run `i2cdetect -l` again. Now i2cdetect -y 0 should also work and display

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: —- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 --
```

**Software prerequisites:**

In the following paragraph, we provide you with a short guide on how to set up the prerequisite software components. Please note that this is a high-level overview, and the specific installation details may vary depending on your operating system and hardware.

- InfluxDB:
  - Install InfluxDB by following the installation instructions for your operating system from the InfluxDB website.

- Grafana:
  - Install Grafana by following the installation instructions for your operating system from the Grafana website.

- Installing the MQTT broker:

  To use MQTT on our Raspberry Pi, we needed to install an MQTT broker. We chose the popular Mosquitto broker that can run on a Raspberry Pi. Setup is straightforward:

- Connect to your Raspberry Pi via SSH. The default login credentials should be
  Username: pi
  Password: raspberry
- After that you need to update the system packages by typing the following command: `sudo apt update && sudo apt upgrade`. This will take some time depending on the internet speed and the number of packages to be updated.
- Then, you can install Mosquitto by typing: `sudo apt install mosquitto mosquitto-clients`.
- After installing Mosquitto, start and enable its service by typing the following commands in the terminal: `sudo systemctl start mosquitto` and `sudo systemctl enable mosquitto`.

Here you can find details about more advanced configuration options.

To test the Mosquitto broker open two terminal windows and run the following commands, one in each window:

- `mosquitto_sub -h localhost -p 1883 -t test -u pi -P raspberry`: This subscribes to a topic called “test” on our local broker using port 1884 and user pi with password raspberry.
- `mosquitto_pub -h localhost -p 1883 -t test -m "Hello world" -u pi -P raspberry`: This publishes a message with the content “Hello world” to the same topic using port 1884 and user pi with password raspberry.

You should see the message appear in the subscriber window.

**InfluxDB Setup:**

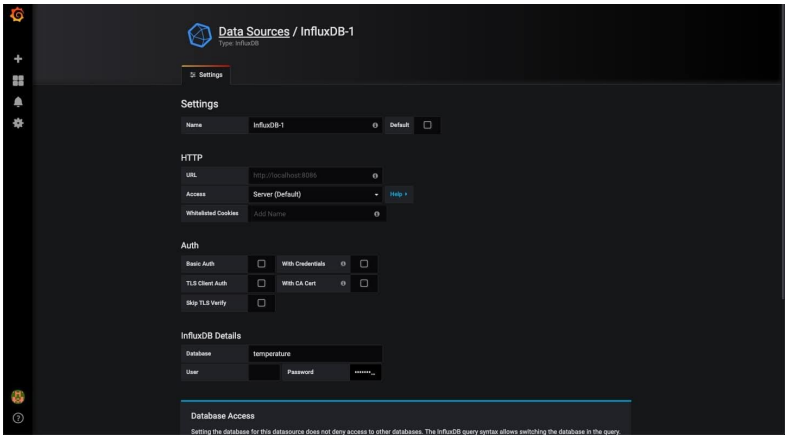
- Create a new InfluxDB database for storing data.
- Configure authentication settings (username and password) as needed.
- Start InfluxDB and ensure it's running.

[ref: How To Visualize Your Data at the Edge With Losant and InfluxDB | Losant Documentation]

**Configuring Grafana:**

- Access the Grafana web interface (usually at http://localhost:3000) using the default username/password (admin/admin) or as configured during installation.
- Add a data source:
- Choose InfluxDB as the data source type.

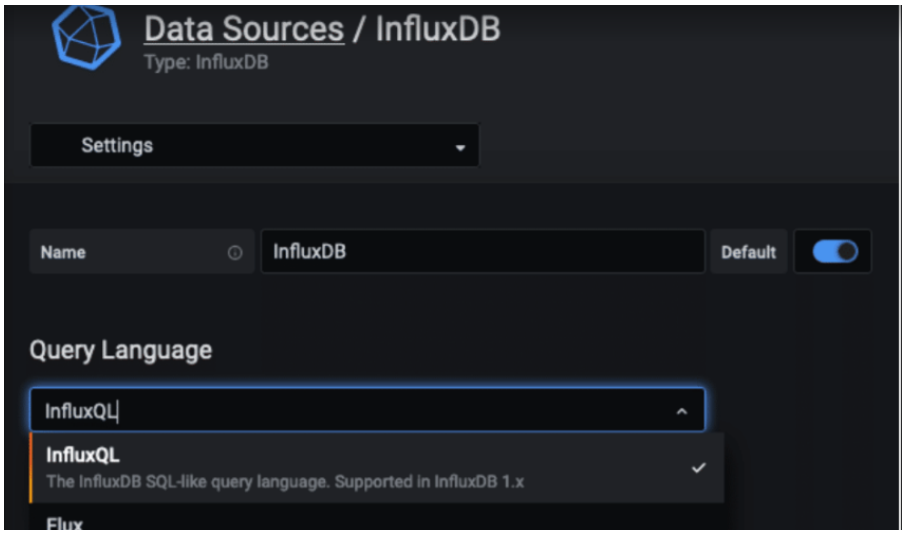
- Enter the InfluxDB database details, including URL, database name, and authentication information.
- Save the data source.




[ref: https://terminalbytes.com/images/2019/02/raspberry-pi-grafana-data-source-influxdb-2.jpg ]

**Create Dashboards in Grafana:**

- To visualize your Influxdb data create a dashboard in Grafana.
- Use the InfluxDB data source you configured earlier to query and display data in Grafana panels.




[ref: How to Build Grafana Dashboards with InfluxDB, Flux, and InfluxQL | InfluxData ]

**Python script running on the Raspberry Pi:**

The Python script obviously starts with the imports of the used libraries. In addition to some system libraries like time, io, and os the most important imports were paho (mqtt client), wakeonlan, InfluxDBClient, Picamera2, bme280, and hx711. We will show you how we used each of those in the following section.

```
#MQTT Address/Port
ADDRESS="localhost"
PORT = 1883

#Influx credentials
influxdb_client = InfluxDBClient(url='http://141.99.144.112:8086',
                                 token='y-NOpMwuT6Z8dEuOW_q2VSya7KXipjnEXvSq62uM_nC
                                 org='BeeOrganisation')
write_api = influxdb_client.write_api()

#define BME Configuration
bus1 = smbus2.SMBus(1) # Thermal, 1 BME
bus2 = smbus2.SMBus(2) # 2 BME
calibration_params1 = bme280.load_calibration_params(bus1, 0x76)
calibration_params2 = bme280.load_calibration_params(bus2, 0x76)
calibration_params3 = bme280.load_calibration_params(bus2, 0x77)

#Setup Scale
hx = HX711(5, 6)
hx.set_reading_format("MSB", "MSB")
hx.set_reference_unit(47.85062680375406)
hx.set_offset(328796.33333333)

#Setup camera
picam2 = Picamera2()
still_config = picam2.create_still_configuration()
video_config = picam2.create_video_configuration()
picam2.configure(still_config)
picam2.options["quality"] = 95 #for jpeg
#picam2.options["compress_level"] = 9 #for png
picam2.start()
encoder = H264Encoder(bitrate=10000000)

#Setup MQTT
client =mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect(ADDRESS,PORT)
client.loop_start()
```

In the beginning, we define and configure all necessary components that we need to use afterward. Two important things to note are the configuration of the hx711 and the Picam. Regarding the HX711 it is important to set the offset and reference unit according to the

calculations earlier in this article. The interesting thing about the Picam configuration is that we define two different modes, one for images and one for taking videos.

```
def captureSensors():
    global data
    currentTime = datetime.utcnow()

    #one sample per sensor
    data1 = bme280.sample(bus1, 0x76, calibration_params1)
    data2 = bme280.sample(bus2, 0x76, calibration_params2)
    data3 = bme280.sample(bus2, 0x77, calibration_params3)
    averageWeight=hx.get_weight(20)

    #append each sample to data list
    data.append(
        {
            "measurement": "temperature",
            "tags": {
                "hive": 1
            },
            "fields": {
                "temperature1": data1.temperature,
                "temperature2": data2.temperature,
                "temperature3": data3.temperature
            },
            "time": currentTime
        }
    )
```

In this code snippet, you can see how we get the sensor data from the BME and store it in the global variable “data”. In exactly the same fashion we append the data of the other sensors (other BME and weight). One thing to note is the line

averageWeight=hx.get_weight(20)

With “get_weight(20)” the function takes 20 measurements, sorts the list by value, and removes the first and last elements, as they are likely outliers. After that, the function averages the remaining values and then returns the cleaned-up result.

```
157    def captureVideo():
158        global capturing
159        global capturingVideo
160        print("starting recording")
161        while(capturing): #if camera is in use wait 1 second and try again
162            time.sleep(0.1)
163            print("waiting for camera")
164        #lock camera to this recording
165        capturing = True
166        capturingVideo = True
167        #switch to video mode
168        picam2.switch_mode(video_config)
169        picam2.stop()
170        dt = datetime.now()
171        picam2.start_recording(encoder, "images/"+str(dt)+".264")
172        print("started recording")
173        #record x seconds
174        time.sleep(1800)
175        picam2.stop_recording()
176        print("stopped recording")
177        #switch mode again
178        picam2.start()
179        picam2.switch_mode(still_config)
180        #release camera
181        capturingVideo =False
182        capturing = False
```

The function “captureVideo()” handles, as the name implies, video capturing. In line 161 we open a while loop, which continuously checks if the capturing is true. We use “capturing” as a mutex to guarantee that we are only using the camera in one process at a time. Therefore, once we leave the while loop, “captureVideo()” sets capturing to “True” itself to acquire the camera. Also, we acquire the mutex “capturingVideo” which delays an upload to the server until the recording is complete. After that we set the capturing mode to video, store the current time in “dt” (we use that later to name the recording) and start recording. After 30 minutes we stop the recording and switch to photo mode again. We also give up camera control again by setting capturing to false and enable uploads by letting go of capturing video.

```
188  def captureImage():
189      print("captureimage")
190      global capturing
191      if(capturing): #skip taking a photo if currently recording a video
192          return
193      #lock camera
194      capturing = True
195      print("capturing image")
196      dt = datetime.now()
197      imageName = str(dt)+".jpg"
198      picam2.capture_file('images/'+imageName)
199      print("captured image: "+imageName)
200      capturing = False
```

Capturing images is way simpler, as it only takes a fraction of a second. Here we also don't need to prevent an upload, as they have a very similar time frame and in the worst case, we lose one in 1000 images. Again we use the current timestamp to give the image a distinct name.

```python
def sendData():
    global data
    global online
    global capturingVideo
    print("sending data now")
    #wake up pc
    send_magic_packet('F0.2F.74.F9.51.AE')
    #wait for online == 1
    while (online == 0):
        print("online = "+ str(online))
        time.sleep(1)
    time.sleep(10)
    #wait for the video capturing to complete
    while(capturingVideo):
        time.sleep(1)
        print("waiting to send")
    #transfer all captured files
    for filename in os.listdir("images"):
        print(filename)
        fi=open('images/'+filename, "rb")
        if filename.endswith('.jpg'):
            fileContent = fi.read()
            byteArr = bytearray(fileContent)
            client.publish("Image", bytearray(filename,'utf-8')+byteArr)
        else: #transfer videos in chunks
            Chunksize=200_000_000
            while True:
                fileContent = fi.read(Chunksize)
                if not fileContent:
                    break
                byteArr = bytearray(fileContent)
                client.publish("Image", bytearray(filename,'utf-8')+byteArr)
        fi.close()
        #delete sent files
        os.remove('images/'+filename)
        print("deleted: "+'images/'+filename)
    write_api.write(bucket="BeeBucket", record= data)
    print("transfered Data")
    data = []
    time.sleep(10)
    client.publish("Image", 0) #turn off command
```

The last function we want to explain is “sendData()”. This function is also the largest and the most complicated one. The very first step is sending a MagicPacket to wake up our server over Ethernet. Once the server is up and running it will send a MQTT message to the pi, changing “online” to 1. After that we stall until a recording, which might have been running, is complete. Following that we open the images folder (which contains all images and videos from that

recording session) using the os library. Sending the images is straightforward, as their size is well within the MQTT package specs. Sending our videos is a little bit more complicated, as they are much larger. Because of this we read the video files in small, 200MB chunks and send them one by one, together with the original file name. That way the parts can be stitched back together on the server side. Once the whole video has been read and sent, the file is closed and deleted from the Raspberry Pi.
Next we write the contents of data (which now includes all sensor values since the last transmission) to the InfluxDB and reset the “data” list again.
Lastly, after a few seconds, we send a random MQTT message to the server which it will interpret as the signal to shut down again.

```
255    schedule.every(10).seconds.do(run_threaded, captureSensors)
256    schedule.every(7).hours.do(run_threaded, sendData)
257    schedule.every(6).hours.do(run_threaded, captureVideo)
258    schedule.every(20).seconds.do(run_threaded, captureImage)
259
260    while True:
261        schedule.run_pending()
262        time.sleep(1)
```

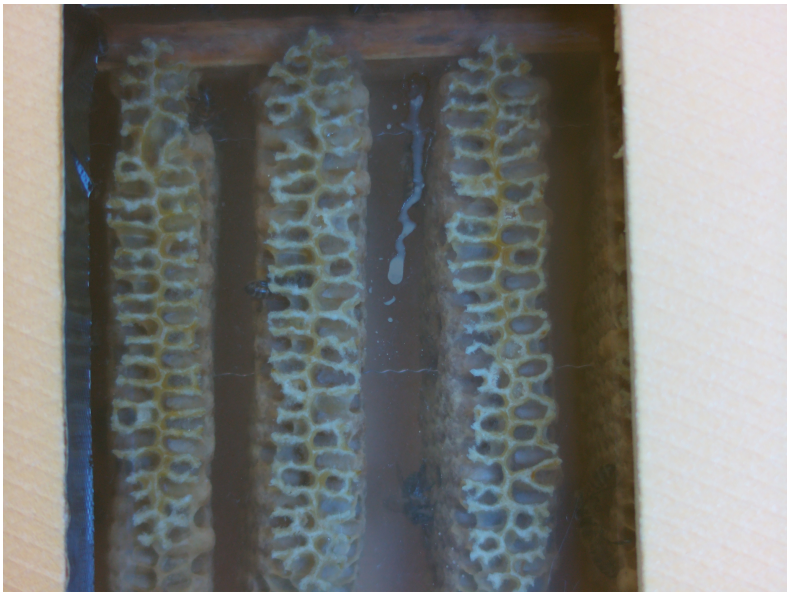
Using the scheduler library we can run our functions at set time intervals. Also, every function gets executed in its own thread, which allows parallel execution.

## 3. **Experiment**

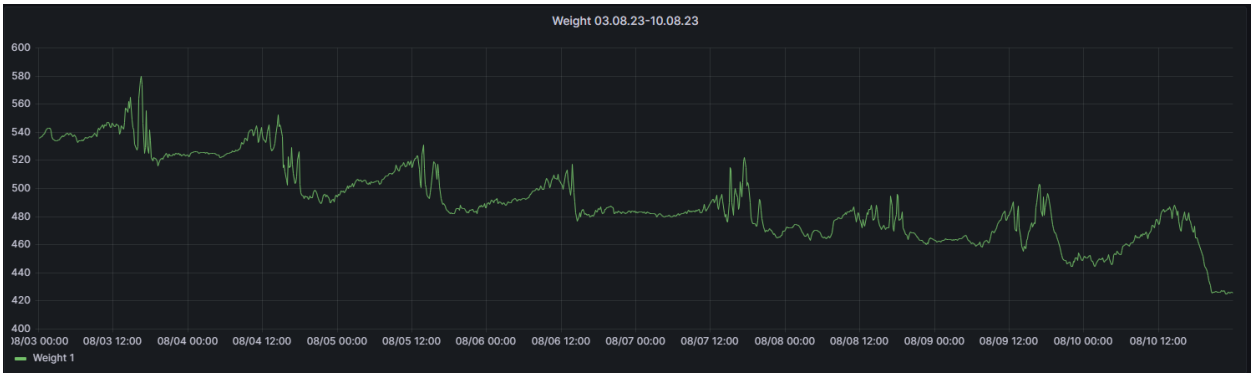
In our experimental setup, we introduced bees into our newly upgraded bee-hive system, which was a significant improvement over the previous version. These bees were provided by our dedicated beekeeper and were integrated into our state-of-the-art system. The bees were healthy and very calm and did not cause any issues when they were being brought in by the beekeeper. The fact that none of the people had to wear any protective gear during the process, says it all. To get the bees into their new home we attached a small edge to the lowest module, equipped with a makeshift roof to create a dark “cave” for the bees to go into by themselves.

Once the bees moved into the hive, they were allowed to move freely within the 3 modules plus they could use a connecting pipe to explore the outside. Right away, the bees started to build up the wax middle walls that we glued to the lowest frame, which

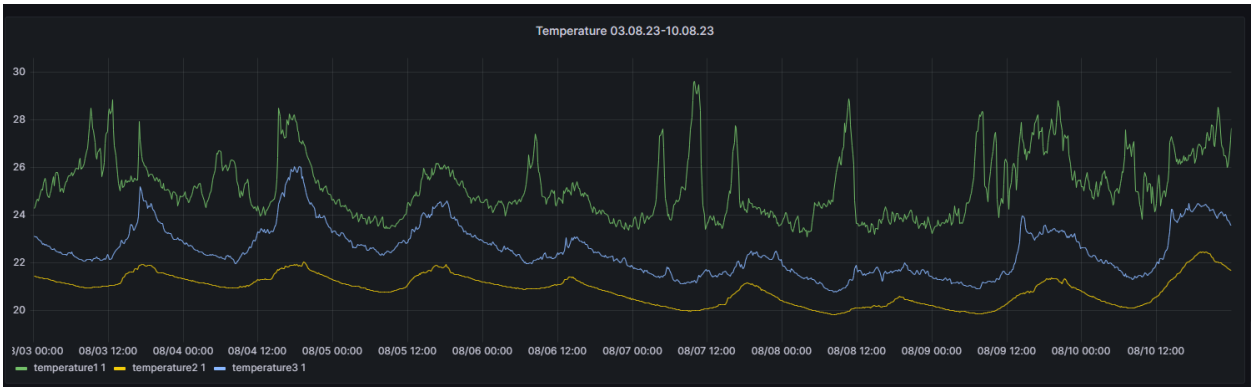
after a few days already looked like this:
As you can notice, we attached these starter plates at a 90° angle to (hopefully) get a direct look into the cell. Meanwhile, we were developing a robust back-end infrastructure to capture and step-by-step attach and connect more sensors to the bee colony. After some time the queen also started laying eggs within the area we observed with the camera. Upon closer inspection, we noticed that the queen didn't lay a single egg in a cell touching the glass. Instead, she always left exactly one cell empty, which was very disappointing, as those were the exact cells we were the most interested in observing. After some measurements with a thermometer we concluded that the glass in the viewing window got too cold compared to the rest of the hive (it was approximately 3 degrees colder). Because of that we installed the IR-Spot we talked about earlier, which worked out great in normalizing the temperatures. The second challenge we faced came as a shock. One morning we got to the BeeLab to work on the sensors and we noticed that around 80% of the bees were gone! We checked thoroughly to find the queen, but she was nowhere to be found. After reviewing the footage we learned that the whole process only lasted five minutes, from the 30th of June 17:02 until 17:07. The bees had swarmed and had probably found themselves a new home without us noticing in time. The sad thing is, that we didn't have our sensors set up at that time yet (apart from the camera). Still, we have some very nice footage to show:
YouTube
How many of you have noticed the mysterious blob on the window, right as the bees left? Here it is again for everybody who missed it:



We still have no clue what that could have been. Maybe you have an idea? Leave them in the comments below!




When looking at the weight data (of course we only have data after most of the bees left) we can clearly notice a day and night cycle, which is probably from the bees leaving the hive together and then coming back spread out over a larger time period.




Another interesting graph we can look at is the temperature one. We saw a day and night pattern again but also noticed large deltas between the 3 different sensors. Also, the sensor represented by the green line seems to be capturing very noisy data.

In total, we accumulated a substantial dataset consisting of **915 GB** in images and videos totaling 310.000 files. We hope that this data can be a rich source of information to further understand bee behavior and hive dynamics. Once the next bee year starts in spring we hope to get the hive populated again to get the first real data with an intact colony!

## 4. Conclusion

In conclusion, we built a system as a combination of hardware and software components that collected and displayed data from the beehive. The data was carefully captured and stored over a period of time to enable a better understanding of the behavior of the bees. Although our relationship with the bees did not last as long as we hoped for, we still got some valuable insight with respect to the reaction of the bees in different environmental conditions like temperature and humidity, which were captured by the sensors alongside visual data. The pictures and videos of the bees were captured by a high-quality camera placed outside the system. Lastly, the collected and observed data was planned to be handed over to the biologists for further research with respect to the behavior of the bees. At the end we would like to leave you with 30 minutes of relaxing bee footage: YouTube