D3.2 Big Data Analytics

CTT2.0 Carbon Track and Trace Deliverable

Lasse Engbo Christiansen, Tobias Straarup Andreasen, Xiufeng Liu, Per Sieverts Nielsen (DTU)

Corresponding author: Lasse Engbo Christiansen

Lyngby, Denmark| 23 January 2017

climate-kic.org

Climate-KIC is supported by the EIT, a body of the European Union

Contents

1. Preface

1.1. About LoCaL

This report was written through support from Low Carbon City Lab (LoCaL). LoCaL aims to reduce 1Gt of CO2 and mobilize €25 billion of climate finance for cities annually by 2050. It is an innovation platform aiming to provide cities with better tools for assessing greenhouse gas emissions, planning, investing and evaluating progress. Started in 2015, LoCaL is a growing community of more than 20 organisations dedicated to unlocking climate finance for cities. This report was realized as part of the project Closing the Gap through Transformative LoCaL Action (CGTLA) under LoCaL.. LoCaL is a Climate-KIC flagship programme.

http://local.climate-kic.org. Contact: victor.gancel@climate-kic.org

1.2 About Climate KIC

Climate-KIC is the EU's largest public private partnership addressing climate change through innovation to build a zero carbon economy. We address climate change across four priority themes: urban areas, land use, production systems, climate metrics and finance. Education is at the heart of these themes to inspire and empower the next generation of climate leaders. We run programmes for students, start-ups and innovators across Europe via centres in major cities, convening a community of the best people and organisations. Our approach starts with improving the way people live in cities. Our focus on industry creates the products required for a better living environment, and we look to optimise land use to produce the food people need. Climate-KIC is supported by the European Institute of Innovation and Technology (EIT), a body of the European Union.

1.3 About Carbon Track and Trace

The Carbon Track and Trace (CTT) project is intended to provide cities with real-time greenhouse gas (GHG) measurement capability. Traditional methods of building and maintaining municipal GHG emission inventories are expensive, time-consuming, and are of questionable utility for mitigation decision and planning support processes. CTT couples low-cost, open source sensors to a Big Data analytics platform that provides cities and regions with a unique capacity to directly measure the impacts of their policy and planning decisions and to develop a semi-autonomous system for building, maintaining, and reporting their annual GHG emissions.

2. Collecting and presenting data

2.1 Intro

One of the goals in the CTT 2.0 project was to provide real-time data from the IoT sensors to different user groups including decision makers and the public. This part will provide an overview of the data flow from the sensors through the cloud and to databases. Collection of data from other sources will also be discussed. Furthermore, a flexible visualization tool is presented.

2.2. CTT Dashboards

Figure 2.2.1 illustrates the architecture of the CTT dashboard system, which consists of the components of data sources, time series database and visualization. In this project, the data sources include the GHG data collected by the sensors installed on the poles in Vejle. The GHG sensor data is transmitted through the IoT network, accessed by the program written in Python, and saved to the time series database, OpenTSDB. The other data source is the traffic data of Vejle that is collected by Here.com. The two types of time series data are streamed into the time series database, OpenTSDB. The OpenTSDB manages a time series as metric added into its underlying database management system, HBASE. The CTT dashboard uses Apache Zeppelin as its visualization engine, which displays the time series from OpenTSDB through the program written in Javascript. The charts in each Zeppelin paragraph are published through the link embedded into a web page using an iFrame. Wordpress is used for the demonstration in CTT dashboard system.

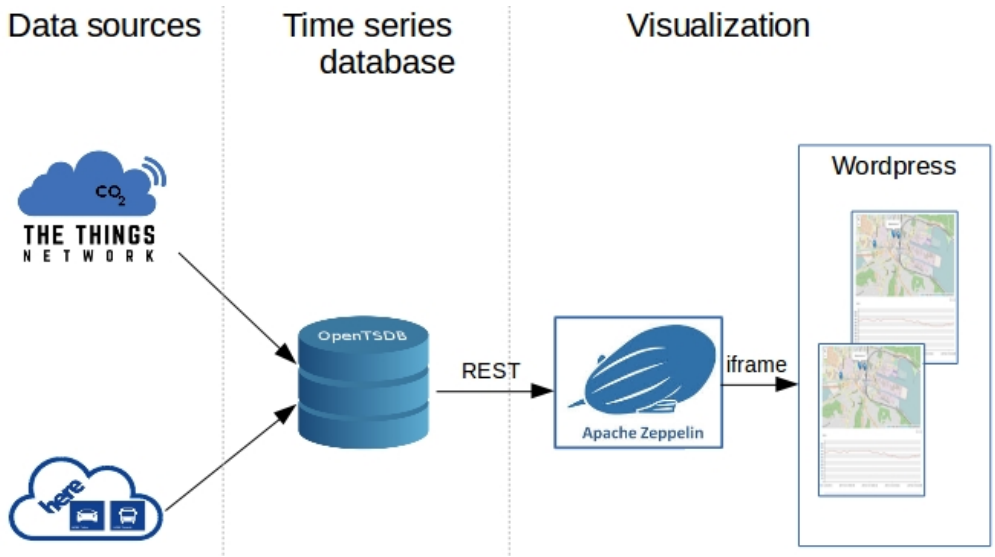
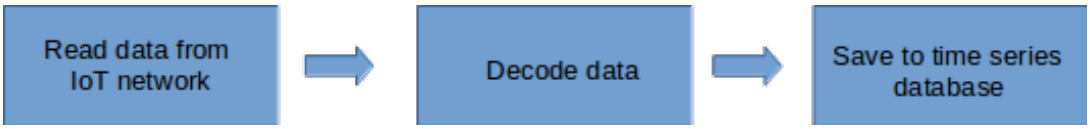


Figure 2.2.1. The architecture of CTT dashboards

2.2.1. Data sources

- Sensor data from IoT network

There are five sensors collecting the air quality data and weather data for roughly every six minutes, including the data of CO2, NOx, PM1, PM2.5, PM10, and weather conditions (incl. temperature, humidity, and pressure).



First, in order to access the data from IoT network, some information has to be provided, including the name of application, the hostname (or IP address) of the IoT network broker, the port, and the credential, e.g.,

```
{
    'applicationName':'CTT_Vejle',
    'brokerHost':'staging.thethingsnetwork.org',
    'brokerPort':1883,
    'appEUI':'<insert appEUI here>'
     'accessKey':'<insert accessKey here>'
}
```

These information can be acquired from the IoT network broker website, where users creates the application, and registers sensor nodes.

The CTT dashboard system reads the data from the IoT network through the following URL format:

```
www.thethingsnetwork.org/api/v0/nodes/{Sensor_ID}/?limit={data_limit}&offset={offset}
```

Sensor_ID is the label for uniquely identifying each sensor, and currently there are four sensors installed in Vejle whose ID are 02032220", "02032221", "02032222", "02032201" (but the first three are active). Data_limit specifies the size of data accessed for each time, and the offset specifies the starting position of reading the data.

The raw data read from the IoT network is encoded, and it has to decode to be human readable. Following is an example of the data package after the decoding:

```
{

  'SENSOR_BAT': 78,
  'SENSOR_GP_CO2': -99.0,
  'SENSOR_GP_HUM': 100.0,
  'SENSOR_GP_NO2': 0.0,
  'SENSOR_GP_PRES': 100954.546875,
```

```
  'SENSOR_GP_TC': 19.440000534057617,
  'SENSOR_OPC_PM1': 104.28593444824219,
  'SENSOR_OPC_PM10': 160.12428283691406,
  'SENSOR_OPC_PM2_5': 143.34170532226562,
  'altitude': 10,
  'channel': 3,
  'codingrate': '4/5',
  'counter': 8390,
  'crc': 1,
  'datarate': 'SF7BW125',
  'dev_eui
  'frequency': 867.1,
  'gateway_eui': '0000024B080E06B3',
  'gateway_time': '2016-08-28T21:21:24.707556Z',
  'gateway_timestamp': 3165322931,
  'latitude': 55.70799,
  'longitude': 9.53225,
  'lsnr': 6.5,
  'modulation': 'LORA',
  'payload':
'PD0+ADdk4IcYVkpDVFQwMSPDjwAAxsKJAAAAAJAfhZtBkgAAyEKTRi3FR5dmktBCmHpXD0OZ0
R8gQzRO',
  'port': 3,
  'rfchain': 0,
  'rssi': -106,
  'server_time': '2016-08-28T21:21:24.72138545Z'
}
```


The following table lists the measures collected from sensors, and the metric name when they are stored in OpenTSDB, and the description.

Time series	Metric name in OpenTSDB	Description
SENSOR_BAT	SENSOR_BAT	Battery level, [0-100%]
SENSOR_GP_CO2	SENSOR_GP_CO2	CO2 value [ppm]
SENSOR_GP_HUM	SENSOR_GP_HUM	Relative humidity [%]
SENSOR_GP_NO2	SENSOR_GP_NO2	NO2 value [ppm]
SENSOR_GP_PRES	SENSOR_GP_PRES	Atmospheric pressure, [Pa]

SENSOR_GP_TC	SENSOR_GP_TC	Temperature, [celsius]
SENSOR_OPC_PM1, 2.5, 10	SENSOR_OPC_PM1, 2.5, 10	Particle pollution value [micrometer]

- Traffic data from Here.com

The traffic data in this case study is retrieved from Here.com. Here is a company which provides mapping data and related services in relation to location content, such as road networks, buildings, parks and traffic pattern. To access the data from here.com, users have to register account, and acquire an app id and app code to access the data (Please note that the trial account is only valid for three months). There are several methods to retrieve traffic flow information, and in our approach, we retrieve data using a Bounding Box, which specifies a rectangular area centered on the city center. The request is made using the bbox parameter in the request URL with the latitude and longitude. Following shows the example of the request URL that we use to retrieve traffic data in Vejle.

```
https://traffic.cit.api.here.com/traffic/6.1/flow.json?app_id=FWrbTzLYjI7sIMRotCpH&app_code=phkACBYX2U-IoBoQ4PMReQ&bbox=55.729597,9.487217;55.670084,9.592913
```

Figure 2.2.2 Bounding Box of data accessing

The response data can be in XML or JSON format, while in this case study we read the data as JSON format shown in the following.

```
[
```

```
        {
          "TMC":{
            "PC":4287,
            "DE":"Vedelsgade",
            "QD":"+",
            "LE":0.03416
          },
          "CF":[
            {
              "JF":4.28462,
              "CN":0.72,
              "SP":20.35,
              "TY":"TR",
              "FF":33.0,
              "SU":20.35
            }
          ]
        },
      ...
]
```


The following shows the transport metrics in the data source, the metric names in OpenTSDB and the description.

Time series	Metric name in OpenTSDB	Description
JF	ctt.traffic.JF	Traffic jam factor. The number between 0.0 and 10.0 indicating the expected quality of travel. When there is a road closure, the Jam Factor will be 10. As the number approaches 10.0 the quality of travel is setting worse. -1.0 indicates that a Jam Factor could not be calculated.
CN	ctt.traffic.CN	Confidence, an indication of how the speed was determined. -1.0 road closed. 1.0=100% 0.7-100% Historical Usually a value between .7 and 1.0.
SP	ctt.traffic.SP	Speed (based on UNITS) capped by speed limit
TY	ctt.traffic.TY	TMC for TMC roadways/linears. Ignore RWS element if TY is unknown.
FF	ctt.traffic.FF	The free flow speed on this stretch of road.
SU	ctt.traffic.SU	Speed (based on UNITS) not capped by speed limit

2.2.2 Time-series data management

- Time series database - OpenTSDB

The CTT dashboard system employs OpenTSDB as the time series database for storing the sensor data, as well as the traffic data. OpenTSDB is an open source distributed and scalable time-series database, developed by StumbleUpon. It supports real time collection of data points from various sources, and is designed to handle terabytes of data with better performance for different monitoring needs. OpenTSDB uses HBase as the underlying data storage. Figure 2.2.3 describes the architecture of OpenTSDB. OpenTSDB has a set of so-called Time Series Daemon (TSD) and command line utilities. Each TSD is an independent process, a wrapper for accessing HBase database. Each TSD uses HBase to store and retrieve time series data. TSD itself supports a set of data accessing protocols, including telnet-style protocol, an HTTP API or a simple built-in GUI.

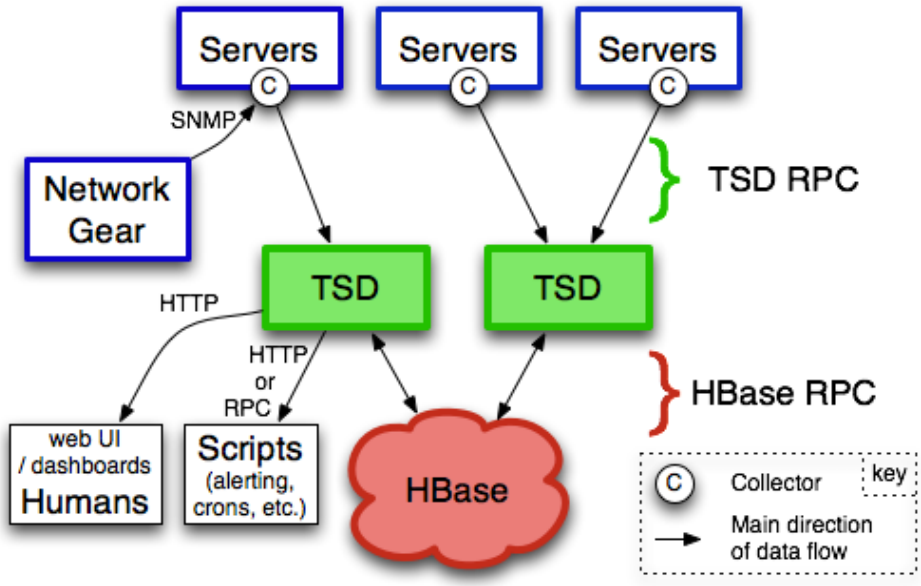


Figure 2.2.3 OpenTSDB architecture

HBase is an Apache an open source distributed column-oriented NoSQL database. The biggest advantage of HBase is its supporting scalable real-time reads and writes. Data in HBase are logically organized into tables, rows and columns. Columns in HBase can have multiple versions with the same row key. HBase can efficiently manage millions or billions of rows. All rows in HBase are sorted lexicographically by the row keys. HBase provides java API for client interaction. Therefore, HBase is a good choice for both operational and analytical applications, including IoT, user analytics, and financial data analysis.

OpenTSDB, built on top of HBase, allows to collect thousands and thousands of metrics from different sources at a high rate. OpenTSDB stores the data in HBase in two different tables: tsdb table and tsdb-uid table.

- tsdb table provides storage and query support over time-series data.

- tsdb-uid table maintains an index of globally unique values for all metrics and tags.

In OpenTSDB, the row key is composite and combination of: the metric ID, a timestamp, the tags. Each tag is combination of tag key ID and tag value ID.

<metricID><base-timestamp><tag-key-id><tag-value-id>....

A metric ID is located at the start of the row key if a new set of busy metric are created, all writes for those metric will be on the same server until the region splits. With random ID generation enabled, the new metrics will be distributed across the key space and likely to wind up in different regions on different servers. OpenTSDB also uses HBase caching mechanism called the Block Cache which provides quick access for fetch data point. OpenTSDB provides Java API and HTTP API (JSON). OpenTSDB is built using Java library and Asynchronous.

- Data Addition

There are several OpenTSDB clients implemented in different programming languages found at the resources webpage. The write and read programs in CTT project were implemented using the Python Client, potsdb . potsdb client makes time series addition rather simple. For each time series, it is just simply specify the name of the metric, its value, and the timestamp, while all the other attributes are added as <tag, value> pairs into the OpenTSDB.The following code snippet shows adding the SENSOR_GP_TC time series values as an example.

```
import potsdb

client = potsdb.Client('129.241.107.186', port=4242)

def add_message_to_opentsdb(message):
    try:
        client.log('SENSOR_GP_TC', message['SENSOR_GP_TC'], # name of metric, value
                   timestamp=message['server_time'], #timestamp
                   node_eui=message['node_eui'], # tag1=value1
                   gateway_eui=message['gateway_eui'], # tag2=value2
                   rssi=message['rssi'], # ..
                   channel=message['channel'],
                   counter=message['counter'],
                   datarate=message['datarate'],
                   frequency=message['frequency'],
                   lsnr=message['lsnr'])
        client.close()
    except Exception as e:
        print(e)
```

- Data Retrieval

To query data from OpenTSDB, the endpoint API, /api/query, extracts the data from the storage system in various formats determined by the serializer selected. For example, the following URL queries the last added data points of the time series of ctt.traffic.JF. The more advanced queries format can be found at OpenTSDB's online document

http://129.241.107.186:4242/api/query/last?timeseries=ctt.traffic.JF&resolve=true

OpenTSDB responses this query with the following results in JSON format:


```
[
 {
   "metric":"ctt.traffic.JF",
   "timestamp":1484397372000,
   "value":"2.806950092315674",
   "tags":{
     "EXTENDED_COUNTRY_CODE":"E1",
     "DE":"Strandgade",
     "EBU_COUNTRY_CODE":"9",
     "PC":"5094"
   },
   "tsuid":"000003000001000001000002000011000003000003000004000042"
 },
...
]
```


2.2.3 Data visualization

- Apache Zeppelin

In the CTT dashboard, we use Apache Zeppelin as the visualization system. Zeppelin is a web-based data science tool for interactive data analytics. Zeppelin supports a variety of data analytics systems, including Spark, PostgreSQL, Hadoop, Hbase, etc, by introducing the "interpreter" concept, i.e., by implementing the interface for interpreting the programming language of a system. In Zeppelin, users can make beautiful data-driven, interactive and collaborative documents with SQL, Scala and more.

The following figure shows the user interface of implementing the dashboard using javascript in Zeppelin. The javascript program is interpreted by “angular” interpreter to access the time series data in OpenTSDB. This program also uses the leaflet javascript library to for showing the sensors location on open map, and the echart library for showing time series in charts.

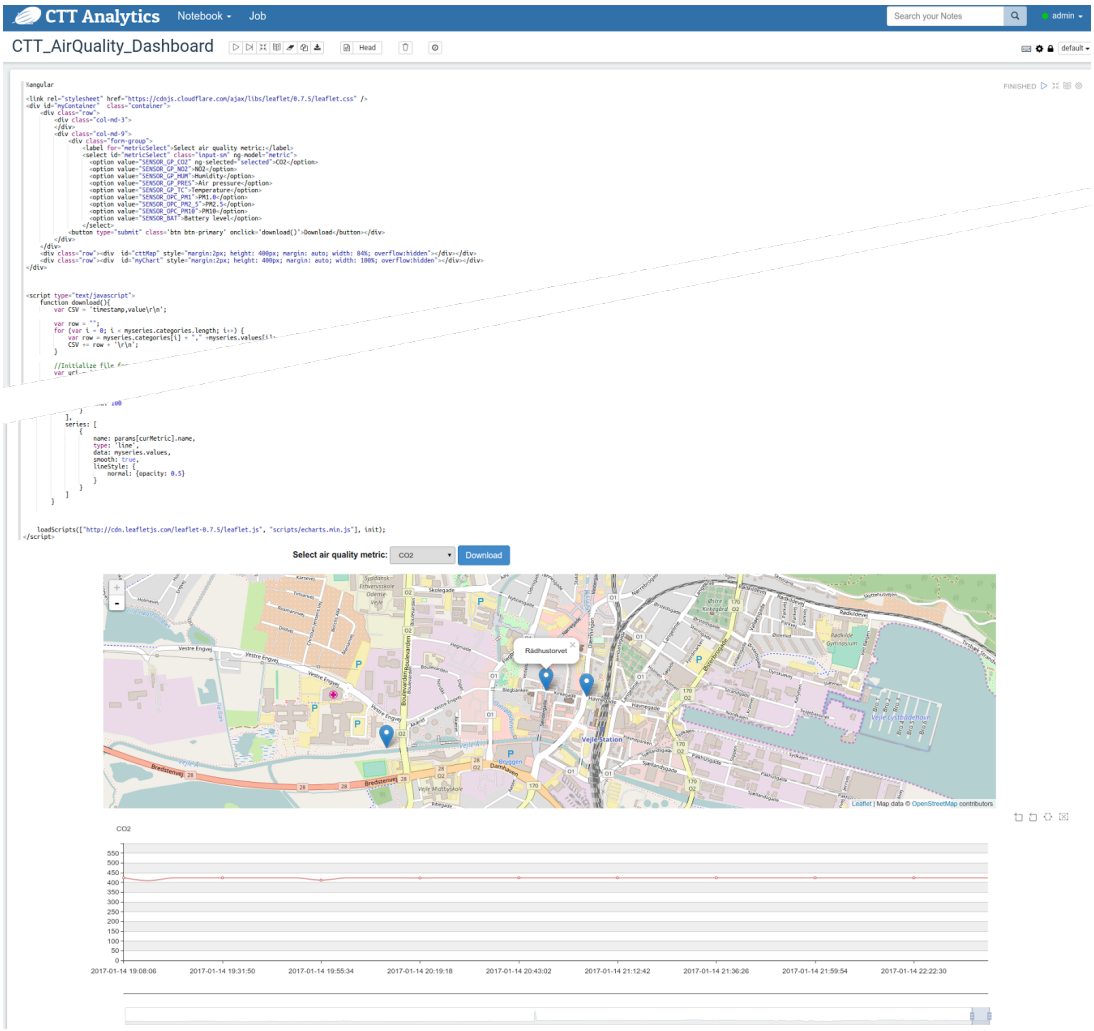


Figure 2.2.4 Zeppelin notebook

The benefit of using Zeppelin as the visualization tools is that the “paragraph” in zeppelin notebook can be shared by a link, which can be embedded into any web page with iFrame. For example, we use the following iframes embedded into the wordpress web page to display the dashboards.

<iframe src="http://129.241.107.186:8081/#/notebook/2C4T4W8ZJ/paragraph/20161205-204745_218250481?asIframe" width="100%" height="960" scrolling="no" frameborder="0"> </iframe>
<iframe src="http://129.241.107.186:8081/#/notebook/2BZ2N9FU9/paragraph/20161030-210122_1775943929?asIframe" width="100%" height="960" scrolling="no" frameborder="0"> </iframe>

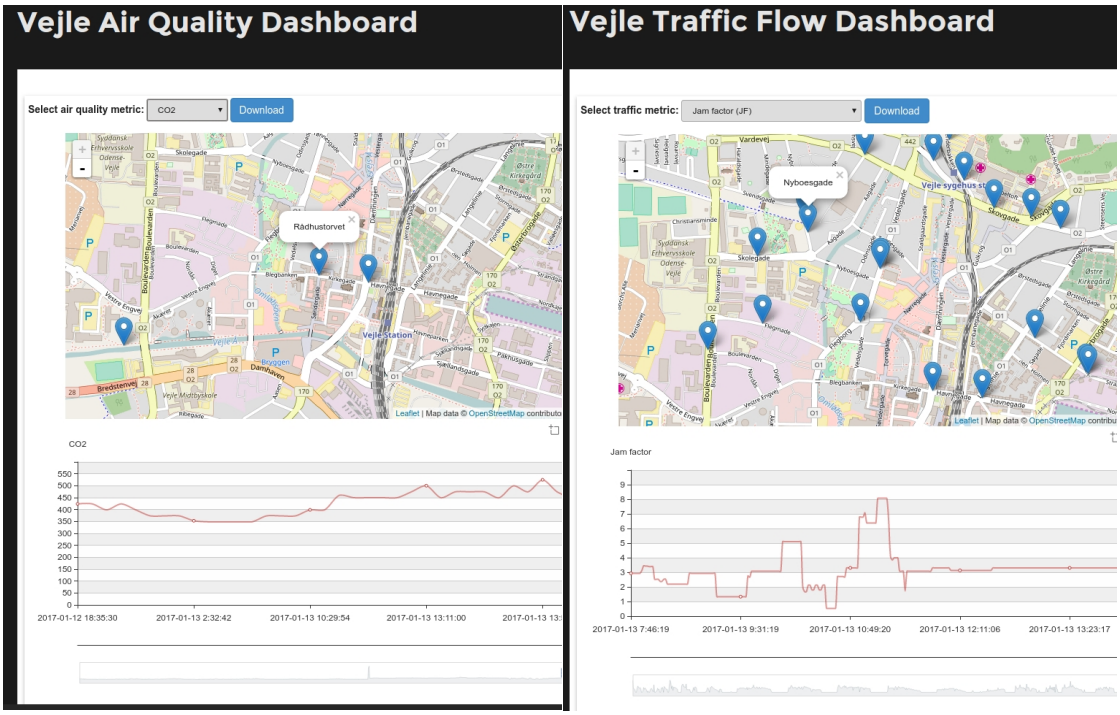


Figure 2.2.4 Embed the dashboard in Wordpress web pages

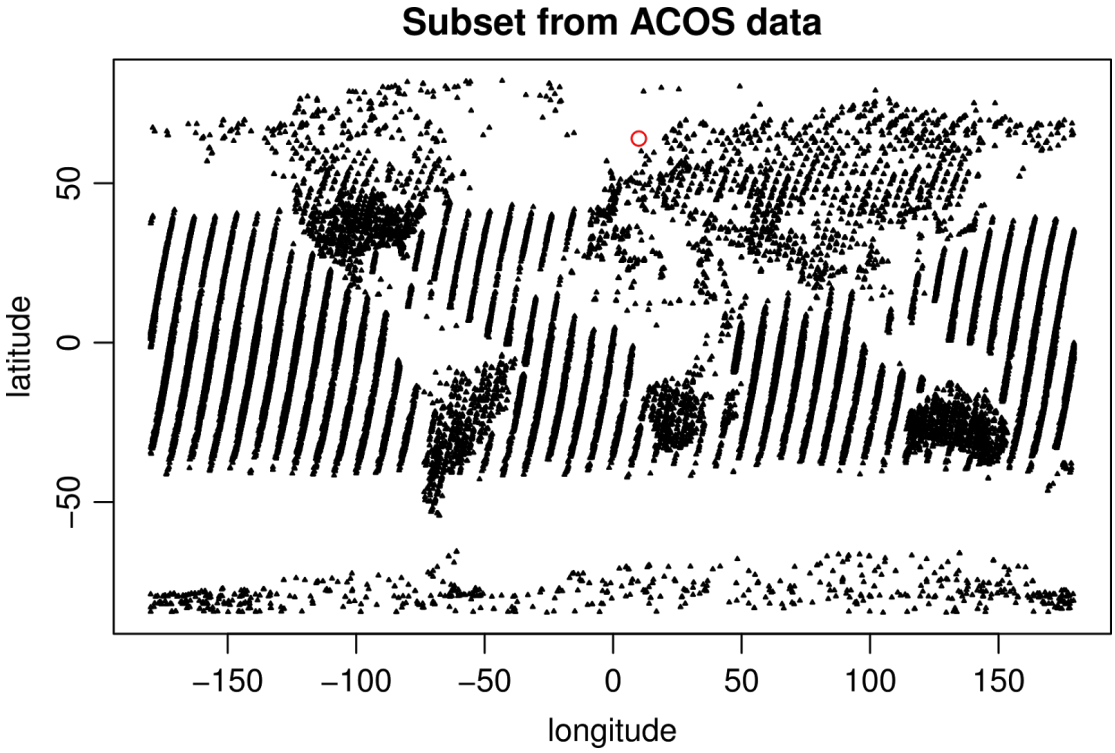
2.3 Other data sources

The main data sources are presented in the CTT Dashboard. However, other data sources have also been approached. Here a short overview is presented for each source.

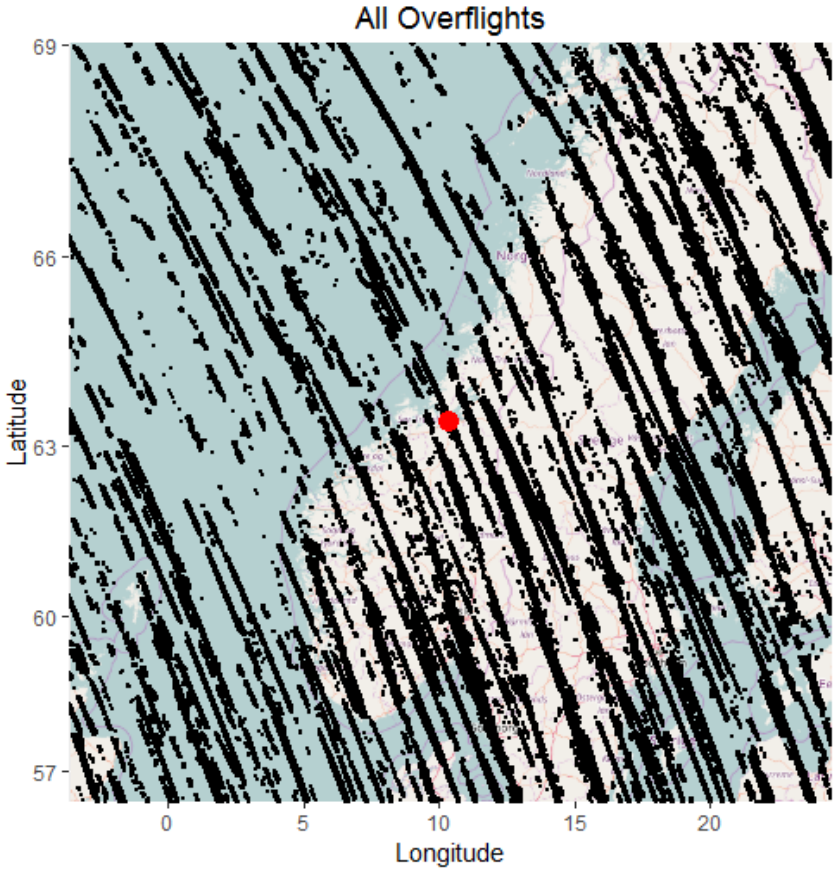
2.3.1 Atmospheric CO2 from satellite missions

It is of interest to see if it is possible to see the influence of cities from above. The two cities in the project, Trondheim and Vejle, have quite different surroundings. Trondheim was chosen as the first city to investigate, as it doesn't have any other cities of comparable size in within a rather long range. The first step was to investigate the coverage of the city and the time between over-flights.

The NASA JPL ACOS mission is part of the Earth Observatory [ref] and provides measurements of atmospheric CO2. The plot below presents the locations where there are measurements (black) - a fixed pattern is observed over the oceans as well as selected land areas. The red circle marks the location of Trondheim and no measurements are made in that region - so no further analysis is made with this data.

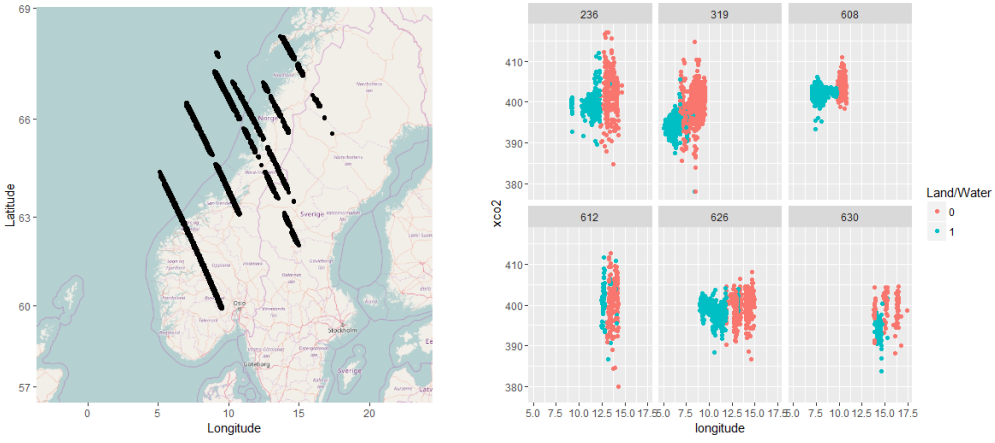


The OCO-2 satellite mission [ref] delivers atmospheric measurements of CO2 with a reasonable good coverage. Below is a map showing all observations in the lite version that only includes observations fulfilling some quality criteria.



OCO-2 overflights lite version (Black dots) around Trondheim (Red dot). (Made by Kasper Einarson)

The OCO-2 clearly provides a much better coverage around Trondheim. However, many of the overflights are very patchy (as only observations with good quality are included) and thus it is difficult to assess the contribution of the city.



Six selected overflights that contain many observations crossing Norway and into the ocean. Map (left) and observations (Right). (Made by Kasper Einarson)

The first thing to observe is that the variance seems to be higher when over land. And the overpass near Trondheim doesn't seem to differ from the others. What can be said is that the background atmospheric CO2 concentration is expected to be around 400ppm.

2.3.2 Local city data

Many suggestions for getting local data have been suggested but there have been difficulties acquiring the data. One example was the use of tracking data from all public busses in Vejle as a measure of the traffic flow. It was easy to get arrival times for bus stops however the data didn't include if the bus actually stopped nor for how long. We were told that data on the actual position is recorded however it was not possible to get hold of that data within the timeframe of the project.

Another idea was to mount a node on dump trucks serving the entire municipality and thereby get readings near every household. Again, this wasn't possible within the timespan of the project - and some testing should be do to find the best way to avoid just measuring the exhaust from the truck it self.

It was decided to start with a focus on the data that is collected through the project.

3. Analysis of the data

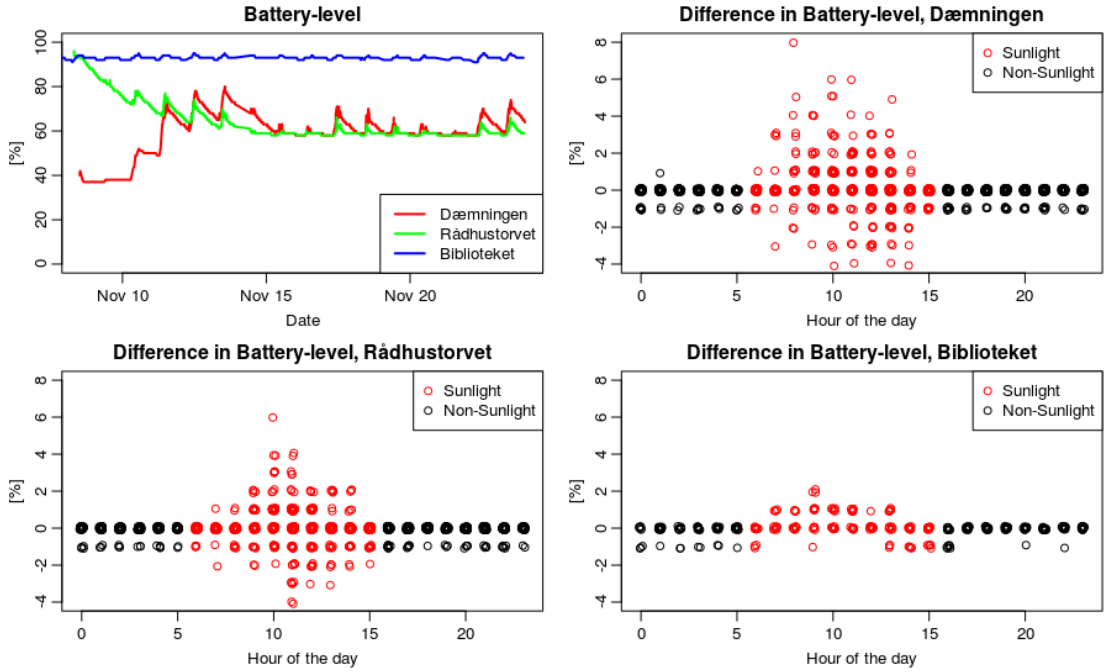
This section will focus on analysis of the data from the nodes that were mounted in Vejle as part of the CTT 2.0 project.

3.1 Battery Consumption

One of the cornerstones within the Carbon Track and Trace project is the ability to make local measurements. For the first 3 months of the project in Vejle we were working without a fixed power source for the nodes, and therefore we had to rely on solar panels to power the node- battery level high enough, so that we could keep a steady flow of data coming from the nodes.

The main issue is that every day have a limited window, where the solar panels are actually charging the batteries. Therefore it is crucial that we gain enough power during this short window to keep the nodes running throughout the night and on grey days.

The battery consumption for the three nodes in Vejle and their charging windows looks as follow. Where the top left plot shows the battery-level as a function of time, and the remaining three plots shows the difference in battery-level from previous send package versus time of day, and where the sunlight indicates (red) whether there is a chance that the nodes has charged since the previous package.



Here it becomes clear that we are consuming more energy than we are actually producing for for the two nodes at Dæmningen and Rådhustorvet. The reason why we see a larger consumption for only those two are that we are using different intervals and that these two also have a PMx nodes opposed to the last node at Biblioteket.

In the product information provided we can see the expected energy consumption for the different node devices. We want to investigate if this is equivalent to the actual consumption from the nodes, in order to find the optimal setting for running these without a fixed power source.

For calculating the actual consumption of energy by the nodes, we need to look at them at times where we are sure, that these are not charging which is at night, which is the areas indicated with black dots on figure 1.

For the model our response is the time that it have taken a node to drop one percentage point (timeStep [minutes]) of the battery-level given the average temperature (temp [°C]) within that time window together with the amount of send packages (step), the type of package send (PS) and the ID (node_eui) for the given node. It is crucial that we are looking outside of the time window where the nodes are able to change, so that we avoid overlaps between a possible and non-possible changing time, within both the time for a drop in percentage, and the amount of send packages. The model we want to use is a linear model given as:

$$timeStep \sim (temp + step + PS + node\_eui)^3$$

Where higher order terms represent interactions. For the type of package send we are working with three different; package one where only the battery-level is measured, package two where everything except PMx is measured and package three where all sensors are used.

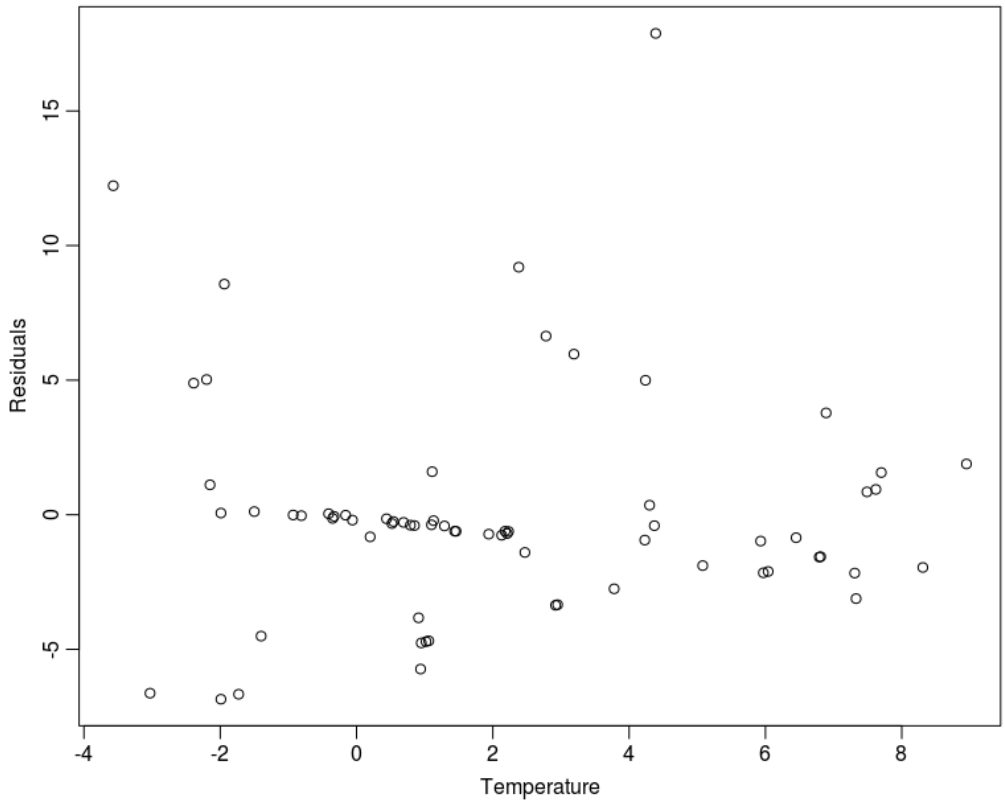
By creating the model in R and then reduce it to only containing the significant parameters (Using a 5% level), gives the following summary.

```
Call:
lm(formula = TimeStep ~ node_eui + temp + step + PS + node_eui:temp
+ node_eui:step, data = NewNodeP)

Residuals:
    Min      1Q  Median      3Q     Max
-6.8536 -1.9071 -0.4034  0.1755 17.8822

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)         -3.2260     3.0274  -1.066  0.29080
node_euinode2        2.6333     4.4081   0.597  0.55246
temp                -0.7109     0.2064  -3.445  0.00104 **
step                11.0628     0.3212  34.447  < 2e-16 ***
PS2                 -7.7613     2.4836  -3.125  0.00272 **
node_euinode2:temp   0.9306     0.4190   2.221  0.03007 *
node_euinode2:step  -1.0488     0.4343  -2.415  0.01874 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.346 on 61 degrees of freedom
Multiple R-squared:  0.9747,    Adjusted R-squared:  0.9722
F-statistic: 391.6 on 6 and 61 DF,  p-value: < 2.2e-16
```

What is interesting to notice is that we are not including node 3 in the model. This comes from the fact that we are only looking at time windows where we see a drop in battery-level and where the node is not charting - this is not happening for node 3 at any time. This comes from the fact that node three only send package two, which is the one where we do not include the PMx-sensor, and it only sends once per hour.

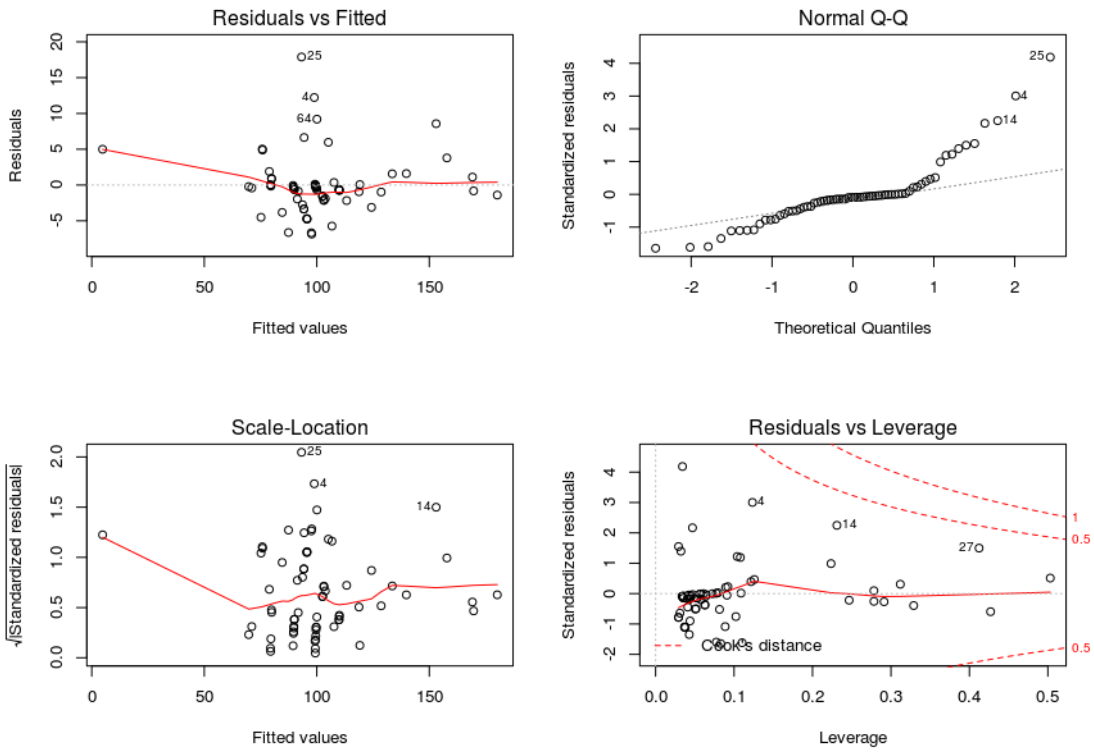
Further, from the summary it can be seen that every time a package type three is sent, it can be expected to account for around 11 minutes of the time until the next drop in battery-level. If only the packages type two were sent, this could be expected to account for only around 4 minutes. Also, there is no significant difference between the nodes number one and two.

Last the temperature is indicated as significant, therefore the following plot shows the temperature plotted against the residuals.



Here it is seen that the residuals are not perfectly distributed. Since these are located close to zero degrees, these are not expected to have a large impact on the model, but these can still account for some of the impact that the temperature parameter has on the model.

In order to talk about the accuracy of our reduced model, we can investigate if our residuals is expected to follow a normal distribution given the following tests.



Here it is seen that the normal distribution can be expected to be shifted toward the positive side, which is not ideal. Further, we can see that outliers can not be expected.

The conclusions that we can draw from this is, when talking about drop in battery-level, the most significant parameter is the amount of packages send, and especially which type of sensors are used.

3.2 NO_2 measurements

One problem within the sensor setup is the NO_2-sensors provided by Libelium. By looking at the product informations provided, we see that the sensors are able to measure in the range of 0-20 ppm, with an accuracy of 0.1 ppm.

When looking at the Danish limit values for air quality for NO_2, we see that the value for the annual average is correspond to 40 $\mu g/m^3$ and for the maximum one hour exposure it is 200 $\mu g/m^3$. In ppm these two values are equivalent to 0.04 and 0.2 ppm.

Therefore with our current NO_2-sensors we would only be sure that the concentration is above the maximum exposure if it is actually 50% above the maximum level. And the annual average concentration should be rounded to zero. This is the reason why we get mainly measurements equal to zero from the NO_2-sensors.

3.3 Modelling the dynamics in CO_2 emission

One thing we want to investigate throughout the project is whether we can model the dynamics within the CO_2-emission measured by the three sensors located in Vejle. As for the NO_2 it is important to consider the specification of the sensor. Libelium states that sensor has a resolution of 25ppm and an accuracy of 50ppm. Furthermore, the reading is expected to drift up to 250ppm per year.

The fact that the nodes are different further complicates the data as we cannot expect to receive data-packages from each of our sensors at the same time and with the same frequency; to comprehend this problem we use a Kalman Filter. This gives us the ability to reconstruct our data including the missing observations based on the expected variance.

Using a Kalman Filter also allows us to model each of our time series relatively to the overall mean value - that is allowed to drift slowly to accommodate the annual drift. Since we expect the offsets for each of the series to be incorrect, we are only interested in investigating the short-term dynamics within the two series.

The first model investigated is an AR(2)-model, with parameters estimated based on minimizing the negative log-likelihood. Since we are interested in estimating the model based on the times series relative to the overall mean, the model looks as following:

$$\begin{bmatrix} node_1 \\ node_2 \\ mean_1 \\ mean_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -\phi_1 & 1 \\ 0 & 0 & -\phi_2 & 0 \end{bmatrix} X_{t-1} + \begin{bmatrix} 0 \\ 0 \\ \Theta \\ 0 \end{bmatrix} u_{here} + \{\sigma_{1,X}, \sigma_{1,X}, \sigma_{2,X}\}$$

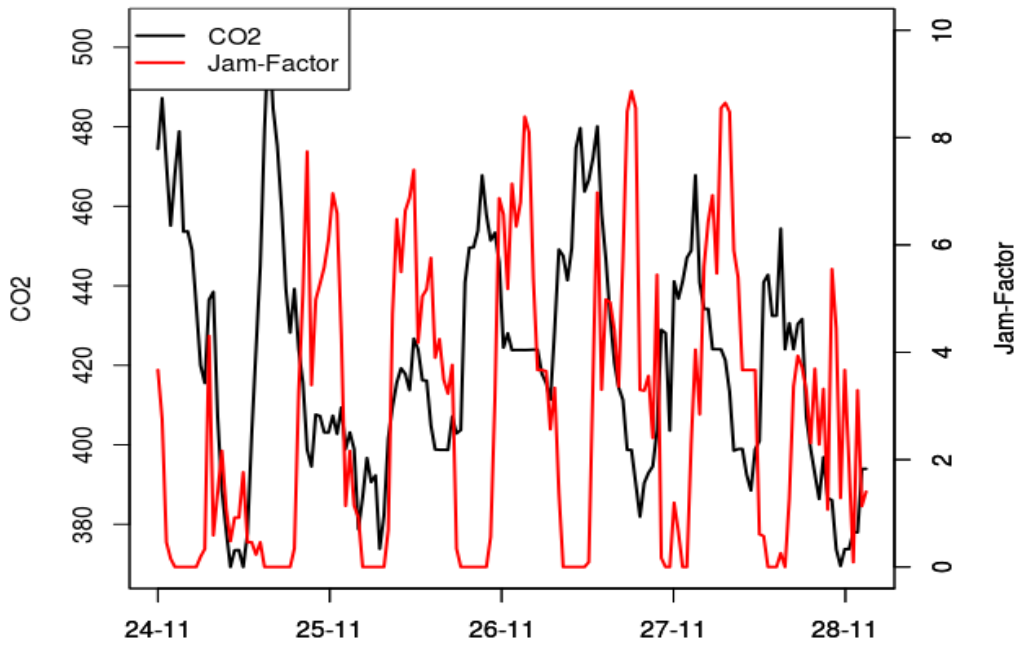
$$\begin{bmatrix} ynode_1 \\ ynode_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} node_1 \\ node_2 \\ mean_1 \\ mean_2 \end{bmatrix} + \{\sigma_Y, \sigma_Y\}$$

Where $ynode_1$ represent the CO_2-emission measured by the node at Rådhustorvet and $ynode_2$ represent the CO_2-emission measured at the node at Dæmningen.

Further, we are including the Jam-Factor [u_{here}] taken from our own Dashboard. The Jam-Factor takes a value from zero to ten and represent how congested a road is most likely to be at a given time. The Jam-Factor is added to the overall mean and is expected to account for some of the dynamics within the CO_2-emission. After minimizing the negative log-likelihood the following parameters are estimated to be the most like:

Parameter	Values
φ_1	-1.43
φ_2	9.12e-9
$\sigma_{1,X}$	2.00
$\sigma_{2,X}$	5.000
Θ	1.00e-e3

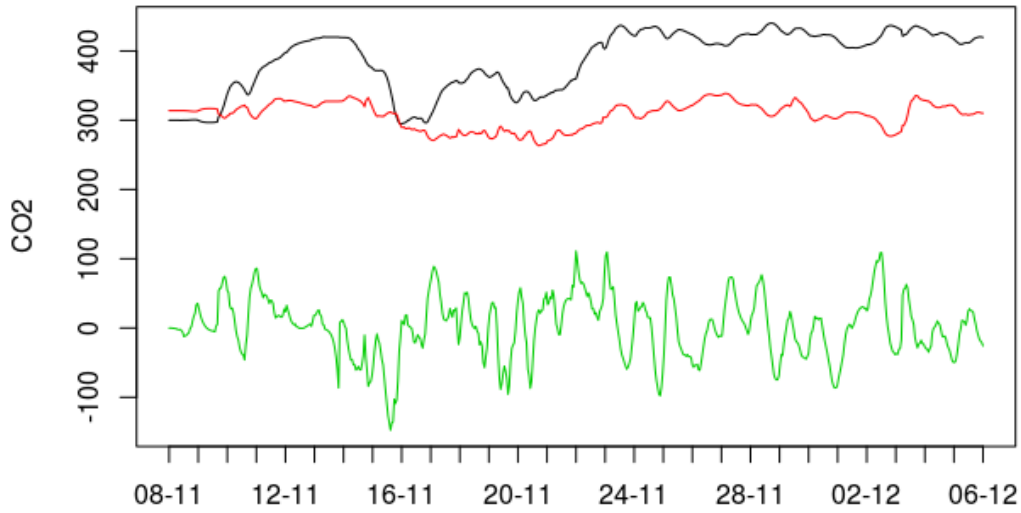
What is interesting is that the Θ-parameter takes a value close to zero. This means that the Jam-Factor is not accounting for any of the dynamics within the CO_2-emission. This can be the case since the expected daily-profile for the Jam-Factor takes sort of a m-shape with the peaks in the morning and evening rush hours. The daily-profile for the CO_2-emission is more likely to take a s-shape, since we are expecting it to stack up doing the day and the drop at night. This is also what the following plot aims to show:



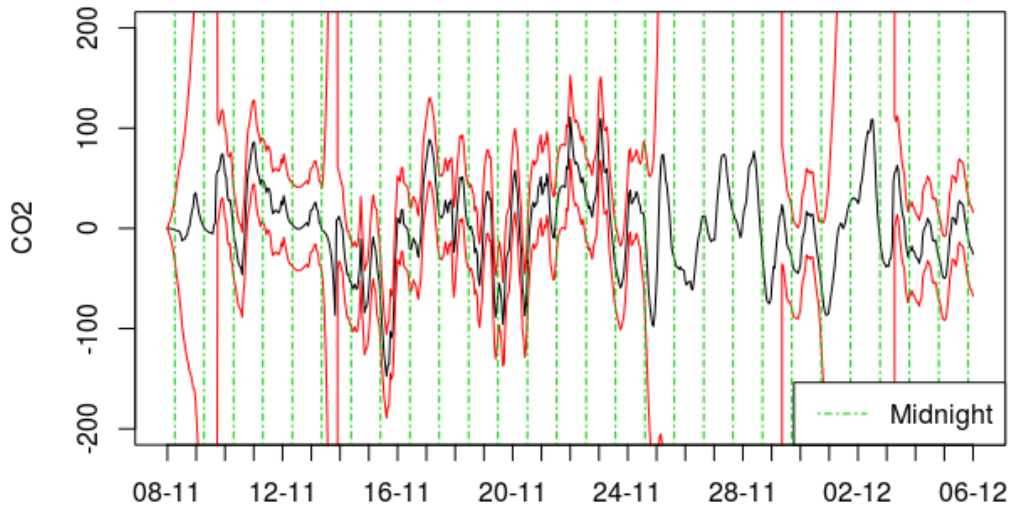
Where we see the s-shape for the CO_2-emission and the m-shape for the Jam-Factor. Therefore it is preferable to use models that are able to account for this dynamics.

Further, the error added to the system is estimated by the negative log-likelihood whereas the error added to the observation is based on the uncertainty given by the product information, which happens be an accuracy of ±25 ppm.

The CO_2-emission relative to the overall mean looks as follows, where the green line represents the overall mean, and the red and black lines represents the relative CO_2-emission.



Here it can be seen that the two sensors are expected to drifting over time from the overall mean, but also from each other. The overall mean with a 95% confidence band looks as follows.



Here it can be seen that we can expect the most days to have a daily-pattern equivalent to a raise of around 100 ppm during the day and a drop of around 100 ppm during the night in CO_2-emission. The reason for the confidence bands to go towards infinity is that for some periods we are lacking data, but as soon as we start to receive data again, these starts to stabilize.

From the estimated parameters we cannot argue that this pattern come from the traffic, this is most likely the case since multiple factors is accounting for the CO_2-emission.

4. Future Work

Due to issues with the nodes then there was only limited time for analysing the data. Hence, the above modelling results should be considered as first steps and it is expected that better models for CO_2 and PM_x can be obtained.

It is worth mentioning that the Here data records the so-called free speed to be 40 km/h where it is known that the speed limit is 50 km/h and that most cars drive at speeds near the speed limit when not congested.

In mid January a bachelor of science thesis with analysis of the PMx data will be handed in. The thesis will be shared when available.

5. Conclusion

This part of the CTT 2.0 project created a united platform for collecting and presenting the data from sensor nodes in a city. The code is wrapped in a Docker image for easy deployment in other cities. Satellite data has been investigated but appeared to be of limited value to the CTT 2.0 project as update cycles and coverage are far from real time but can help estimate and calibrate to background concentrations.

It is worth mentioning that this range of sensors isn't sufficiently sensitive to handle free atmospheric air - In particular the NO_2 and CO_2 sensors have problems with the accuracy under the given conditions.

Initial analysis of the data has been performed and diurnal variations have been detected for the CO_2 concentration. However, the large and repeated gaps in data due to malfunction of the nodes reduced the information that can be extracted from the data. It is expected that the setup - if stabilized - can provide information on the climate footprint of a city by assessing the magnitude of the diurnal variation. But further development will be needed both on the sensor platform and the modelling of the data.