University of Twente

Creative Technology

# Low-cost Autonomous Temperature Measurements System

Max Pijnappel

supervised by
Ir.Ing. Richard Bults and Ir. Hans Scholten

14-02-2020

## Abstract

The ITC wants to develop an accurate weather model for the city of Enschede. To accomplish this eighty sensor nodes will be distributed through the city of Enschede. The goal of this graduation project is to build one of those low-cost autonomous sensor nodes. The sensors of the node are under € 400,- and measure air temperature, solar radiation, relative humidity, and wind speed. The sensors also meet the requirements set by the ITC. Besides the senors, a power management system was built and a housing that protects it.

This study finds that the sensors perform well and an housing that allows for easy swapping of sensors. And power is provided by solar energy with a li-ion battery. The whole system is controlled by an ESP32. For future work, it would be recommended to implement a system that stops charging at sub-zero temperatures since that can damage the battery.

## Acknowledgement

There are several people I would like thank for there help during this project. First of all I would like to thank Richard Bults and Hans Scholten for supervising me during the project. Secondly I would like to thank Alfred de Vries for assisting me in building the prototype and allowing me to use all 3D printer for an week. Lastly I would like to thank Wim Timmermans my stakeholder in helping me set requirements for the sensors.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

## 1.1 Problem Statement

Due to the hot summers lately, like the summer of 2018 which was the warmest in 300 years [1], the city of Enschede has identified heat stress as a problem [2]. To try and combat this problem the city of Enschede has started climate adaptation initiatives. An example of such an initiative is "Wat Heet Eanske Greune Stad!" (WHEGS) . This is a project that aims to measure the air temperature in various locations with the help of low-cost autonomous sensor nodes. The WHEGS project is currently running for six months at UT-CreaTe, with a focus on functionality, feasibility and provides the requirements for this graduation project. Prior to the WHEGS project, several students of UT-Create worked on the development of prototype autonomous sensor nodes. Four parameters were found to get an accurate air temperature measurement. These parameters are: air temperature, relative humidity, wind speed and solar radiation. To make these sensor nodes autonomous there is also some work done on energy harvesting via solar energy and a low-cost weatherproof sensor node housing. And to collect the data autonomously LoRaWAN was used because of a fairly large range with low energy consumption.

However, during this development work, some problems with the reliability of the temperature sensors and energy harvesting subsystems have been found. This is why the challenge of this graduation project will be getting reliable measurements within the given budget of €400. And since this budget would be too small to have all the sensors of high-quality, research has to be done and what compromises can be made. This means finding out which parameters are most important in getting the minimum required accuracy of air temperature measurements. But the accuracy and quality of sensors are not the only factors in gathering accurate measurements. The processing of the data from the sensors is a step in the data gathering process that could result in a loss of quality. For example a bad analogue to digital converter (ADC) that has either a low resolution or not a linear measurement curve. Loss of measurement quality can also occur in the data transmission phase. For example, this can be due to a limited bandwidth of the data communication infrastructure which will not allow sending all data with a high resolution. Another problem could be lost packages when sending data. Measurements will arrive incomplete and will become unusable.

A second challenge is going to be making a low-cost sensor node housing that can protect the embedded sensors during all weather conditions while these sensors maintain the ability to keep measuring accurately. The housing also needs to able to let solar rays trough so the solar panel can charge the battery and power the controller en sensors.

## 1.2 Research question

From the previous discussion, the following research question has been made:

How to develop a low-cost autonomous system to measure air temperature developments in the city of Enschede?

To solve this, a few sub-questions have to be made concerning the quality of data since that is a major focus of the project. And not just quality but also to get the best possible quality within the given budget. So the first sub-question is:

"How to optimize the quality of measurements?"

Besides the quality of data, another aspect is protecting the sensors, controller and energy harvesting sub-systems. Since without a good sensor housing the sensors vulnerable to weather elements. And if sensors are damaged or broken they will not be able to give qualitative data or any data at all. So the second research question is:

"How to optimize the protection of used sensors and energy harvesting sub-systems?"

# 2 State of the art

To design and built the sensor node some research has to be done about the relevant sensors and the design of the sensor node housing. As well as looking at some comparable existing sensor nodes.

That is why this chapter I will examine what other sensor nodes are already made and deployed. What sensors are available for the parameters that need to be measured. And some option related to sensor node housing design. These options are for material selection but also curtain parts like the radiation shield.

## 2.1 Requirements

The four sensors that need to be chosen have to comply with some requirements set by the stakeholder Wim Timmermans. As can be seen in Table 1.

Table 1: Sensor requirements

| Parameter | Accuracy |
|---|---|
| Relative Humidity | 3% |
| Temperature | 0.3 - 0.5 °C |
| Wind Speed | 0.1 - 0.2 m/s |
| Solar Radiation | 20 W/m2 |

The sensors can, of course, have better accuracy but they at least have to fill these requirements. Another requirement set by Wim Timmermans is that at least every thirty minutes data has to be received and the data that is then sent need to be averaged. The preference of Wim is a measurement done every minute. And since the current wind sensor has to measure for 2.25 seconds, the other sensor can do multiple measurements in that minute since they are nearly instantaneous. This means that every minute average data is collected and every thirty minutes average data of those average data. In theory, this is possible but research has to be done on what that will do to the power consumption of the system.

## 2.2 IoT based weather stations

First, research will be done about a few already existing weather station sensor nodes. To see what is out there and what sensor they use. What method of transmitting data is used and what problems they come across. What advantages they have or with the disadvantages they have when compared to each other.

### 2.2.1 EnskeTemp

This graduation project will build upon the results of the EnskeTemp project. This project has the aim to set up autonomous sensor nodes all over Enschede to measure the temperature build-up in the city. The first iteration is made by Tom Onderwater[3] and only measured the air temperature. The hardware consisted of a Sodaq one which has GPS and LoRa build in. The power was provided by a solar panel of 1W and the power storage is done with a 1200 mAh LiPo battery. And the temperature is measured by the DS18B20 digital temperature sensor[4]. The temperature data is sent via LoRaWAN to the things network (TTN).

The second iteration is done by Laura Kester[5] and from the research was learned that more factors had to be measured to get an accurate temperature reading. Due to this a WH1080 anemometer from Froggit[6] was added and a SHT15[7] humidity sensor. This gives a better insight into the temperature build-up. Since wind speed and humidity influence the way air temperature feels. It also has a direct influence on the measure air temperature, this way correction to the measure air temperature can be mode when making a model.

The last build iteration is done by David Vrijenhoek[8] some small changes where made in the housing. An addition of a second solar panel was done since the changes done in iteration 2 increased the power consumption and one panel was no longer sufficient. Mostly this iteration was about testing the dependability of the sensor node. And the sensor node matched the given requirements.

All the iterations used the Sodaq one and transmitted their data over LoRaWAN to TTN. The casing overall stayed mostly the same, consisting of a radiation shield to protect the temperature and humidity sensor from the rain while allowing for airflow. A 3D printed exterior made from PLA, because it is 3D printed since that is the easiest when prototyping or when making a low volume custom product. The solar panel in a separate housing from the rest. Every iteration also used deep sleep to save on battery life.

For the moment the biggest problems are the reliability of the sensors and low quality of data. The reliability is a problem because some sensor nodes that have been deployed by Tom Onderwater stopped reading temperature values. As for quality the temperature sensor (DS18B20[4]) also was lacking since it was 0.5°CC.

And with the addition of the humidity sensor SHT15 there where now two temperature sensors. This makes the DS18B20 redundant and drains unnec-

essary power. The problem with the SHT15 is that it is an older model and no longer available so a new sensor has to be chosen that is currently available. And the froggit WH1080 is a replacement part for a different weather station and gives no real specifications but from David reliability research came forward that the accuracy was lacking.

For this graduation project, a more professional wind sensor needs to be found and a more up to date temperature and humidity sensor that is more reliable. This means the node will be more expansive but there is a budget this time. This budget allows for some more expansive sensors than previously used.

### 2.2.2 Design of an Autonomous Wireless Weather Stations

This was also a graduation project which used a Sodaq to make a weather station [9]. This weather station measured the air temperature (SHT30), relative humidity (SHT30)[10], air pressure (BMP180), wind speed (ATMOS 22), wind direction (ATMOS 22) and precipitation (RG-11). It measures all these parameters every minute and is autonomous with the use of solar panel and battery. The data is transmitted with the LoRaWAN protocol. The sensors used in this Sodaq based weather station are of high quality and can be good considerations for this graduation project. Although not all sensor are necessary and some are to expansive to use. But the SHT30 is a good candidate for high quality and reliable temperature and humidity sensor.

The power management is done with a 15-watt solar panel (Eco Line ES10P36) that chargers 3 li-ion batteries (NCR18650B) of 3350 mAh in series. The charging is managed by a battery management system (HX-3S-FL25A-A) and a DC buck regulator (LM2596).

Compared to previous iterations of the EnskeTemp project. The power system is more powerful and the used wind sensor of a higher quality without moving parts. It also is made for a higher measuring frequency than the autonomous sensor node from this GP.

To conclude this sensor ticks a lot of boxes the WHEGS sensor node needs to have. For instance, the temperature and humidity sensor and the wind sensor would be perfect if it would have been cheaper. However, it is missing a solar radiation sensor. And also the more powerful solar panel and power buffer increase its reliability, might be overkill for the WHEGS sensor node.

### 2.2.3 Weather Station Design Using IoT Platform Based On Arduino Mega

This project [11] describes a small weather station, that can display loads of weather parameter that can be read from a screen, SD card or on a website. The goal of the project was to give weather information of a place without being there. This goal was reached of course with the website this weather station offers.

The core of this weather station is an Arduino mega 2560 that connects to the internet via an ESP-8266 WiFi module. The sensors used to measure the parameters are a DHT-22 for temperature and humidity, a BMP180 for pressure and FC-37 to measure precipitation and lastly an RTC DS-3231 to keep track of the time. Part of this paper was testing the DTH-22 against a professional temperature and humidity module, the PCE-TBH 40. During the testing, they found that the mean error of the DHT-22 temperature sensor was 1.35 $^{\circ}$C with the biggest error being 3$^{\circ}$C. The humidity performed even worse with a mean error of 2.24% and the biggest error being 5%.

To conclude it is a cheap do it yourself alternative to expensive weather stations but it is not suitable for professional use. Since it currently does not have a housing which makes it vulnerable. A second problem is the lack of an autonomous power harvesting like solar power. The whole node is not aimed at

functioning autonomously. This can be seen by using Wi-Fi and an LCD screen. These additions do not add extra features when collecting data and consume a lot of power. And lastly, as for the temperature sensor they used performed poorly so the DTH-22 is not a viable option for this GP project. This is because it does not fit the given accuracy requirement of 0.3°C for temperature and 3% for humidity.

## 2.3 Non-weather sensor nodes

Besides weather sensing nodes other sensor nodes exist. These have a different goal than measuring the temperature built up, but so have things in common. Since they either function autonomously or also have to have a high quality of data in for example temperature or humidity. The challenges they come across might also occur when a sensor node is built that measure the temperature built up.

### 2.3.1 The prototype of an infant incubator monitoring system based on the internet of things using NodeMCU ESP8266

This prototype [12] is made to monitor important environmental parameters in an infant incubator. This is done by monitoring the head position, the air temperature, the air humidity and the weight of the baby.

The sensors have to be fairly precise since a low quality of data could have detrimental effects on the baby. Especially since the babies in an incubator already are at a higher risk.

The sensors that measure the data for this system is an SHT31 sensor to measure the temperature and humidity. The weight of the baby is measured with a load cell and the location of the baby is measured with the help of two HC-SR 04 ultrasonic sensor. The computing power and local internet web server hosting is done with an ESP8266 microcontroller.

This node does measure some air temperature parameters like humidity and temperature. But it is not autonomous and uses WiFi to show and send data. But since the data quality used had to be of higher quality, it can give an idea of the sensor that can provide that, which in this case is the SHT31.

### 2.3.2 Design of Mushroom Humidity Monitoring System Based on NB-IoT

This project [13] is about automating the humidity in the greenhouse that grows mushrooms. This is done by measuring the humidity and controlling the misting installation. Together they make sure the humidity is perfect for growing mushrooms. This is important since when the humidity is too high or low the mushrooms develop rust spots, hollow fruiting bodies and slow growth. But when the humidity is to low it will also slow the growth of the mushroom. The goal of the system to keep the relative humidity as close as possible to 90%. The test is done in five greenhouses

The setup works with central nodes that receive all the data over narrowband IoT from the sensing nodes and upload it to the remote management software so the data can be viewed. The central nodes can support 50.000 non-delay sensitive low-speed services with an up and downstream of at least 160 bit/s per user. The sensor that is used to monitor the temperature and humidity is an SHT20 from Sensirion. All the sensing nodes are powered with the use of a 5 Wh battery which will last 10 years. This is an excellent example of a functioning node network which can work autonomously for 10 years. However, the use of narrowband IoT might not be ideal since it is a commercial infrastructure in contrast to for example LoRaWAN. And since a special narrowband IoT band is needed it might not be cost-effective since it will not have that many nodes in its range. This can be a good example of what extra functions a network can do like controlling sprinkler systems at ideal times. The solution of this article is not applicable to this graduation project since it misses some measuring parameters. And this system is used meant for indoor use.

### 2.3.3 IoT Enabled Intelligent Sensor Node for Smart City: Pedestrian Counting and Ambient Monitoring

This article [14] is about making a sensor node that can track the number of pedestrians come past a certain point. It senses the number of pedestrians with the help of multiple sensors. It will upload this information over the LoRaWAN network and works autonomously with the help of solar power.

This node senses the number of pedestrians with a combination of sensors. The first and biggest difference it has compared with comparable sensor nodes is via a passive infrared sensor (PIR). This sensor detects when an object passes through its line of sight. By using six of such sensors it can see in what way object travel and at 3 different distances.

The second method of detecting pedestrians is with the use of a sensor (CCS811) that detects CO2 and total volatile organic compounds in the air. If the amount of CO2 rises in the air this means that more people are in the vicinity of the sensor.

The last method is by measuring the air pressure, humidity and temperature. These parameters are measured with the help of a BME280. The influence of these parameters is not really explained but it supports the PIR measurements. It also makes the system have multiple functions since it can track the weather and the number of people that walk past a certain point. This sensor node has a lot in common with what the node of this graduation project has to do. It can function autonomously and does this with the help of a solar panel and data transfer over LoRaWAN. It also measures air temperature and humidity. But it lacks the function to measure solar radiation and wind speed. It also quite bulky but it can give an idea on vandalism or theft since it also stands out and is placed in a public place. Unfortunately, the article mentions nothing about this.

## 2.4 Component and material selection

An important part about a sensor node is off course the material of the housing and the sensors that are chosen. That is why this section will focus on what 3D print filament works best and what temperature and humidity sensor fits best with in the requirements.

### 2.4.1 Sensor survey

When getting reliable quality data, choosing the right sensor is important. The first choice, one has to make is getting a digital sensor or an analogue sensor. Since the project will need more than one sensor node, but not enough to use mass production, digital temperature and humidity sensor will be used. Since digital sensors have the advantage of having their data pre-processed. However there are a lot of digital temperature and humidity sensors that can be chosen, so a comparison will be made. The sensor will be compared to the Davis Instruments vantage pro[15] and if they match the requirements given by WHEGS. The requirements can be seen in Table 1.

#### Temperature

The temperature sensors will be evaluated based on their accuracy and resolution. These two performance measures are informative indicators for the quality of their provided data. Accuracy is defined as: *the difference between an observed value and the ground truth*. In terms of performance, this means that high accuracy is achieved if the deviations from the ground truth are small. A sensor is said to have a high resolution if it can handle small step sizes. The temperature sensors that will be compared in this literature review are the BME280 [16], the SHT31 [10], the HDC1080 [17] and the MCP9808 [18]. They will be compared to the Davis temperature and humidity sensors [19], since this is a sensor used in professional weather stations. The performance measures for all temperature sensors can be seen in Table 2.

Table 2: The performance comparison of the temperature sensors

| Sensor | Accuracy (°C) | Resolution (°C) | Requirement met |
|---|---|---|---|
| Davis Instruments [19] | 0.3°C | 1°C | yes |
| BME280 [16] | 1°C | 0.01°C | no |
| SHT31 [10] | 0.3°C | 0.01°C | yes |
| MCP9808 [18] | 0.25°C | 0.5 - 0.0625°C | yes |
| HDC1080 [17] | 0.2°C | 0.01°C | yes |

The accuracy of the chosen sensors is close to each other except for one as can be seen in table 1. To illustrate, the Davis sensor [19] gives an accuracy of 0.3°C. This value holds also true for the SHT31 [10]. The accuracy is better for the MCP9808 [17] with 0.25°C and the HDC1080 [18] with 0.2°C. However, the BME280 [16] has an accuracy that is more than 3 times higher than the Davis sensor [19], with an accuracy of 1°C. So when looking at the accuracy of a sensor all the chosen sensors are candidates except for the BME280. Most likely the HDC1080 [17] will be chosen since it does have the best accuracy. Secondly, the resolution of the sensors is going to be compared. The BME280 [16], the SHT31 [10] and the HDC1080 [18] all have a step size of 0.01°C. These values are good when compared with the MCP9808 [19] which has a resolution of 0.5°C to 0.0625°C depending on which mode is chosen. However, the baseline is the Davis temperature and humidity sensor [19] and since it has a resolution of 0.1°C, all sensor options are better. To conclude the best option based on resolution is are the BME280 [16], the SHT31 [10] and the HDC1080[18]. Since the HDC1080 was the best option based on accuracy the HDC1080 is the best option based on temperature to replace the Davis temperature and humidity sensor.

## Humidity

The second part of the sensor comparison part is humidity, which again will have some sensors compared to the Davis temperature and humidity sensor. Since most sensors from the previous comparison also have both temperature and humidity. It will mostly be the same however, the MCP9808 will be left out since it does not have a humidity sensing part. Just as with the temperature sensing part some parameters will be used to compare the sensors on. For humidity, these are accuracy, resolution and long term drift. The performance of the sensors will be shown in Table 3.

Table 3: Humidity sensors

| Sensor | Accuracy(%) | Resolution(%) | Long term drift(%/year) |
|---|---|---|---|
| Davis Instruments [19] | 2% | 0.1% | <0.25 %/year |
| BME280 [16] | 3% | 0.008% | 0.5 %/year |
| SHT31 [10] | 2% | 0.01% | <0.25 %/year |
| HDC1080 [17] | 2% | 0.006% | 0.25 %/year |

The accuracy of the humidity sensing part is just like with the temperature sensing of the sensors close together. With the SHT31 [10] and HDC1080 [18] being the same as the Davis temperature and humidity sensor [19] being 2%. And only the BME280 [16] has a slightly worse accuracy with 3%. So solely based on accuracy the SHT31[10] and the HDC1080 [18] are the best options but looking at the other parameters might give a clear best option. The second parameter that is going to be compared is the resolution. And just as the case was with

temperature all sensors perform better than the Davis temperature and humidity sensor [19] which has a resolution of 0.1%. The HDC1080 [18], BME280 [16] and SHT31 [10] are close with 0.008% for the BME280 [16], 0.006% [18] for the HDC1080 and 0.01% for the SHT31. Although the resolutions are similar, the HDC1080 is the most precise and thus the best option. Thirdly the long term drift will be compared, this parameter tells how much the sensors can drift off from the ground truth over a year. The lower this value is the more precise the sensor stays over time. This parameter is the only parameter where the Davis temperature and humidity sensor [19] had the best value. That value being below 0.25 %/year and is shared with the SHT31 [10]. The datasheets do not tell how much under below 0.25 %/year the long term drift is. From the datasheets, no clear difference with the previous two sensors and the HDC1080 [17] can be derived since the HDC1080 [17] has a long term drift of 0.25 %/year. The BME280 [16] has a long term drift of 0.5 %/year which seems like a small difference with the rest. However, the drift is per year so over a span of 5 years this small difference will accumulate. Based on the long term drift the SHT31 [10] is the best replacement for the Davis temperature and humidity sensor [19]. The best replacement for the Davis temperature and humidity sensor [19] is going to be the HDC1080 [18] since its data quality was the best for all parameters both in humidity and temperature. Only in long term drift had it not the highest quality however it was close and since the best option did not have an exact value, the difference could be in terms of 0.01% per year.

### 2.4.2 Housing material

To make a sensor node not only choosing the right sensor is important, but it is also relevant to protect the sensors of the sensor node. The casing is needed to protect the sensors against the weather (rain, hail or snow). The method of making this protective casing is going to be 3D printing, but what type of material (filament) will be used to print. A lot of materials are available and one of them is Acrylonitrile Butadiene Styrene (ABS). There is some variation in ABS like the amount of silica in the ABS material. Adding 5 to 10% silica to ABS increases both the tensile strength and hardness of the filament [20]. Hence when choosing for an ABS 3D filament using one with a higher silica content will be preferred.

Besides ABS, PLA is also a popular 3D print filament. It is worthwhile to mention that the fabrication method affects the quality of the print. Letcher [21] found that when choosing a print orientation of 0°C or 45°C gave the best results when it comes to printing. Likewise, these results could be used for other materials like ABS because the printing technique is generally material-independent. The result of Letcher's [21] paper did show the best prints with a 45°C raster orientation. This would be the prefered method. However, in slight disagreement with Letcher [21], Fernandes [22] states that the best results are obtained with a raster orientation of either 0°C or 90°C. But this was only because this gave the highest tensile strength. In conclusion, the general consen-

sus for an ideal raster orientation is 0°C.

Raster orientation is not the mere variable that should be considered. The temperature of the extruder, the layer height and print speed also plays an important role. Liu [23] states that when using PLA these parameters deliver the best result when the extruder temperature is 220°C, the build layer is 0.1mm and the print speed is 60mm/s. In accordance with Liu [23], Fernandes [22] confirms that an extrusion temperature of 220°C and a build layer of 0.1mm gives the best results These would then be the preferred printing parameters with PLA. But these parameters only deliver the best result when using PLA, since these parameters are dependent on the used print material. Therefore, when PLA would be the material of choice, an extrusion temperature of 220°C and a build layer of 0.1mm will be used.

ABS and PLA are the most commonly used printing filaments, each with their own benefits. Therefore, these will be compared to choose the most adequate material for this project. The research from Shabana [24] shows that PLA is the strongest when compared to ABS. With regards to ultimate tensile strength, a combination of ABS and PLA gives the best result. When looking at microhardness and compressive strength it was equal to ABS and the PLA/ABS sandwich. The sandwich did have a higher flexure strain value but PLA was almost the same. Therefore, PLA would be the best choice for the housing of the weather station.

### 2.4.3 Discussion (temperature and humidity sensor, housing material)

To conclude the literature review when looking at the research question "How to develop a low-cost autonomous system to measure air temperature developments in the city of Enschede?" Some parts can be answered like which digital sensor fits the specified parameters the best? This review only answers this for temperature and humidity and the best sensor to replace Davis one is the HDC1080. This is because the data it gathers is either of a higher quality or equal quality when compared to the Davis one. And the sensor is cheaper than the Davis sensor.

The question "What 3D filament is best suited for printing an outside sensor node?" can be answered with PLA printed at 220°C, with a layer height of 0.1mm, a feed rate of 60mm/minute and a roster angle of 0°C. Since this was proven to be the most flexible and durable of the researched methods and materials.

However, some questions are still left unanswered. This has to do with the fact that documents and prices of pyranometers and anemometers were not readily available. Additionally, no peer-reviewed papers on sensors could be found. The only academic papers that are available are datasheets. Fortunately, datasheets are provided by the creator of the sensors. It could be helpful if independent third parties would check the given specifications. The lack of academic resources for this particular topic confirms the novelty and relevance of this study.

As of now, the research on different types of 3D printing filament for sensor node housing is rather limited, due to the absence of scientific literature on this topic. However, mainstream 3D printing technology is still up and coming. Consequently, more papers might be published over time. Therefore, for future studies, it is recommended to find more information on the use of different types of filament.

Since direct literature relating to the previously addressed topics is unavailable, it is recommended to look into related domains. Alternative fabrication methods could be insightful as well. Furthermore, material properties of other filaments, such as nylon and PETG could be investigated separately outside of the context of 3D printing. Based on the properties, it could potentially be inferred whether they are suitable candidates for this project.

Finally, since the academic community has failed to provide resources on all research questions, there is no guarantee that the conclusions drawn from this literature review are the most optimal ones. Therefore, it is time to set one of the first steps in this field by performing an exploratory study to contribute to the research community.

## 2.5 Pyranometer and wind sensor

Temperature and humidity are not the only parameters that have to be measured. The solar radiation and wind speed/direction also have to be measured. This is why this part of the state of the art will focus on the sensors that measure these parameters.

### 2.5.1 Reference from Davis

The reference sensors used will be from Davis and are the anemometer [25] and solar radiation sensor or solar pyranometer [26]. These sensors are expensive but do fit within the budget and provide sufficient quality of data. The anemometer measures wavelengths in the electromagnetic spectrum from 400 to 1100 nanometers. This is visible light and a bit from the infrared spectrum. The range of the light intensity measured in W/m2 is from 0 to 1800 W/m2. It gives an analogue output from 0 to 3V with every step of 1.67 mV being 1W/m2.

The anemometer measures the wind speed and wind direction. The wind speed can be measured in a range of 1 to 322 km/h with steps of 1km/h. The wind speed is outputted every 2.25 seconds. The sensor works by counting pulses that are created when the wind cups rotate. When the cups rotate a switch is closed which will create a short pulse. The pulses are counted for 2.25 seconds and will then be converted to wind speed. The wind direction is measured with the help of a potentiometer so by measuring the voltage output. This value can be mapped from 0 to 360 °C if the circuit inputs 3V that means that when the controller measures 3V the direction is 360 °C. If it is 1.5V the direction is 180 °C. To get the accurate direction, the north has to be facing the right side otherwise the data is only comparative within the node self.

Both these sensors will deliver a high enough quality of data. However, the anemometer contains moving parts which are vulnerable and attract attention. This attention could result in vandalism or theft since the node will be placed in areas accessible to the general public. So an option to look for wind speed and wind direction sensor without moving parts might be preferable. An example of such a sensor was in the article about a different Sodaq weather station build [4], which used the ATMOS 22 [27].

### 2.5.2 Sonic wind sensor

The best solution which is robust, accurate and attracts less attention would be a sonic sensor. However, these sensors on itself are more than the per node budget of €400. Two examples of such nodes are the ATMOS-22 [27] from Metergroup or the WindSonic from Gill Instruments [28]. But the Windsonic starts with a prize of €780 and according to the other sodaq weather station [4], the ATMOS-22 is €540. This means it is not feasible to have such a sensor on every station. But an option can be to put such a sensor on half of the sensor or a third. Since it could be possible the sensor nodes will be clustered together. This way a sonic sensor can be used which is preferred by the client.

## 2.6 Design radiation shielding sensor node

All radiation shield designs function better when the wind speed is higher. This is due to the fact that the air forces out the heat build-up that is created in the housing. However there a few types of designs of radiation shielding. Two of these are the commonly used multi-plate radiation shielding and the newer helical radiation shielding.

### 2.6.1 multi-plate radiation shielding

Multi-plate radiation shielding is used in most weather stations and allows fresh air to cycle through the housing while protecting the temperature and humidity sensor from rain and direct solar radiation. It is made by stacking multiple plates that face down at the rim as can be seen in figure 1. Between every plate, there is an air gap. An example of such a radiation shield is made by Davis [29].

The effect is not completely gone but can be corrected with a Solar radiation sensor [30, 31]. The effect can also be minimized in a few ways. The first one is making the space where the sensor is placed as small as possible. The second is making the outside of the radiation shielding of highly reflective material.

This type of radiation shield can easily be 3D printed since it could be printed per plate. This is also the used design in the previous project Enske Temp [3, 5, 8] and did not cause any known problems in measuring.

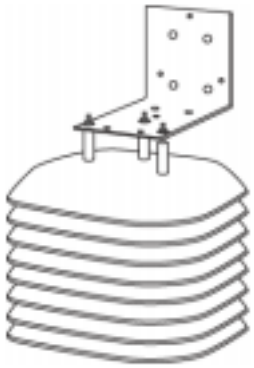


Figure 1: Multi-plate radiation shield (Davis)

### 2.6.2 Helical radiation shielding

A newer and less used form of radiation shielding is a helical design. This type of radiation shielding is made in one piece, instead of the layer design that is used in the multi-plate type. An example of this helical can be seen in Figure 2. This design is created by Barani design. The design is used in the Meteo shield line of weather stations [32]. They claim that the performance of their helical design without a fan, is better than one with a multi-plate design with a fan, as can be seen in Figure 3. It is not used widely as of yet but if the claims that are made hold, it can be a good design for a radiation shielding. The modelling and printing might be more of a challenge especially compared to the multi-plate radiation shielding. However, it can be considered and tested to see if this design works as the creator claimed.

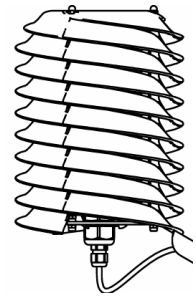


Figure 2: Helical design radiation shielding

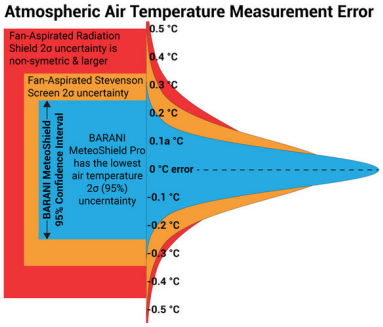



Figure 3: Atmospheric air temperature measurement error

## 2.7 Conclusion

There are some conclusions that can be made from the state of the art. The first one being that currently, no one has made a sensor node that fulfils all the requirements. Some weather nodes but they always seem to lack something. Too expansive, not autonomous, too low quality of data or do not measure a certain parameter.

The second conclusion that can be made is about the temperature and humidity sensor and three options seemed viable and need to be tested. Those sensors are the MCP9808 for temperature and the SHT31 and HDC2080 for humidity and temperature. These are cheap and match the given requirements.

The third conclusion is about the 3D print filament or the sensor node housing. The most durable material is PLA printed at 220 °C, with a layer height of 0.1mm, a feed rate of 60mm/minute and a roster angle of 0 °C. This should result in the most durable housing when using 3D printing as fabrication.

The fourth conclusion is that the pyranometer from Davis Instruments is the best option since it fits in the budget and fits the given requirement. And that either every sensor node gets an anemometer from Davis Instruments or that part of the sensor node get a more expensive sonic wind sensor from ATMOS-22 made by meter group.

The last conclusion is about the design of the radiation shielding. This is that a helical design works most effective in keeping the sensor clean and protected from the weather while not interfering with the measurements. However, some tests might be needed to conclude if this holds true since no real independent test could be found about this subject.

# 3 Method

This chapter describes what methods and techniques are used in this graduation project. The main method that structures this graduation project is the creative technology design process. The method of choosing what components are going to be in the sensor node will be the MoSCoW method. This together with interviewing the stakeholder (Wim Timmermans) to see what he finds important.

## 3.1 creative technology design process

To structure of this graduation project is given by the creative technology design process[33] and starts with a design question. In the case of this graduation project that is " How to develop a low-cost autonomous system to measure air temperature developments in the city of Enschede?". After this, four phases are used to find a solution to the design question and the whole process can be seen in Figure 4.

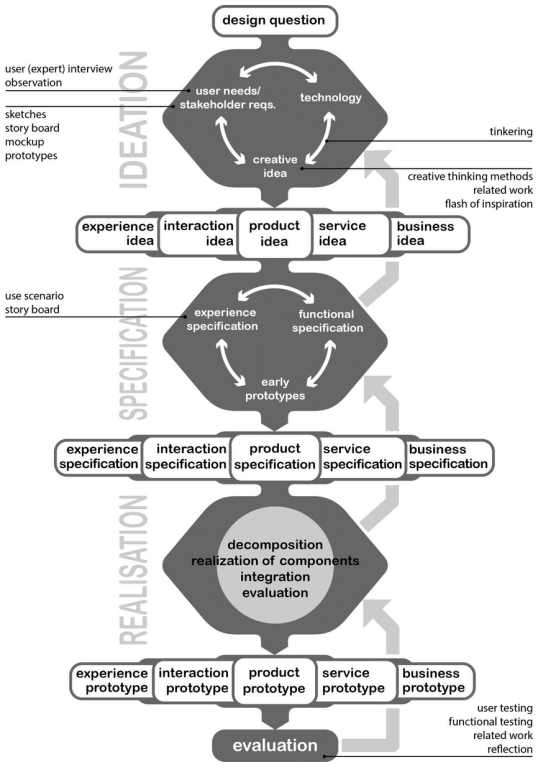



Figure 4: Creative technology design process diagram

### 3.1.1 Ideation

The ideation is used to come up with requirements that need to be filled. This is done by looking at stakeholders and see what they want. Most requirements were already known at the beginning of this graduation project. Those being the budget that a sensor node has and the minimum quality the four sensors need to have (accuracy). From this, some sensors that fulfil these requirements will come forward and the best combination will be chosen. The same goes for the way the sensor node housing will be designed.

Of course, some other requirements will be found during the ideation phase. Since there are some more basic requirements that need to be solved.

### 3.1.2 Specification

The main goal of the specification is describing the final requirements this graduation project aims to solve. Besides the final requirements, a basic overview of the system will be made of how everything is going to be connected to each other. But also what type of components will be in the system on both the hardware side as well as the hardware side.

### 3.1.3 Realisation

In this phase, the system is going to be built with components chosen that fit within the requirements chosen by the specification. Multiple tests will be done in this phase and the problems encountered during this phase will be addressed in the next iteration,

### 3.1.4 Evaluation

During the evaluation phase, the requirements will be tested. To see if they are met in the design. This means the components, software and the sensor node housing. The chosen sensors will be compared to Davis instruments vantage pro 2 [15]. The sensor housing by simulating some weather conditions.

## 3.2 Stakeholder identification

When designing a product not only the end-user is important. People that are affected during development are also important and how they influence the design en development of the product. The main stakeholder will be the one that gives the assignment and will have the most influence. But other parties that are involved also have some influence. For example, the party that the budget, or people that do not buy or want the product but are going to interact with the product. These parties can not be ignored.

## 3.3 MoSCoW

The MoSCoW method[34] is a method where requirements are categorized into four categories: most have, should have and could have. Must have requirements are requirements that have to be in the project. Should have requirements are requirements that have to be in the product if it does not interfere with the must-have requirements. Can have requirements are requirements that are implemented when they do not interfere with the must and should have requirements. And the will not requirements will not be in this project. Since this graduation project requires allocating budget to sensors it is a good way to see how much money should be used for each sensor.

## 3.4 Interview

For this graduation project, an unstructured interview will be used with the main stakeholder Wim Timmerman. The focus will be on what sensors he would prefer, and if some extra requirements are needed for the system.

An unstructured interview means that a topic is set beforehand and that questions arise during the interview. This is much like a normal conversation and can help stimulating more out of the box requirements or ideas.

# 4 Ideation

## 4.1 Stakeholders

To know who is going to be involved and has interaction in the project, it can be good to have a stakeholder analyze. And by knowing how much power those stakeholders have in the project it can also help to sort the ethical by importance.

### 4.1.1 ITC

The ITC is the faculty of Geo-information science and earth observation of the Univerity of Twente. It wants to use the nodes to make a model for accurate weather predictions in the city of Enschede. This is why they determine the specifications of the sensor node. Since they want to use the data of the sensors node, to do research on the temperature build-up in Enschede. Due to this, they have the most interest in this project and also the most power.

So any problems that involve the ITC, must have a high priority. However, they are not responsible for funding. This means that they have little to say about the budget of this project.

### 4.1.2 Twente47

The sponsoring will be done by twente47, which is an IoT accelerator for projects in Twente. Their goal is to make Twente a leader in the area of IoT. Since they provide the budget for the project they have a lot of power in the project. This means that if the problems are against their views they could stop the project until a new sponsor is found.

So just like with the ITC any problems that Twente47 has also have a high priority. So listening to them is important, since they also have experience in similar projects. So it is also smart to listen to them because of their experience.

### 4.1.3 The city of Enschede

Since the nodes and research will be done in the city of Enschede they are a big stakeholder in this project. They, of course, have an interest in the nodes since it can be beneficial to have accurate weather data and predictions in the city. It can help them with where to plant trees to provide shade. Or other measures in building to counter heat build-up, since that happens in cities. This is the so-called urban heat island effect, and this effect can be minimized with certain building styles.

They have a lot of power because if they do not allow sensor nodes to be places in Enschede, the project can not take place. So if the nodes are going to be an eyesore they probably will not like it to be placed all over the city. And

they of course also listen to any complaints the inhabitants of Enschede could have with the project.

#### 4.1.4 Inhabitants of Enschede

Another big stakeholder is the people that live in the city of Enschede. Since they can come in contact with on a daily basis. And they also are going the results of the nodes since one of the goals to improve Enschede. They also are with a lot of people so the chance that they have problems with the finished result of the projects is also bigger.

So it is also important to account for the effect it is going to have on them. And if the node does damage it will likely involve these stakeholders the most. Even dough they have the least power or direct interest in the project. They probably are an important group to analyze from an ethical point of view. But more so the finalized product and not the development.

### 4.2 Housing

When designing the housing some key factors have to be taken into account. Especially when the final sensors are not necessarily chosen. These factors are maintenance, prototyping and flexibility.

Maintenance is important to keep in mind since the sensors will run outside for a long time. Since they are outside and various weather conditions, some part may get broken. And if that is the case it would be inefficient to replace all components while only one is broken. So a design where a single component can be easily replaced is preferable.

Prototyping, since the final sensors are not yet determined. So a design where a single sensor can easily be replaced can save a lot of work. Since redesigning a complete node just to change one small part takes a lot of design and 3D printing time.

Finally, flexibility, since the goal is not one but a whole network of nodes. The situation could occur that sensor nodes with multiple configurations are deployed. This way a few sensor nodes could have more expensive sensors while others some cheaper sensors. So a design that can accommodate both with changing much to the design saves designing time and allows for easy changes to a deployed node.

This is why the design of the sensor node should be modular. So the main housing that houses the controller and power management. But the ability to attach the various amount of sensor with having the make big design changes.

#### 4.2.1 Central housing

The main part of the system is going to be the central housing. This will house the microcontroller or the brains of the system. Connected to this is are the

systems that send the data over to LoRaWAN, the circuits for the sensors and the circuit for the power management.

Since the housing must be able to easily connect sensors to the main controller. Some connectors have to be on the outside so no soldering work is required. And for the sake of prototyping more connectors than sensors should be there. Since internally a digital sensor needs a different connection to the controller than an analogue one. The connectors should not compromise the waterproofing of the housing.

Not only do the sensors have to be able to attach there wiring to the controller. They should also be able to attach physically to the main housing. And this also needs to be done with compromising the waterproofing of the main housing or the sensors. And the parts of the connections on the main housing should not have to change when a different sensor is used. Since then it would have to be 3D printed again.

There are a few ways to have connections to the main housing with having the screw indirectly in the main housing. The first one is a sliding mechanism that is open on top and closes on the bottom. This way gravity will keep it in place and no screws are needed this can be seen in Figure 5.

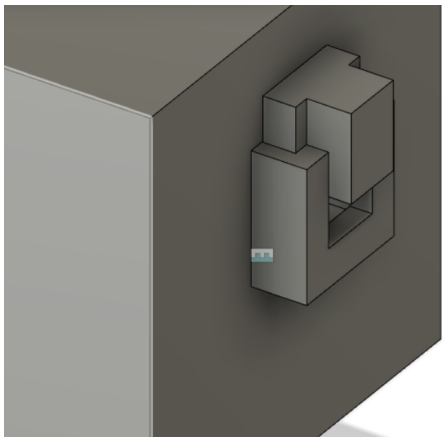


Figure 5: slider sensor Connection

A second way is a sort of hinge mechanism that is connected with a nut and bolt. A single has the ability to be adjusted and when the tightened stay in that place. This can be handy for sensors that need to be directed directly at the sun or a Solar panel. This type can be seen in Figure 7. Or with two holes so it stays in a non-adjustable position perpendicular to the housing. This is a stronger connection and does not move and this type can be seen in Figure 6.

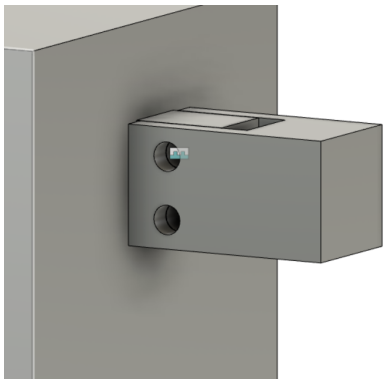


Figure 6: double hinge sensor Connection

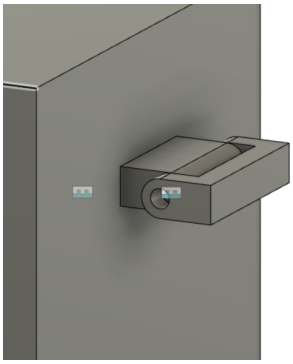


Figure 7: Hinge Sensor Connection

### 4.2.2 Radiation shield

The radiation shield is the part of the node that will house the temperature and humidity sensor. Special housing is needed for these because the sensors need to be kept safe from the weather elements. However, in order to get reliable data from these sensors, there has to be air circulation. This will be done with a radiation shield since that is designed specifically for that purpose.

The radiation shield that is used most for weather stations consists of a stack of hollow rings that are slanted downwards like in Figure 1. There is a gap between each ring so air can get trough but rain can not go inside. There is a different design that is a helix and made in one piece like in Figure 2. However, this is not widely used and some tests can be done to see differences.

### 4.2.3 Attaching the external sensor

Since modularity is key the sensors cannot be directly soldered to the microcontroller. And jumper cables are not exactly sturdy or waterproof. This is why every sensor will be connected with the use of a connector. This connector can be placed on the outside of the node so it is easy to change a sensor. An example of such a connector is made by Weipu[35], the Weipu SP13 is fairly small, waterproof and available in multiple connection types.

Since the sensors that are going to be attached do not all need the same amount of connections. Some only need a plus and minus connection like the solar panel. But some sensors use a digital protocol or have two analogue connections. And they use four different wires and thus four connections. By using different connections amount is will also be less likely that sensors are going to be connected to the wrong microcontroller io pins. It will mean that some connections are redundant.

## 4.3 Sensors

The node is going to have multiple sensors attached so it is able to measure all the different types of parameters. Since the quality of data is important choosing the right sensor is important and the research for that has been done in the state of the art. From the research, some sensors have been chosen that are attached to the node.

### 4.3.1 Power consumption

Being able to run day and night autonomously is not only determined by how much energy the node generates and can store. The power consumption is also very important and there some tricks to reduce this. The first one using an efficient controller so instead of a computer or raspberry pi a microcontroller. This is not as fast is more efficient in small tasks like reading sensor data.

The second is only using the controller when something has to be done like measuring or data sending. Since data does not need to be measured constantly but once every minute for a few seconds. This means that the rest of the time node can go in a deep sleep mode that consumes a lot less power.

The last trick is to make sure the sensors that are attached do not consume any power. Since when having three or four sensors attached and all of them use a small amount of power it will add up. A way to make sure that the sensors do not consume any power, a switch can be used. The microcontroller can switch the power to the sensor when leaving deep sleep and turn it of when entering deep sleep. The switching can be done with a relay, MOSFET or transistor.

## 4.4 Communication

Since the node is functioning autonomously the data has to send a wireless to a receiver node. There are multiple forms of wireless communication and it is key to choose the right one. Since they can vary in range, power consumption, and data transfer rate.

### 4.4.1 Wireless

There are three options when choosing wireless communication those being WiFi, IoT band 4G and LoRaWAN. The first way is WiFi it is widely spread and openly available in Enschede due to the "Enschede stad van nu" WiFi network. WiFi is also reliable since a lot of conformation is done when sending data. But it uses a lot of power, and in more rural areas may be not available. The network is also used a lot which could cause issues.

The second wireless communication method is IoT band 4G. The network has good coverage and high data rate. There also are a lot of conformations in place when sending data. And the possibility to route data to the correct place. But just as with WiFi the power consumption is fairly high.

The last option is LoRaWAN, the conformations with sending data are not as good as the other two options. The data rate is also not high but the amount of data that has to be sent is also low. And a big advantage of LoRa is that it has a large range so getting good coverage is fairly easy. The power consumption is also the lowest of the three communication options.

### 4.4.2 Protocols

The data that can be sent over the TTN network is limited because it can be used by everyone and if too much is send it could block someone's data. So the data has to be sent in an efficient way without losing quality. So sending the different data types in the same order with the same amount of data per packet can ensure this. So when it is decoded by the ttn the data can be parched correctly. If only averages are send every 30 minutes this should not be a problem and every parameter can start with a character. The data limit might become an issue when the min and max of that half-hour also have to be sent.

## 4.5 Microcontrollers

There are a few microcontrollers that can be used to send LoRaWAN data and readout sensor data. They can differ in compute power, energy consumption or features. The microcontroller of this project must be able to read I2C data since a lot of sensors use this protocol. The analogue reading will also be done with an I2C board. The controller also must be able to have a LoRaWAN chip built in to send data. And it would be nice if it is widely used and not too expensive so problems can be solved with the IoT community or the microcontroller supplier.

### 4.5.1 ESP32

When researching two microcontrollers filled in the criteria the ESP32 and Sodaq One. both can be used in the Arduino IDE which was handy since I was familiar with that environment. The ESP32 is the cheaper of the two and is more used in the DIY community. It also has the benefit of having a Wi-Fi and Bluetooth module. They can be turned off, but Wi-Fi can be useful to update the software when sensors are exchanged. The ESP32 also can save data in non-volatile memory meaning that if the batteries are completely empty data can still be saved for when energy is available. However, it can be difficult to find ESP32 modules that have the LoRaWAN module attached. They are available in china but that can be unreliable and take a long time.

### 4.5.2 Sodaq ONE

The benefit of the Sodaq One is the better availability by a more professional company. Which means better support from the company self. However, the price is higher and it is not as widely used in the DIY community. So in most cases, support from Sodaq is necessary of something strange happens. Storing things non-volatile is also not possible, so when power is lost data will also be lost. The Sodaq One also does not have a Wi-Fi chip so updating software has to be done with a cable.

### 4.5.3 Data processing

The microcontroller has to process the data, this can be digital or analogue. Analogue could be done directly which both the Sodaq One and the ESP32 can do. Or with an external board that sends the data digitally with I2C to the microcontroller like the ADS1115[36]. It has a higher resolution and can focus on one task.

For now, only digital I2C data and analogue data have to be read but it can be good to support more than those. Because in the future other sensors are used that do not use I2C or analogue signals it has to be able to support those.

# 5 Specification

This chapter tells everything about the specification phase. It will contain the final requirements and chosen ideas from the ideation phase. And also an overview of how the system will be put together. This is shown with a system diagram for the hardware. The software will be explained.

## 5.1 Basic system overview

Figure 8 gives a global overview of what the sensor node is going to do. This means that the Sensor node gets input from its environment and collects data from this. In the case of the sensor node the air temperature, relative humidity, solar radiation and wind is measured. And solar radiation is not only used for measuring but also for power. The node then sends the measured values to a database so the data of all nodes can be processed. The data go to the database via the TTN network. This means the sensor node sends the data over LoRaWAN to a TTN gateway. This will get the data on the internet where a server can access the sensor data. The data will then be put in a GIS database so data can be analysed based on location.

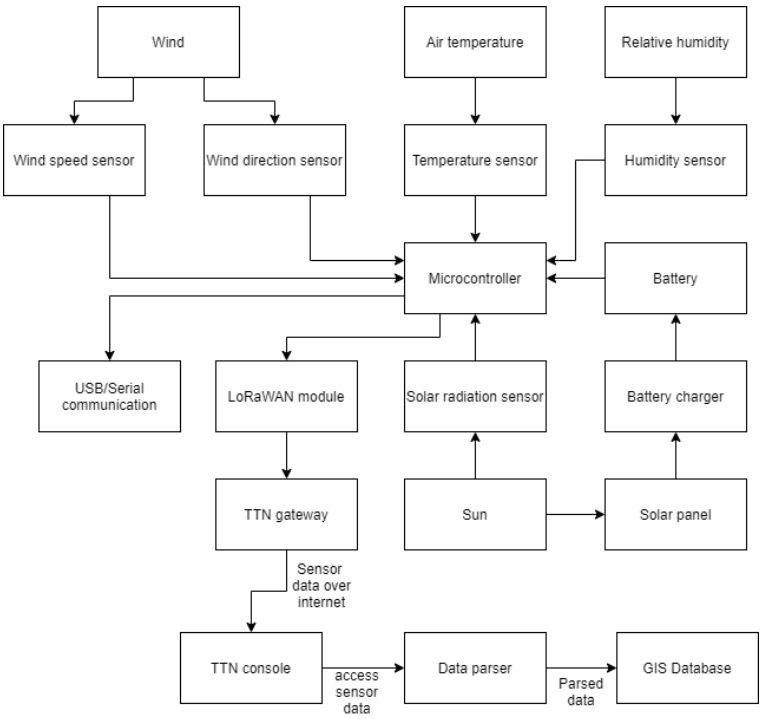



Figure 8: A basic overview of the system

The data collected from the sensors can be sent with an analogue and digital signal. If the sensor sends an analogue signal the data processing is done in the controller. When a digital sensor is used the controller only has to read the values that are already processed by the sensor. They're also a USB cable connected to the controller. This is necessary for uploading code, but also for debugging. Since the controller only sends average data to the TTN from 30 measurements. So to find weird values and to debug it is good to read every measurement that is done via the serial monitor.

## 5.2 Software overview

The software of the sensor node is also designed with modularity in mind. So the software is build-out of a few functions. Every function belongs to a category. The categories are temperature and humidity, solar radiation, wind speed, wind direction, battery, communication and a main.

The main consists out of two functions the setup and loop. The setup is used to setup op every sensor or communication system. In the loop, the actual measurements and data communication are actually done. Most of the time no data communication over LoRaWAN is done. Since the system will measure 30 times before sending. So the LoRaWAN module is only turned in in the 30th loop. This is tracked with a counter variable. At the end of the loop, the system is put in a deep sleep. This means that the power to the sensors is turned off of the controller goes in a standby mode. All the other functions are called in the main, but since they are separated from each other it is fairly easy to change just that part of the code when a different sensor is used.

## 5.3 Final Requirements

In the end, a list of requirements has been made sorted with the MoSCoW method as can bee seen in Table 4

Table 4: Final MoSCoW requirements

| Must Have | Should have |
|---|---|
| The system must be able to function autonomously | The system should be able to remain data when power is lost |
| The system must be able to function in freezing conditions | The system should be able to measure every minute and calculate the average value every 30 minutes |
| The system must function outside in the rain | The system should be able to modular very easy sensor changes |
| The system must be able to function 24/7 | Could have |
| The system must be able to send data every 30 minutes to TTN | The system could have OTA software updates |
| The temperature sensor must have an accuracy of 0.3-0.5 °C | Will not have |
| The humidity sensor must have an accuracy of 3% | System will not be able to charge when it is freezing |
| The wind speed sensor must have an accuracy of 0.1 - 0.2 m/s | |
| The solar radiation sensor must have an accuracy of 20 w/m2 | |

# 6 Realisation

The final prototype that has been developed with the help of the requirements will be described in this chapter.

## 6.1 Components

In the end, the system consists of a lot of components. This entails sensors, energy system and controller. All these together form the functional part of the sensor node. The way everything is connected to each other can be seen in Figure 9. What connections and colours are actually used can be seen in Table 5.

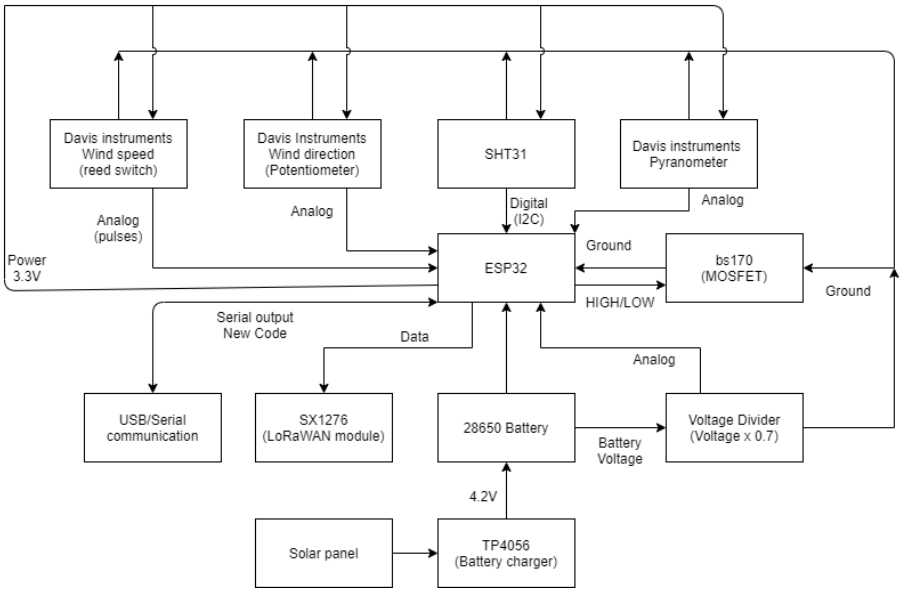



Figure 9: slider sensor Connection

Table 5: All the connections from the sensors to the ESP32

| Connection Sensor | Wire colour from sensor | Connection ESP32 | Wire colour to controller |
|---|---|---|---|
| SHT31 | | | |
| SDA | Green | Pin 21 | Orange |
| SCL | Blue | Pin 22 | Yellow |
| Vin | Red | 3.3V | Red |
| Ground | Black | ground | Black |
| Davis Instruments Pyranometer | | | |
| Analog out | Green | Pin 34 | Green |
| Vin | Yellow | 3.3V | Red |
| Ground | Red and Black | ground | Black |
| Davis Instruments Anemometer | | | |
| Wind Speed Out | Black | Pin 36 | Blue |
| Wind Direction Out | Green | Pin 35 | Purple |
| Vin | Yellow | 3.3V | Red |
| Ground | Red | ground | Black |
| Voltage divider battery voltage | | | |
| Divider out | Grey | Pin 39 | Grey |
| Vin | Red | + Battery | NA |
| Ground | Black | Ground | Black |
| SX1276 (LoRaWAN) | | | |
| nss | NA | Pin 18 | NA |
| rst | NA | Pin 14 | NA |
| dio | NA | Pin 26, 32, 33 | NA |
| bs170 | | | |
| gate | White | Pin 23 | White |

### 6.1.1 Davis instruments Anemometer

The first component of the system is the Anemometer from Davis instruments [25] as can be seen in Figure 10. This sensor measures two parameters. The wind speed and wind direction and they output there data on separate wires. Both will be discussed separately since they work quite differently.

Figure 10: Anemometer from Davis Instruments

Wind Direction

The wind direction part of the sensor is the easiest of the two sensors. The Wind sensor is a potentiometer that is turned in the wind by rotating a wind vane. The code is fairly simple and is just mapping the analogue value from 0 - 360 degrees. As can be seen in appendix A.7.

Wind Speed

The second sensor in the anemometer is the wind speed sensor. This sensor works quite simple but the processing of the signal is a bit more difficult. The sensor itself is a reed switch connected to the input power with a 1K resistor. Every time the sensor has made a turn the switch opens. This means that the analogue value received by the controller is usually maximum and in this case 4095. And when the switch is opened to value is zero. The wind speed sensor does not quite fit within the requirements set in Table 1. But sensors that did were to expensive for now, and this sensor is also the one that is tested against.

The code is a bit more difficult A.8 and works by counting the pulses for 2.25 seconds. However, the value stays zero for more than one measurement which means that a pulse can be counted again only after the analogue value has returned to 4095. This is done with some sort of smith trigger in code. It uses some a Boolean that is true and set to false when a pulse is detected. The Boolean is set to true when the value is high again. The conversion from pulses

to wind speed is fairly simple $V = P(2.25/T)$. In this equation V = speed in mph, P = the number of pulses per sample period and T = sample period in seconds. This means that the amount of pulses measured in 2.25 seconds is the wind speed in mph. To get the wind speed in kph this value has to be multiplied with 1.61.

Besides the code for measuring the wind speed, the wind speed measure function also calls the function that measures battery voltage, temperature, humidity, solar radiation and wind direction. This is because those value can be measured almost instantly. So once the 2.25 seconds of measuring is done for wind speed the other parameters are measured.

### 6.1.2 Davis Instruments Pyranometer

The Pyranometer from Davis Instruments [26] is the sensor that measures the solar radiation. This is a photodiode with an amplifier that outputs a digital signal that has that can be converted to a value in W/m2 with the simple formula $solarradiation = mV/1.67$. These components are then put in a waterproof housing to protect them against the rain which can be seen in Figure 11. The accuracy of the pyranometer is given in percentages, and not in absolute values so it does not always meet the requirements given in Table 1.

Figure 11: Pyranometer from Davis Instruments

The code what is shown in Appendix A.4 is almost the same is the one used for wind direction. The analogue value that is received by the ESP32 is first divided by 3300 to get the amount of mV that is received. This is because the maximum value the ESP32 can receive on an analogue pin is 3.3V. That value is then divided by 1.67 to get the amount of W/m2.

### 6.1.3 SHT31 Temperature and Humidity

The last sensor attached to the sensor node is the SHT31 [10] as seen in Figure 12. Since this sensor is digital all processing is done by the sensor self. The sensor outputs the values over the I2C protocol to the ESP32. The I2C protocol uses two wires for power and ground, but also two wires for communication one for the clock (SCL) and one for the data (SDA). This sensor fits within the specifications given in Table 1.

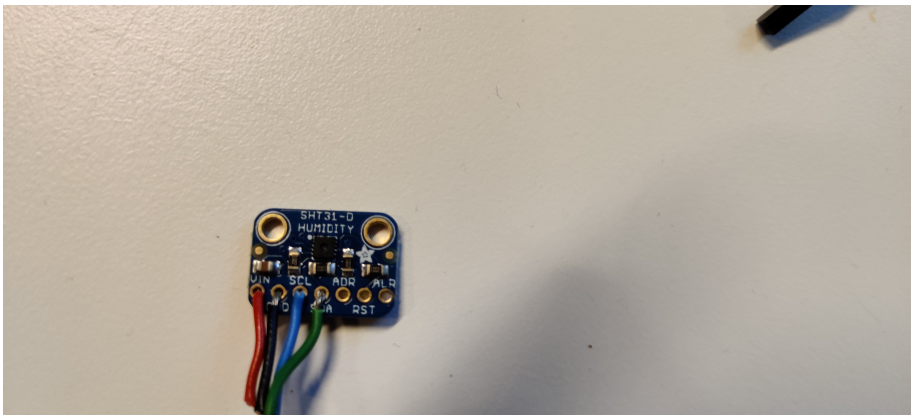



Figure 12: SHT31 temperature and humidity sensor

### 6.1.4 Power management

The power management system is built out of a couple of components; solar panel(s), Li-ion 28650 battery, tp4056 and a voltage divider. Together with the components supply power and battery management. The solar panels used Figure 13, supply a peak voltage of 5V but less when the sun is not ideal. They have two wires one ground and one plus. They are connected to the tp4056 li-ion charge controller. This controller board makes sure the health of the Li-ion battery stays good. This means it only charges when the solar panels deliver 4.2V or step the voltage down to 4.2V. But also ensures they are not discharged when they drop under 2.5V.

The tp4056 is connected to the battery and the ESP32. The Battery is also connected to a voltage divider that multiplies the voltage of the battery with 0.7 since 4.2V is to high for the analogue pins of the ESP32. This way the battery level can be measured to check on the power of the system.

All the sensors and the voltage divider above have there ground connected to a MOSFET (bs170). This is set up as a switch so that those components do not use power when no measurements are done. When the gate pin of the MOSFET is pulled to ground the switch is open and the sensors are then turned

off. This way when a voltage is applied to the gate power can flow and the sensors turn on and can measure.

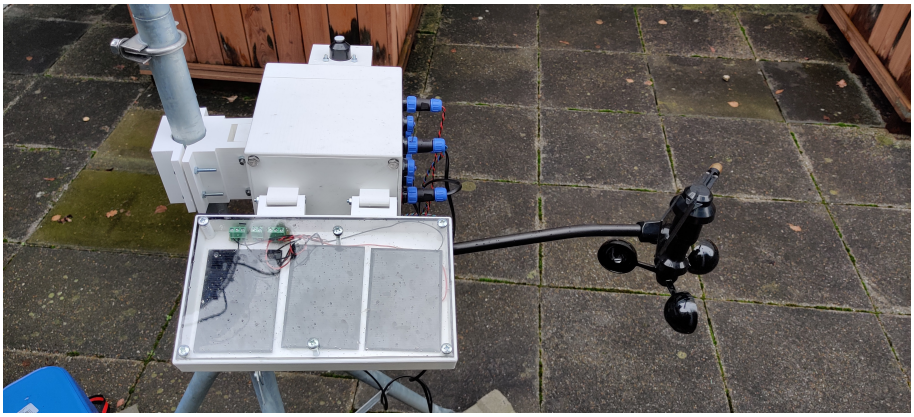


Figure 13: Solar panels

### 6.1.5 ESP32

The ESP32 with an integrated LoRaWAN chip is chosen to control all the components. All the code that is running on the ESP32 can be seen in Appendix A. Due to some problems with the TTN gateway WiFi was used for some of the tests. This is why two main code parts are in the appendix since they function somewhat different. The code on the ESP32 is set up in such a way that the code that reads out the sensors is separated and can be easily adapted or completely changed when a different sensor is used. Between cycles as described as in 5.2 Software overview, the measured sensor values are stored in the flash memory. This is one of the reasons the ESP32 is chosen over the SODAQ ONE. This means that when power is completely lost to the microcontroller measurements are not lost.

### 6.1.6 Complete system

The complete system should have a prize of around € 400,- and the complete cost now is € 435,45 (Table 6). But officially that budget is for the sensors and those together are below € 400,-. The sensors together cost € 300,86 And at the moment most Weipu connectors are redundant so of the redundant connectors are removed everything fits in the budget.

Table 6: Prize over view sensor node

| Description | amount | Price / piece | total |
|---|---|---|---|
| Sensiron SHT31-D | 1 | 14,17 | 14,17 |
| Davis Instruments Pyranometer | 1 | 156,98 | 156,98 |
| Davis Instruments Anemometer | 1 | 129,71 | 129,71 |
| resistors (1k, 43k, 100k) | 1 | 2 | 2 |
| bs170 mosfet | 1 | 0,10 | 0,10 |
| 3x 5V solar panels | 3 | 5,04 | 15,12 |
| TP40561A Lithium Battery Charging | 1 | 2.65 | 2.65 |
| Samsung ICR186500 | 1 | 9,75 | 9,75 |
| Samsung 3.7V 18650 Lithium-ion Battery, 2600mAh | 1 | 7,91 | 7,91 |
| lilygo ttgo esp32, | 1 | 12,- | 12,- |
| Breadboard wish big with power lane(s) (73x142 mm) | | | 9,92 |
| Wires in various colours | | 5,- | 5,- |
| 4 polig cable connector weipu (3x) | 2 | 4.18 | 8,36 |
| 3 polig cable connector weipu (4x) | 1 | 4,18 | 4,18 |
| 2 polig cable connector weipu(2x) | 1 | 4,18 | 4,18 |
| 4 polig node connector weipu (3x ) | 3 | 4,18 | 12,54 |
| 3 polig node connector weipu (4x) | | 4,18 | 16,72 |
| 2 polig node connector weipu (2x) | 2 | 4,18 | 8,36 |
| protective cap node weipu connector | | | 2,95 |
| - plexiglas 4mm | 1 | 5 | 5 |
| - seal material (Black Natural Rubber Sheets) | 1 | 5 | 5 |
| - PLA | 1 | 22,50 | 22,50 |
| Total ex btw | | | 435,45 |
| Total incl btw | | | 526,89 |

## 6.2 Sensor housing

The final design of the sensor housing which will protect all the hardware components. The sensor node is made up out of a lot of parts. These parts can be seen in Figure 14 and Figure 15 in a exploded view. The assembled sensor node can be seen in Figure 16 and Figure 17

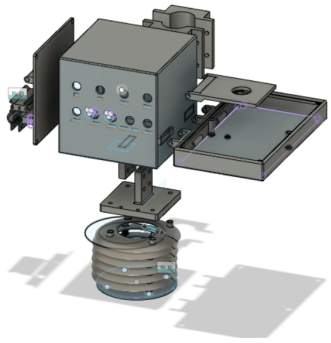


Figure 14: Exploded view sensor node

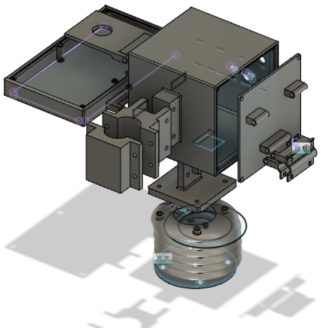


Figure 15: Exploded view sensor node

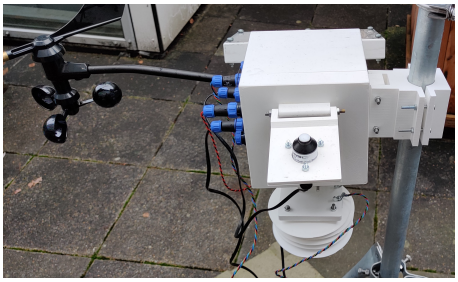


Figure 16: The assembled sensor node housing

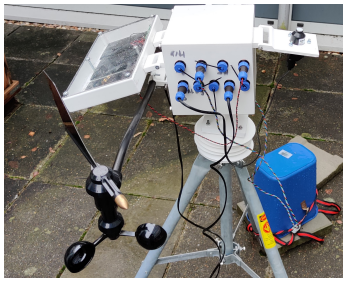


Figure 17: The assembled sensor node housing

### 6.2.1 Main Housing

In the centre is the main housing where the controller is situated plus components to help the sensors or energy management. To be able to be used this housing has to be attached to something and in the case of this graduation project that is a simple pole.

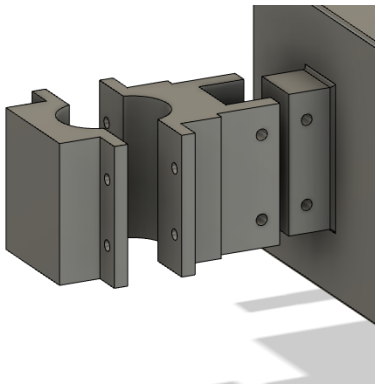


Figure 18: This allows the node to be attached to a pole

Securing the node

This attachment can be seen in Figure 18. It is designed in such a way, that if the node has to be attached to a different type of object. Only the two parts not connected to the node have to be redesigned. And the current mounting method works by putting a bolt through the holes on the node and middle part and securing it with a nut. And to actually attach it to a pole put a nut and bolt combo trough the holes on the two left parts. By really tightening it together it will stay in the pole.

Connectors

As can be seen in Figure 19 the sensor node housing has a few holes. These holes allow the Weipu [35] to be attached to the sensor node. The holes have to fit tightly between the two parts of the Weipu connector. If that is not the case water can get in the node and damage the components.

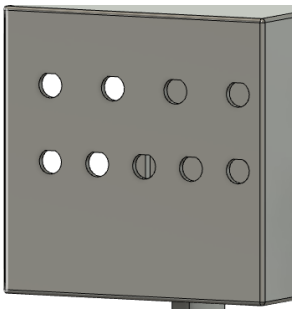


Figure 19: Holes where Weipu connectors can be put trough

Lid

To be able to put the components in the sensor housing there has to way to access the inside. This is done with some kind of lid as can be seen in Figure 20. Since the housing has to be waterproof with the lid attached there are a few important things in the design. The first one is a small grove on the inside of the node, in this grove a rubber seal [37] can be placed so no water can enter. The second one has to do with the bolt that secures the lid in place. On every bolt, there has to be an o-ring so no water can seep through the hole where the bolt is.

Figure 20: The lid of the sensor node

### 6.2.2 Attaching components

Many components need to be secured to the sensor node, with the option to replace them easily. Without having to redesign the whole housing.

Attaching the Pyranometer

The Pyranometer uses a single hinge design to attach to the sensor node as can be seen in Figure 21 and Figure 11. The sensor itself is mounted to a sort of plate that clamps on the main part. The hinge part connected to the main part is printed separately and glued with PVC glue. This is done to make 3D printing easier since that side can be used as a plat starting surface. Since a single hinge design is used there is some degree of movement so the sensor can be mounted directly at the sun.

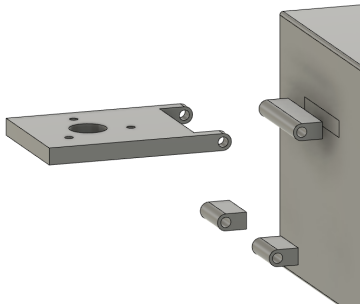


Figure 21: The lid of the sensor node

### Attaching the Wind Sensor

Attaching the wind sensor is similar to how the sensor node is attached to a pole. Only the pole is turned 90 degrees since that is how the sensor needs to be orientated. The mounting mechanism can be seen in Figure 22 and Figure 10. However, the hinge self is a single hinge is it can rotate somewhat to make sure the wind sensor is oriented correctly.

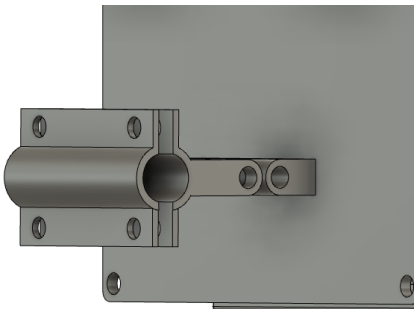


Figure 22: The lid of the sensor node

### Attaching the Temperature and Humidity sensor

The temperature and humidity sensor has more components than the other sensor. This is because it needs to be protected from the rain. But to get accurate air temperature data senors does need airflow. The radiation shield allows for this and the design can be seen in Figure 23 and Figure 24.

The rings are separate objects and are attached with a piece of wire steel and two bolts. At the bottom of the radiation, shield is a closed piece to protect the components. At the top is a piece the allows the SHT31 [10] to be secured and the shielding to be attached to the main part of the housing. This connection is a bit different than the other connection and does not really use a hinge but two plates that can be bolted together.

Figure 23: Picture of radiation shield

Figure 24: 3D model of solar radiation shield

Attaching the Solar Panel

The solar panels are placed in a separate housing that is attached to the main part of the housing. Just as with the wind sensor and pyranometer this is done with a single hinge. This allows the solar panels to be angled for maximum solar power. The housing where the panels are placed in is a box that is open at the top. And just as with the lid part of the main housing a small grove for some rubber is made. This is to make the housing waterproof that is why there are also o-rings used on the bolts. The top is Plexiglas so the sun can enter the housing while protecting the components. The designs can be seen in Figure 13 and Figure 25.

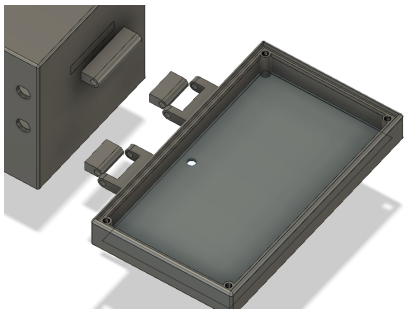


Figure 25: The lid of the sensor node

# 7 Evaluation

In this chapter, the sensors from the build sensor node will be tested. Both to see if they work and give sensible data and in comparison to the Davis Instruments vantage pro 2 [15]. Besides the sensors, the housing will also be tested to see if it meets the requirements.

## 7.1 Component test

The first test is testing of the components, this is without the final housing and not compared to the vantage pro 2. The setup can be seen in Appendix C.1. The data is gathered from the things network (TTN) with the help of a python script shown in Appendix B.1. In this test the parameters air temperature, relative humidity and solar radiation will be examined since the place where the measurements are done is closed off and has no wind. The values are not yet the averaged like with measurements done later on. The measurements took place on 27th of November from 10:06 until 8:40. somewhere after 8:40, there was some water damage in the temporary housing and the controller broke. These measurements were done with the SODAQ ONE instead of the ESP32 [38].

### 7.1.1 Temperature

The first parameter that will be evaluated is the air temperature. As can be seen from Figure 26 the data gathered from the SHT31 [10] is data that can be used. Since it makes sense that the temperature decreases as night, which is seen in Figure 26.

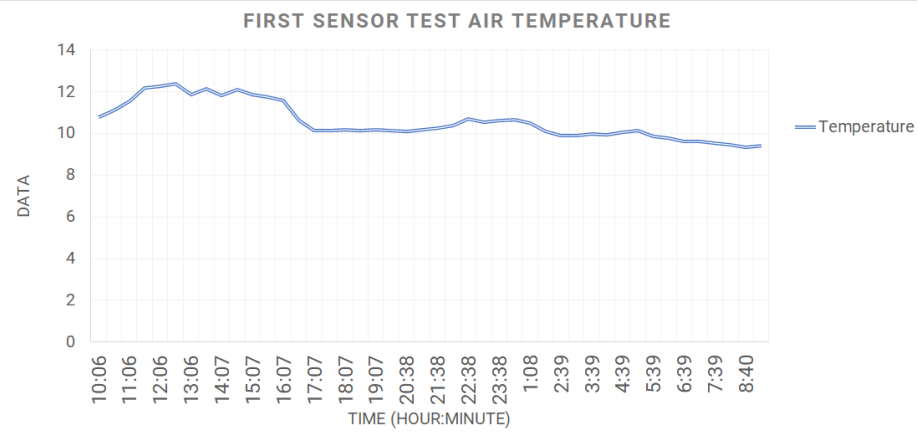



Figure 26: Temperature data from first sensor test

### 7.1.2 Relative Humidity

The second parameter is that is going to be evaluated is the relative humidity. Since it was rainy the air was quite humid so then it makes sense that the humidity was at least 80% as can be seen in 27.

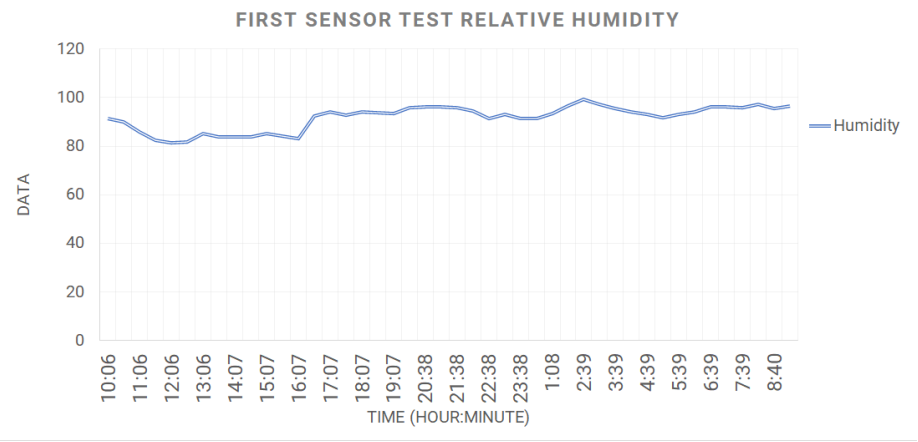



Figure 27: Humidity data from first sensor test

### 7.1.3 Solar Radiation

The last parameter that will be evaluated is solar radiation. This makes not as much since it would be expected that at night there is no solar radiation. It does get lower but is it still quite high as can be seen in Figure 28. So there might have been some problems in the processing of the pyranometer sensor.

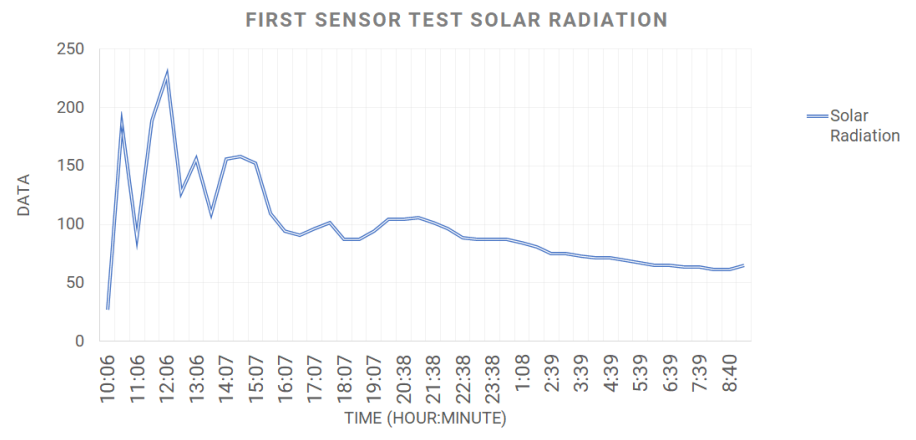



Figure 28: Solar radiation data from first sensor test

## 7.2 Test outside

The seconds test is with the complete system, pictures of the system can be seen in Figure 16 and Figure 17. The test is done on the 27th of January from 12:23 to 17:34. Every thirty minutes of data sent to a MySQL database that runs on a raspberry pi. The data is first sent to a WiFi router that then routes the data to a MQTT broker. A python script (Appendix B.2 listens to the MQTT broker inputs the data in the MySQL database. This had to be done since at the time the LoRaWAN gateway was offline so LoRa could not be used. The data from the Davis Instruments vantage pro 2 was collected by filming a time-lapse of the console that displayed the data. This was done manually entered in the database with the help of a python script (Appendix B.3)

### 7.2.1 Temperature

First, the temperature data from the test with the final node outside will be compared. The comparison is between the self-build node and the Davis Instruments vantage pro 2 [15] and the data is shown in Figure 29. As can be seen from data the values measured by the two sensors are close to each other. The only real difference is the resolution, the data from the vantage pro has a resolution of 1.0°C. Meanwhile, the resolution of the SHT31 [10] is 0.01°C, so the highest value of 9.4°C would be rounded to 9°C. The same value as the vantage pro. This means that the SHT31 is comparable to the vantage pro.

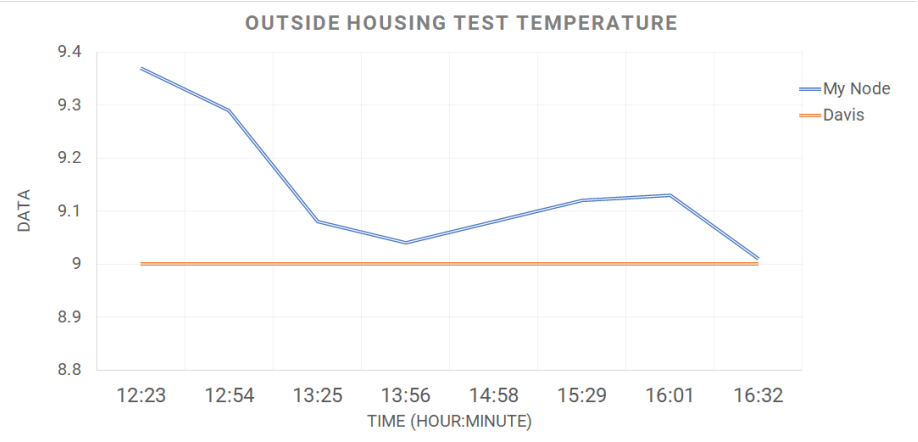



Figure 29: Temperature data from sensor node with housing outside

### 7.2.2 Relative Humidity

The second parameter humidity also is measured with a higher resolution in the self-made node, when compared to the vantage pro. When looking at the data in Figure 30 it can be seen that they follow the same trend, but not entirely the same. So gives the self-build node first a lower value and later a higher value than the vantage pro 2.

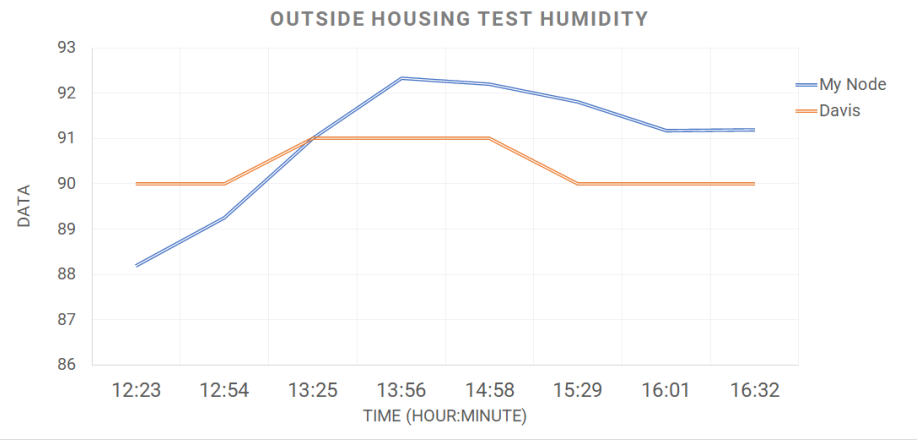



Figure 30: Humidity data from sensor node with housing outside

### 7.2.3 Solar Radiation and Wind Speed

Due to lack of wind, a not much sun in this time of year no useable comparative data came out the test for solar radiation and wind speed. The vantage pro did give some values for solar radiation but very low. So that could have something to do with how the analogue to digital converter. For example that it is non-linear in the beginning and end. And for wind speed was the lack of wind the problem since both the vantage pro and self-made station said there was no wind.

## 7.3 Controlled test

The last test was done in a controlled environment and focused on wind speed and solar radiation. The data is gathered the same way as the last test. In this test, the amount of cycles has been reduced to 5 and the averaged data is uploaded every 5-6 minutes.

### 7.3.1 Solar Radiation

There are three tests for solar radiation, two with a floodlight of 120 watts and one with a halogen light of 650 watts. The floodlight is done with two distances, one where the floodlight is 20 cm from the sensors and one with 35 cm. The pictures from the setup can be seen in appendix C.2.

Pyranometer test with floodlight (20cm)

The values of the two sensor nodes in this test are close together and swing around each other (Figure 31). From this, it can be said that the ESP32 can process the data from the Davis Instruments reliably in the range 240-255 W/m2.

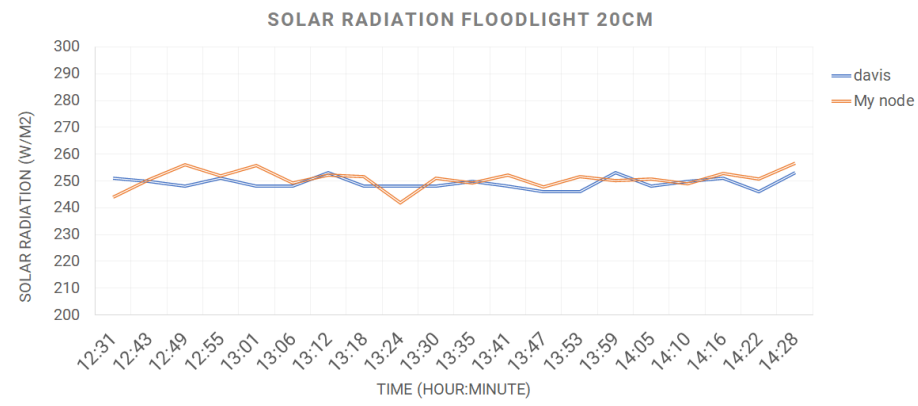



Figure 31: Solar radiation test with floodlight 20cm

Pyranometer test with floodlight (35cm)

The next is done with the same light but a bit further (35cm). The values of the two nodes do follow the same trend but the self-made node is roughly half the value of the vantage pro (Figure 32). This could have two causes, the first one is that the sensors were not exactly the same distance from the light source. Since the sensor is sensitive this can cause this difference. The other explanation is non-linear behaviour in the analogue to digital converter.

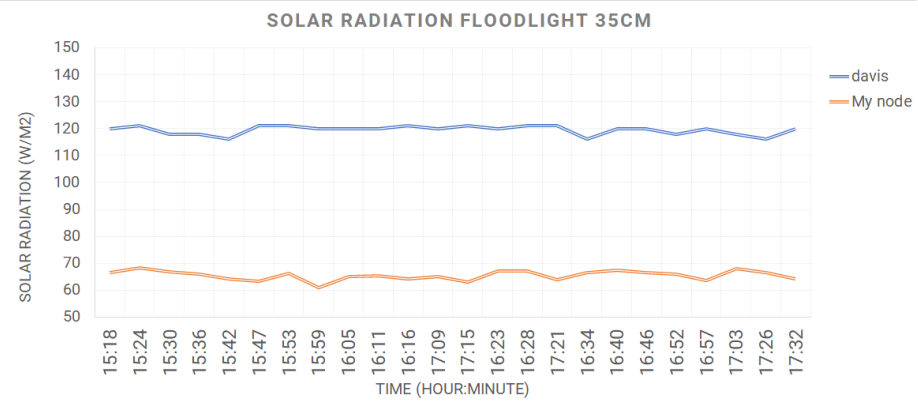



Figure 32: Solar radiation test with flood light 35cm

Pyranometer test halogen light

The last test with the pyranometer is done with a big halogen light of 650 watts. The light source was 30cm away from the sensors and the setup can be seen in appendix C.2. The range of 880 to 930 W/m2 also is nicely in the middle of the total range. And again the measured values are close to each other (Figure 33). Which makes the theory that the analogue to digital converter is non-linear at the beginning of the range very likely. The values also value the same trend although not always exactly the same.

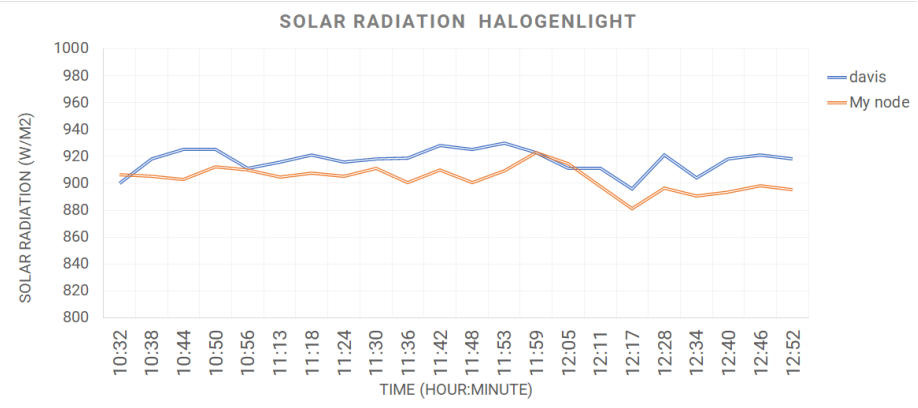



Figure 33: Solar radiation test with halogen light

### 7.3.2 Wind Speed

For wind speed, there were done two tests with a fan, and to make sure the two anemometers turn at the same speed a strobe light was built. Strobe light consisted of a frequency generator connected to the same bright LEDs via a MOSFET. When the frequency of the strobe light matched the frequency of the anemometer it would appear as if the anemometer did not turn. If both anemometers appeared to stead still, they would turn roughly the same speed. But the airflow from a fan is not very consistent so it was not perfect. The pictures of the setup are in appendix C.3

Wind speed test 22Hz The first test is done with the frequency generator set to 22Hz and the two anemometers matching that. When testing they sometimes seemed to stand still but not always. When looking at the data from Figure 34 the values are very close.However, the self-made node has an offset but seems to follow the same trend. This could be due to a difference in wind flow since a fan was used. But it also can be a problem with the processing. Like in the accuracy in the time the controller measures per wind speed measurement.

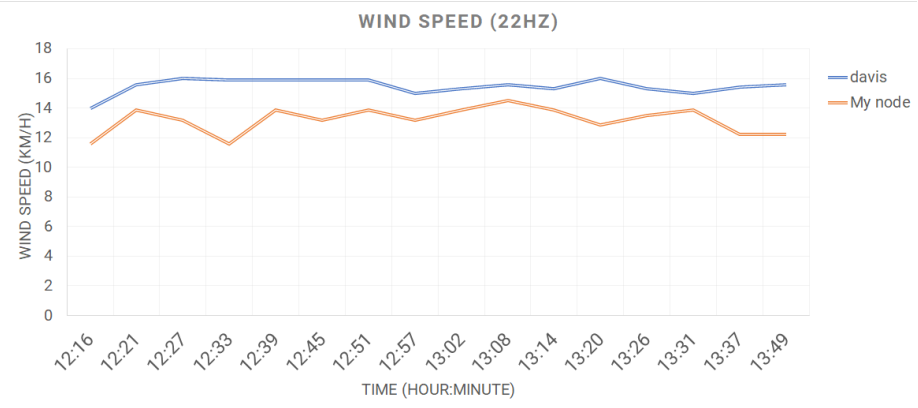



Figure 34: Wind speed test (22Hz)

Wind speed test 39Hz The second test is done with the frequency generator set to 39Hz. And just like with the previous test the speed of the anemometers were placed in such a way that it seemed they stood still. And the data in Figure 35 shows fairly similar values. However, the self-made node again has a small offset down. So it could mean that it measures for less than 2.25 seconds or maybe it is in the setup. It could be something small as a difference in airflow when setting up the sensors. Since I stand behind it when setting up but am not when measuring. The test would have been more accurate if it would be done in a wind tunnel.

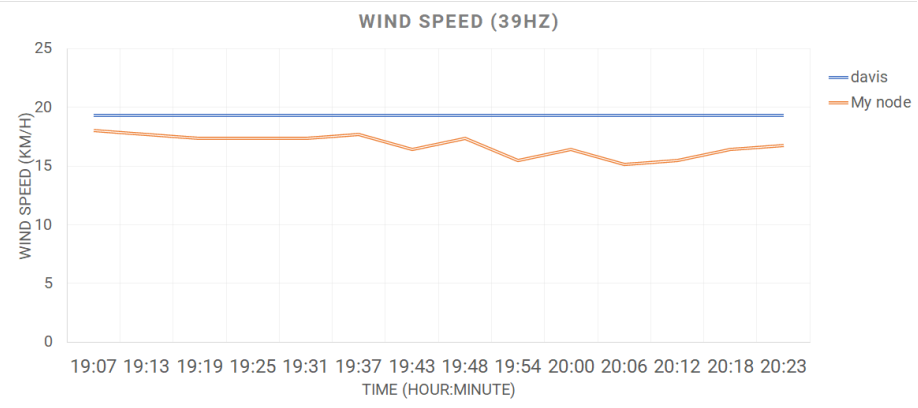



Figure 35: Wind speed test (22Hz)

Temperature data gathered during controlled test

The room where the tests took place are was climate controlled. However, the temperature was not set constant or monitored. But the data gathered from the two sensor does match except for two peaks from the Davis sensor as seen in Figure 36. Unfortunately due to the low resolution of the vantage pro sensor, it is difficult to see how close they really are. This does show that a resolution of 1°C is not good enough to get real meaning full data.

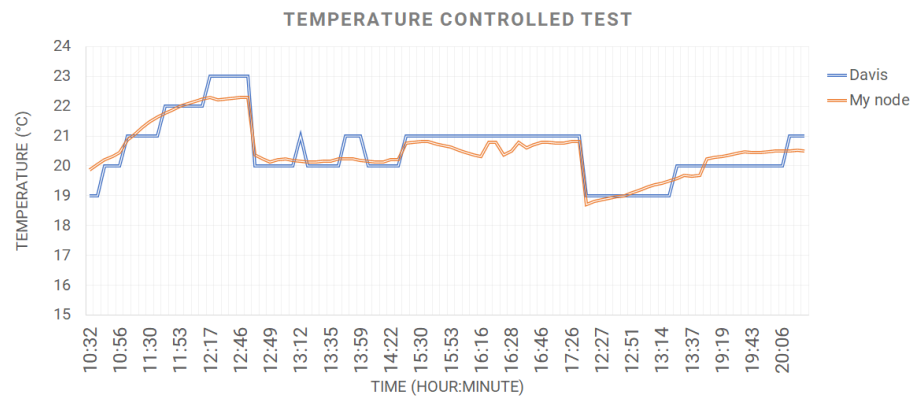



Figure 36: Temperature data from controlled test

Humidity data gathered during controlled test

The same holds for the humidity data gathered from the two sensors. The data is not exactly the same but circle around each other. This can be due to the accuracy of the sensors. But the show the same trend when looking at decreasing or increase of the humidity as seen in Figure 37.

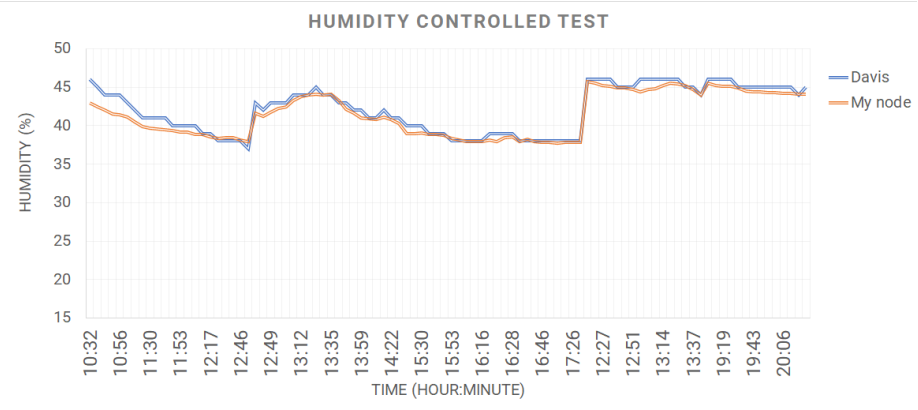



Figure 37: Humidity data from controlled test

## 7.4 Water proofing

The waterproofing of the sensor node has been done with the help of a watering can. And some of the tests have also been done outside in the rain. With both types of tests, no water entered the main sensor housing or solar panel housing.

# 8 Discussion

## 8.1 Waiting on components

A big obstacle I came across was waiting for components since the project required rapid prototyping this was not ideal. The biggest example of this was the Weipu connectors since they had to fit precisely in the main sensor housing. Since this is one of the biggest prints I wanted it to fit the first time. The connectors were ordered before the Christmas holidays but since the university closes around the time they were, send back during the holidays. And after the holidays they still had not arrived and were lost somewhere at the shipping company. This meant that I could not test fit and that the largest and most important print had to wait. This had cost me a lot of time. And for other components, the wait was not that long but since everything had to go through the university it did at a few days.

Of course not only waiting till components arrived added delay, but the 3D printing of components also took some time. Some prints took more than 24 hours and some of them messed up halfway through the print. And since there were so many components that took a week even though I had three to four printers available to me.

## 8.2 LoRaWAN

A big set back came with the final test of the whole system. Since I had to gather the data via LoRaWAN I needed access to a LoRaWAN gateway. During the course of my graduation project, there was a node available that could be accessed on the university. But when I wanted to run the latest test that gateway was offline. This meant that I had to change my way of gathering data from LoRaWAN to WiFi. Luckily I decided during my graduation project that I wanted to use the ESP32 instead of the SODAQ ONE. Since the SODAQ ONE would not have the ability to use WiFi but only LoRaWAN.

Otherwise, I had to switch microcontroller and build a network to collect the data. Now I only had to make my own network with a database to store the data and rewriting some code for the ESP32. It still took some more than a day to set up but it would have taken a whole lot longer if I had to find a new microcontroller and write the code for that.

## 8.3 Water damage

During the first with the sensors without the self-made housing, some water damage occurred. Since the setup was set up with the help of 2 first iteration sensor node housing and a waterproof box with some holes drilled in for wires. The sensor housings were fine but de box that housed the controller had a small leak. This could have been prevented if the hole for the wires were on the sides and not on top. The first time there were a few drops inside and everything was still fine. I added some more hot glue to make everything waterproof. But it still was not since over the weekend a lot of water entered due to a lot of rain. The system died a few hours before a got to the university and a lot of water entered. Luckily only the microcontroller died wich was a SODAQ ONE back then.

The first test after this one I made sure to really test the waterproofing of the housing before putting components in. This was done under the water faucet or with a watering can.

# 9 Conclusion

In this chapter, the research questions will be answered that were set at the beginning of this research project.

How to optimize the quality of measurements?

This sub-question has been solved by making the sensor node modular. Since the time would not allow for big sensor node iterations. This is why I designed the node in such a way that in the future it should be easy to test different sensors.

But on the area of quality measurements, some conclusions can be made. The first one is for the temperature and humidity sensor, the measurements show that the SHT31 has the same data quality as the one from the vantage pro. It might even be higher since the SHT31 as a better resolution. The sensors for wind speed and solar radiation are the same as the vantage pro so the quality of data from the sensors is equal. From testing, it did seem that for solar radiation lower values did deviate from the vantage pro. This could be from the analogue to digital converter or some damage from the water damage after the first component test. As for the wind speed sensor the vantage pro gave higher values across the board which could be from the post-processing but also the test setup. This could be verified in a better test setup like for example a wind tunnel.

But since the SHT31 is the only sensor that deviates from the vantage pro, and the rest still has to be replaced with alternative one can conclude that the temperature and humidity sensors quality of measurements have been optimised.

How to optimise the protection of used sensors and energy harvesting sub-systems?

The protection of the components are protected with a sturdy housing made from PLA. But physical protection is the least important for the sensor node. The most important is being protected against rain. And this was done by making sure that every hole is sealed. Since the node still has to be opened just putting silicone sealant or hot glue every was was not an option. Instead, every bolt that entered the main housing or solar panel housing uses an o-ring. And the lids of the solar panel housing and main housing have a rubber seal [37]. This seal is made from black natural rubber and cut to exact size in a laser cutter. When this seal is placed and the lids tighten correctly, no water can enter the housing.

In the end a system was build that was slightly too expensive, but this was due to the use of redundant connectors. As can be seen in table 6 the sensor node with housing is a bit more than € 400,-. However the sensors together are € 300,- and the sensor budget itself is € 400,- so the sensor budget is met. During most tests the controller was connected with a USB cable to the laptop. This means that no data there is little data on battery life and thus autonomous functionality. The data gathered was autonomous so that the requirements were met. The first test in the housing was outside but quite short, so not much can be said about that.

# 10 Recommendations

## 10.1 Micro controller

The ESP32 that was used for this project has some faults. The first one is that fact it only really is available via aliexpress which is not always reliable. The second might not be an issue but I could not test it. And that is that it has two channels with analogue to digital converters and that ADC2 does not work when WiFi is used. This should not be a problem when LoRaWAN is used but I do not know for certain. There should not be a problem since each channel has more than enough inputs to accommodate all sensors but 3 inputs for channel one will damage the USB controller on something is plugged in. This is probably a problem with this particular ESP32 board version. So one from a different manufacturer can be used and it might not have this problem.

But it might be an option to look at a different controller. When looking at this it is important to keep in mind that must have LoRaWAN, the ability to store values in flash memory and a way to input from battery power.

## 10.2 OTA Updates

Since the whole system is modular and easy to physically change sensors. It would be nice if the software also allows for easy exchange of sensor code. This can be done in two ways, a port in the sensor housing that can access the controller via USB. But if the controller has the ability to use WiFi a button that puts the controller in a state where it turns in to a WiFi access point and allows new code to be uploaded over WiFi. This way of uploading new code for different sensors easy.

## 10.3 Temperature based switch

Since it is bad for the battery to be charged when temperatures reach sub-zero. It would benefit the sensor node system in the long run if a switch is built in that disconnects the battery from the charging circuit when it is freezing. And turns it on when the temperature is big to above zero. The best way to do this is mechanical since this system would be able to function with the microcontroller. Otherwise, the system could shut off permanently if it freezes to long and the battery is completely drained.

## 10.4 Larger production

If the sensor node reaches the phase where 80 units have to be deployed 3D printing is not a feasible solution. Since it could take a week to print all housing parts per sensor node. So finding a production process that is more suitable for a semi-large production is optimal. But since it is still relatively a small scale production an assembly line is not cost-efficient.

Besides the housing, the circuitry between the sensors and the microcontroller could also be replaced with a custom PCB. Instead of a piece of perf board like is used now. But this is relevant when the final sensor configuration is chosen. It would reduce the cable mess that is currently in the sensor housing. Another benefit would be a bit more reliability.

# References


[1] "Hittestress - gemeente enschede.," Oct 2018. Available: https://www.enschede.nl/duurzaam053/klimaatadaptatie/hittestress.

[2] "Zomer 2018 (juni, juli, augustus)." Available at: https://www.knmi.nl/nederland-nu/klimatologie/maand-en-seizoensoverzichten/2018/zomer.

[3] T. Onderwater, *Developing a sensor network for real time temperature monitoring in Enschede*. utwente, February 2018. Retrieved from: https://essay.utwente.nl/74930/.

[4] Maxim Integrated, *Programmable Resolution 1-Wire Digital Thermometer , 19-7487 datasheet*, 2019. Retrieved from: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf.

[5] L. Kester, *Climate measurements in public spaces*. utwente, July 2018. Retrieved from: https://essay.utwente.nl/75748/.

[6] Argent Data systems, *Weather sensor Assembly p/n 80422*. Retrieved from: https://cdn.sparkfun.com/assets/8/4/c/d/6/Weather_Sensor_Assembly_Updated.pdf.

[7] Senserion, *Datasheet SHT1x (SHT10, SHT11, SHT15) Humidity and Temperature Sensor*, July 2008. Retrieved from: https://www.sparkfun.com/datasheets/Sensors/SHT1x_datasheet.pdf.

[8] D. Vrijenhoek, *Improving the dependability of the temperature build-up sensor system in the city of Enschede*. utwente, July 2019. Retrieved from: https://essay.utwente.nl/78700/.

[9] M. Kusriyanto and A. A. Putra, "Weather station design using iot platform based on arduino mega," *2018 International Symposium on Electronics and Smart Devices (ISESD)*, 2018. DOI: 10.1109/isesd.2018.8605456.

[10] Senserion, *Datasheet SHT3x-Dis Humidity and Temperature Sensor*, February 2019. Retrieved from: https://www.mouser.com/datasheet/2/682/Sensirion_Humidity_Sensors_SHT3x_Datasheet_digital-971521.pdf.

[11] T. Brasser, I. Tesselaar, and D. Offerhaus, "Design of an autonomouswireless weather station: Ee3l11 - bachelor graduation thesis," in *Design of an AutonomousWireless Weather Station*, 2018. Retrieved from: https://www.semanticscholar.org/paper/Design-of-an-Autonomous-Wireless-Weather-Station-EE-Brasser-Offerhaus/8abf9597f218188f07e08bd36f5fcfcc1786fb8d.

[12] R. Firmansyah, A. Widodo, A. D. Romadhon, M. S. Hudha, P. P. S. Saputra, and N. A. Lestari, "The prototype of infant incubator monitoring system based on the internet of things using nodemcu esp8266," *Journal of*

*Physics: Conference Series*, vol. 1171, p. 012015, 2019. Retrieved from: https://iopscience.iop.org/article/10.1088/1742-6596/1171/1/012015.

[13] C. Sun and Y. Cao, "Design of mushroom humidity monitoring system based on nb-iot," *Advances in Intelligent Systems and Computing International Conference on Applications and Techniques in Cyber Intelligence ATCI 2019*, p. 281–289, 2019. DOI: 10.1007/978-3-030-25128-4_37.

[14] F. Akhter, S. Khadivizand, H. R. Siddiquei, M. E. E. Alahi, and S. Mukhopadhyay, "Iot enabled intelligent sensor node for smart city: Pedestrian counting and ambient monitoring," *Sensors*, vol. 19, p. 3374, Jan 2019. DOI: 10.3390/s19153374.

[15] "Vantage pro2." Available: https://www.davisinstruments.com/solution/vantage-pro2/.

[16] Bosch, *BME280 Combined humidity and pressure sensor, 0273141185 datasheet*, May 2015. Retrieved from: https://cdn-shop.adafruit.com/datasheets/BST-BME280_DS001-10.pdf.

[17] Texas Instruments, *HDC1080 Low Power, High Accuracy Digital Humidity Sensor with Temperature Sensor , SNAS672 datasheet*, November 2015. Retrieved from: http://www.ti.com/lit/ds/symlink/hdc1080.pdf.

[18] Microchip, *MCP9808 +- 0.5 C Maximum accuracy digital temperature sensor, DS25095A datasheet*, October 2011. Retrieved from: http://ww1.microchip.com/downloads/en/DeviceDoc/25095A.pdf.

[19] Davis Instruments, *Temperature Humidity Sensor, DS6834 datasheet*, March 2019. Retrieved from: https://www.davisinstruments.com/product_documents/weather/spec_sheets/6834

[20] P. Kyratsis and D. Tzetzis, "Investigation of the mechanical properties of acrylonitrile butadiene styrene (abs)-nanosilica reinforced nanocomposites for fused filament fabrication 3d printing," *IOP Conference Series: Materials Science and Engineering*, vol. 416, p. 012086, 2018. DOI: 10.1088/1757-899x/416/1/012086.

[21] T. Letcher and M. Waytashek, "Material property testing of 3d-printed specimen in pla on an entry-level 3d printer," *Volume 2A: Advanced Manufacturing*, 2014. DOI: 10.1115/imece2014-39379.

[22] J. Fernandes, A. M. Deus, L. Reis, M. Vaz, and M. Fátimaand Leite, "Study of the influence of 3d printing parameters on the mechanical properties of pla," *Proceedings of the 3rd International Conference on Progress in Additive Manufacturing*, pp. 547–552, 2018. DOI: 10.1115/IMECE2014-39379.

[23] W. Liu, J. Zhou, Y. Ma, J. Wang, and J. Xu, "Fabrication of pla filaments and its printable performance," *IOP Conference Series: Materials Science and Engineering*, vol. 275, p. 012033, 2017. DOI: 10.1088/1757-899x/275/1/012033.

[24] J. K. V. V. Shabana, R.V.Nikhil Santosh, "Evaluating the mechanical properties of commonly used 3d printed abs and pla polymers with multi layered polymers," *International Journal of Engineering and Advanced Technology Regular Issue*, vol. 8, no. 6, p. 2351–2356, 2019. DOI: 10.35940/ijeat.f8646.088619.

[25] Davis Instruments, *Anemometer, DS7911 datasheet*, February 2013. Retrieved from: https://www.davisinstruments.com/product_documents/weather/spec_sheets/7911_SS.pdf.

[26] Davis Instruments, *Solar Radiation Sensor, DS6450 datasheet*, July 2014. Retrieved from: https://www.davisinstruments.com/product_documents/weather/spec_sheets/6450_SS.pdf.

[27] Meter group, *ATMOS 22 | Sonic Anemometer | METER Environment*. Retrieved from: http://library.metergroup.com/Manuals/20419_ATMOS22_Manual_Web.pdf.

[28] Gill Instruments, *WindSonic Ultrasonic Wind Sensor | Gill Instruments*. Retrieved from: http://gillinstruments.com/products/anemometer/windsonic.htm.

[29] Davis Instruments, *Radiation shields passive and fan aspirated., DS7714 datasheet*, March 2019. Retrieved from: https://www.davisinstruments.com/product_documents/weather/spec_sheets/DS7714_6838_Rad

[30] R. Nakamura and L. Mahrt, "Air temperature measurement errors in naturally ventilated radiation shields," *Journal of Atmospheric and Oceanic Technology*, vol. 22, no. 7, p. 1046–1058, 2005. DOI: 10.1175/jtech1762.1.

[31] S. J. Richardson, F. V. Brock, S. R. Semmer, and C. Jirak, "Minimizing errors associated with multiplate radiation shields," *Journal of Atmospheric and Oceanic Technology*, vol. 16, no. 11, p. 1862–1872, 1999.

[32] "Radiation shields." Available: https://www.baranidesign.com/radiation-shields.

[33] A. Mader and W. Eggink, "A design process for creative technology," in *Proceedings of the 16th International conference on Engineering and Product Design, E&PDE 2014* (E. Bohemia, A. Eger, W. Eggink, A. Kovacevic, B. Parkinson, and W. Wits, eds.), pp. 568–573, The Design Society, 9 2014.

[34] "Moscow method."

[35] "Sp13 series connectors." Available: https://www.weipuconnector.com/Product_show_8.htm.

[36] Texas Instruments, *ADS111x Ultra-Small, Low-Power, I 2C-Compatible, 860-SPS, 16-Bit ADCs With Internal Reference, Oscillator, and Programmable Comparator*, January 2018. Retrieved from: http://www.ti.com/lit/ds/symlink/ads1115.pdf.

[37] RS PRO, *Datasheet black natural rubber sheet*. Retrieved from: https://docs.rs-online.com/fe96/0900766b81580c6a.pdf.

[38] espressif, *ESP32-WROOM-32*. Retrieved from: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf.

# A Code for ESP32

## A.1 Main arduino code mqtt

```
/*
  used pins esp32  = what                      = wire color
          pin 21 = SDA                        = orange or light pink
          pin 22 = SCL                        = Yellow (esp32 without screen not numbered pin
          pin 23 = Sensors on/off (ground)    = White
  I2C device  = I2C addres
    SHT31     = 0x44
   Analog read
    pin 34 = SolarRad = green
    pin 35 = WindDir = purple
    pin 36 = WindSpeed = blue
    pin 39 = BatVoltage = brown or grey
*/

//sets up all libraries
#include <Arduino.h>  // is for easy pin definitions
#include "ArduinoNvs.h" // allows for variables to be stored in flash memory
#include <SPI.h> // is for spi communication
#include <Wire.h> // is for i2c communication
#include <WiFi.h>  // this allows for usage of the WiFi module
#include <PubSubClient.h>  // library for mqtt communication

//sets up the login for mqtt and WiFi
const char* ssid = "*******";
const char* password =  "********";
const char* mqttServer = "*********";
const int mqttPort = ****;
const char* mqttUser = "******";
const char* mqttPassword = "*******";

// is for the setup of the WiFi for the ESP32
WiFiClient espClient;
PubSubClient client(espClient);

//sets up everything for the SHT31
#include "Adafruit_SHT31.h"
Adafruit_SHT31 sht31 = Adafruit_SHT31();


// definitions for the deepsleep and amount of cycles before data upload
#define uS_TO_S_FACTOR 1000000  // turns miliseconds in seconds
#define TIME_TO_SLEEP 56 // amount of seconds the controller goes in deep sleep
int Cycles = 5; // amount of cycles
```

```
// pin used for the different sensors
int PinWindSpeed = 39;
int PinSolarRad = 34;
int PinWindDir = 35;
int RelayPin = 23;

bool Save; // a boolean for the arduinoNVS library
// declaring the analog values for the sensors
float AnalogValueWindSpeed = 0; float AnalogValueWindDir = 0; float AnalogValueSolarRad = 0;

//declaring the values to calculate or store sensor values
float TempSHT, HumSHT, Time, OldTime, WindSpeedMPH, WindSpeedKPH, WindSpeedMS, Pulses, Solar

float MeasureTime = 2250; // ammount of milliseconds the windsensor will be counting pulses

// variables for the NVS Storages
float TempStorage, HumStorage, WindSpStorage, SolRadStor;


//booleans
bool WSMeasured = false; // boolean that checks of the sensor measerments have been done
bool SwitchPulse = true; // boolean that makes sure a pulse is counted once
bool TimeBool = false; // boolean so oldtime is set to time once

String mydata; //declared the string that is send to mqtt
char charMydata[50]; //char array for the data string

void setup() {
  Serial.begin(115200); //setup of serial communication
  Serial.println("========================");
  pinMode(RelayPin, OUTPUT); // initializes the relaypin

  digitalWrite(RelayPin, HIGH); // sets the relaypin to HIGH so the sensors have power
// starts the NVS library and includes the function NVSSTorage
  NVS.begin();
  NVSStorageSetup();


  SetupTAndHSHT(); // set up of temperature/humidity sensor

// initialized the deepsleep timer
  esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);

// if statement that only initialized WiFi when data has to be uploaded
  if (CounterStorage >= Cycles) {
    WiFi.begin(ssid, password);
```

```
    while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.println("Connecting to WiFi..");
    }
    Serial.println("Connected to the WiFi network");
    client.setServer(mqttServer, mqttPort);
    while (!client.connected()) {
      Serial.println("Connecting to MQTT...");
      if (client.connect("ESP32Client", mqttUser, mqttPassword )) {
        Serial.println("connected");
      } else {
        Serial.print("failed with state ");
        Serial.println(client.state());
        delay(2000);
      }
    }
  }
}

void loop() {

  if (!WSMeasured) { // if statement that makes sure measerments are done once per cycle
    windSpeed(); // call the function that measure windspeed and within all other sensors
  }

// prints all measered values in the serial monitor, stores all values in flash storage and
  if (WSMeasured && CounterStorage < Cycles) {
    NVSStorage();
    Serial.print("Humidity = ");
    Serial.println(HumSHT);

    Serial.print("Temperature = ");
    Serial.println(TempSHT);

    Serial.print("Solar Radiation = ");
    Serial.println(SolarRad);

    Serial.print("Wind Direction = ");
    Serial.println(WindDir);

    Serial.print("Wind Speed in km/h = ");
    Serial.println(WindSpeedKPH);

    Serial.print("Battery Voltage = ");
    Serial.println(BatVoltage);
```

```
    Serial.print("Counter = ");
    Serial.println(CounterStorage);
    Serial.println("========================");
    //delay(1000);
    Serial.flush();
    esp_deep_sleep_start();
  }

// if statement that is called when data has to be send
  if (WSMeasured && CounterStorage >= Cycles) {
    PacketString(); // calls function that makes the string that is send
    mydata.toCharArray(charMydata, 50); //turns the string into a char array
    client.publish("GP/Data", charMydata); // send the data to the mqtt broker
    NVSStorageReset(); // reset all stored values to zerp
    Serial.println("========================");
    Serial.flush();
    esp_deep_sleep_start(); // puts the controller in deepsleep
  }
}
```

## A.2 Deep sleep function

```
void print_wakeup_reason(){
  esp_sleep_wakeup_cause_t wakeup_reason;

  wakeup_reason = esp_sleep_get_wakeup_cause();

  switch(wakeup_reason)
  {
    case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("Wakeup caused by external signal using RTC_
    case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("Wakeup caused by external signal using RTC_
    case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused by timer"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Wakeup caused by touchpad"); break;
    case ESP_SLEEP_WAKEUP_ULP : Serial.println("Wakeup caused by ULP program"); break;
    default : Serial.printf("Wakeup was not caused by deep sleep: %d\n",wakeup_reason); brea
  }
}
```

## A.3 Functions for flash storage

```
// function that declared all flash storage variables
void NVSStorageSetup() {

  CounterStorage = NVS.getInt("CounterStorage");
  TempStorage = NVS.getFloat("TempStorage");
  HumStorage = NVS.getFloat("HumStorage");
  WindSpStorage = NVS.getFloat("WindSpStorage");
  SolRadStor = NVS.getFloat("SolRadStor");
 // Testing = NVS.getInt("Testing");
}

// function that adds measured values to the previous one and stores them back in flash stor
void NVSStorage() {

  ++CounterStorage;
  TempStorage = TempStorage +   TempSHT;
  HumStorage = HumStorage + HumSHT;
  WindSpStorage = WindSpStorage + WindSpeedKPH;
  SolRadStor =  SolRadStor + SolarRad;

  Save = NVS.setInt("CounterStorage", CounterStorage);
  Save = NVS.setFloat("TempStorage", TempStorage);
  Save = NVS.setFloat("HumStorage", HumStorage);
  Save = NVS.setFloat("WindSpStorage", WindSpStorage);
  Save = NVS.setFloat("SolRadStor", SolRadStor);

}

// function that sets all stored values to zero
void NVSStorageReset() {

  Save = NVS.setInt("CounterStorage", 0);
  Save = NVS.setFloat("TempStorage", 0);
  Save = NVS.setFloat("HumStorage", 0);
  Save = NVS.setFloat("WindSpStorage", 0);
  Save = NVS.setFloat("SolRadStor", 0);
}
```

## A.4 Solar radiation

```
void solarRad() {

  AnalogValueSolarRad = analogRead(PinSolarRad); //reading the values from the pyranometer
  float SolarRadV = map(AnalogValueSolarRad,0, 4095, 0, 3300); //Maping the analogue value t
  SolarRad = SolarRadV / 1.67; //converting mV w/M2
}
```

## A.5 Forming of the packet

```
void PacketString() {

  String initial = "T";  //Initializing of the string packet
  float TempAvg = TempStorage / Cycles;  //Calculating the average temperature from past cyc
  float HumAvg = HumStorage / Cycles;  //Calculating the average humidity from past cycles
  float SolarRadAvg = SolRadStor / Cycles;   //Calculating the average solar radiation from
  float WindSpeedKPHAvg = WindSpStorage / Cycles;   //Calculating the average wind speed fro

  digitalWrite(RelayPin, HIGH); // connecting the sensors to power to measure the wind direc
  windDir();   //calling the function that measure the wind direction
  BatteryVoltage(); // calling the function that measure the battery voltage (only in TTN ve
  digitalWrite(RelayPin, LOW);  // disconnecting the sensors from power

  //forming the string that is send to mqtt or TTN
  mydata = initial + TempAvg + "H" + HumAvg + "S" + SolarRadAvg + "V" + WindSpeedKPHAvg + "D

// printing the string that is send to mqtt or TTN to the serial monitor
  Serial.print("test mydata: ");
  Serial.println(mydata);
}
```

## A.6 Temperature and Humidity sensor

```
//function that sets up the SHT31
void SetupTAndHSHT() {

  //searches if the SHT31 is on the in i2c addres 0x44
  if (! sht31.begin(0x44)) {   // Set to 0x45 for alternate i2c addr
    Serial.println("Couldn't find SHT31");
    while (1) delay(1);
  }
}

// function that reads the SHT31 sensor
void TAndHSHT() {
//stores the sensor data in variables
  TempSHT = sht31.readTemperature();
  HumSHT = sht31.readHumidity();
}
```

## A.7 Temperature and Humidity sensor

```
//function that measures the wind direction in degrees
void windDir() {

  AnalogValueWindDir = analogRead(PinWindDir); //Reading out the wind direction part of the
  WindDir = (AnalogValueWindDir / 4095) * 360; //Mapping the analog values to degreees
}
```

## A.8 Code for measuring wind speed

```
void windSpeed() {

  Time = millis(); // sets the Time variable equal to runtime of controller

//sets the OldTime variable equal to intitial Time variable
  if (!TimeBool) {
    OldTime = Time;
    TimeBool = true;
  }

  TimeDif = Time - OldTime; // calculates how long the Wind speed is measured
  AnalogValueWindSpeed = analogRead(PinWindSpeed); //reads the analog data of the wind speed
  delay(1); // small delay, otherwise weird values are read from analog pin

//If statement that counts how often the anenometer is turned and adds them up
  if (AnalogValueWindSpeed < 1000 && TimeDif <= MeasureTime && SwitchPulse == true ) {
    Pulses = Pulses + 1;
    SwitchPulse = false; // boolean that makes sure the pulse has returned to normal value b
  }

// if statements the resets the boolean if analog value has returned to normal
  else if (AnalogValueWindSpeed > 500 && TimeDif <= MeasureTime && SwitchPulse == false) {
    SwitchPulse = true;
  }
 // if statements that stops the measering when measure time has past
  else if (TimeDif > MeasureTime) {
    OldTime = Time;
    WindSpeedMPH = Pulses * (2250 / MeasureTime); //calculated the wind speed in mph from th
    Serial.print("pulses: ");
    Serial.print(Pulses); //prints the amount of pulses to the serial monitor
    Pulses = 0; //resets the pulse variable
    WindSpeedKPH = WindSpeedMPH * 1.61; // converts wind speed from mph to kph
    WindSpeedMS = WindSpeedKPH / 3.6;  // converts wind speed from kph to ms
//prints all wind speed types to serial monitor
    Serial.print("  mph: ");
    Serial.print(WindSpeedMPH);
    Serial.print("  kph: ");
    Serial.print(WindSpeedKPH);
    Serial.print("  ms: ");
    Serial.println(WindSpeedMS);
    TAndHSHT(); // calls function to measure temperature and humidity
    solarRad(); // calls function that measure the solar radiation
    //BatteryVoltage(); //calls function that measure to battery voltage
    windDir(); // calls function that measure wind direction
```

```
 // if statement that ensures that the temperature and humdity is done correct
    if (TempSHT != -40) {
      WSMeasured = true;  // sets boolean to measerments done
      TimeBool = false;
      digitalWrite(RelayPin, LOW); //shuts off the power to the sensors
    }
  }
}
```

## A.9 Code for measuring the battery voltage

```
void BatteryVoltage() {
  AnalogValueBat = analogRead(PinBatVoltage); //reads analog value battery
  float AnalogVoltBat = AnalogValueBat * 0.0008058608; // converts analog value to mV
  BatVoltage = (AnalogVoltBat / 0.6993006993); // corrects offset made of voltage divider to
}
```

## A.10 Main arduino code TTN

```
/*
  used pins esp32  = what                      = wire color
          pin 21 = SDA                        = orange or light pink
          pin 22 = SCL                        = Yellow (esp32 without screen not numbered pin
          pin 23 = Sensors on/off (ground)    = White
  I2C device  = I2C addres
    SHT31     = 0x44
   Analog read
    pin 34 = SolarRad = green
    pin 35 = WindDir = purple
    pin 36 = WindSpeed = blue
    pin 39 = BatVoltage = brown or grey
*/

//sets up all libraries
#include <Arduino.h>  // is for easy pin definitions
#include "ArduinoNvs.h" // allows for variables to be stored in flash memory
#include <lmic.h> //library used for LoRaWAN module
#include <hal/hal.h> //library used for LoRaWAN module
#include <SPI.h> // is for spi communication
#include <Wire.h> // is for i2c communication

//sets up everything for the SHT31
#include "Adafruit_SHT31.h"
Adafruit_SHT31 sht31 = Adafruit_SHT31();

// definitions for the deepsleep and amount of cycles before data upload
#define uS_TO_S_FACTOR 1000000  // turns miliseconds in seconds
#define TIME_TO_SLEEP 56 // amount of seconds the controller goes in deep sleep
int Cycles = 5; // amount of cycles

// pin used for the different sensors
int PinWindSpeed = 13;
int PinSolarRad = 34;
int PinWindDir = 35;
int PinBatVoltage = 39;
int RelayPin = 23;

// Various constants to connects to the TTN network
static const u1_t PROGMEM APPEUI[8] = { ********** };
void os_getArtEui (u1_t* buf) {
  memcpy_P(buf, APPEUI, 8);
}
```

```
static const u1_t PROGMEM DEVEUI[8] = { ********** };
void os_getDevEui (u1_t* buf) {
  memcpy_P(buf, DEVEUI, 8);
}

static const u1_t PROGMEM APPKEY[16] = { ********** };
void os_getDevKey (u1_t* buf) {
  memcpy_P(buf, APPKEY, 16);
}

// setup code that confirms connection to TTN
static osjob_t sendjob;


// pins used to interface with the LoRaWAN module
const lmic_pinmap lmic_pins = {
  .nss = 18,
  .rxtx = LMIC_UNUSED_PIN,
  .rst = 14,
  .dio = {26, 33, 32},
};

const unsigned TX_INTERVAL = 60;

bool Save; // a boolean for the arduinoNVS library
// declaring the analog values for the sensors
float AnalogValueWindSpeed = 0; float AnalogValueWindDir = 0; float AnalogValueSolarRad = 0;
//declaring the values to calculate or store sensor values
float TempSHT, HumSHT, Time, OldTime, WindSpeedMPH, WindSpeedKPH, WindSpeedMS, Pulses, Solar



float MeasureTime = 2250; // ammount of milliseconds the windsensor will be counting pulses

// variables for the NVS Storages
float TempStorage, HumStorage, WindSpStorage, SolRadStor;

//booleans
bool Connected = false;  // boolean to make sure the node has made connection to TTN
bool Transmitted = false; // boolean to make sure data has been send to TTN
bool WSMeasured = false; // boolean to make sure all sensor measurements are done
bool Setup = false; // boolean that is executed ones but could not be in void setup
bool SwitchPulse = true; // boolean that ensure a single pulse is counted once

String mydata; //String that contains data that is going to be send
```

```
void setup() {

  Serial.begin(115200); //setup of serial communication
  Serial.println("========================");
  pinMode(RelayPin, OUTPUT); // initializes the relaypin

  digitalWrite(RelayPin, HIGH); // sets the relaypin to HIGH so the sensors have power
  // starts the NVS library and includes the function NVSSTorage
  NVS.begin();
  NVSStorageSetup();

  SetupTAndHSHT();  // set up of temperature/humidity sensor

  // initialized the deepsleep timer
  esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);

}

void loop() {

\\if statement so measerements are done once per cycle
  if (!WSMeasured) {
    windSpeed(); // call wind speed measering function
  }

// sets up the LoRaWAN data sending
  if (WSMeasured && !Setup && CounterStorage >= Cycles) {
    os_init();
    // Reset the MAC state. Session and pending data transfers will be discarded.
    LMIC_reset();

    // Start job (sending automatically starts OTAA too)
    do_send(&sendjob);

// sets boolean setup to has been done
    Setup = true;
  }
  // if statement that is called one no data needs to be send
  if (WSMeasured && CounterStorage < Cycles) {
    digitalWrite(RelayPin, LOW); // shuts off power to sensors
    NVSStorage(); // calls function that stores measured values in flash memory
    Serial.println("Going to sleep now");
    Serial.flush();
    esp_deep_sleep_start(); // puts controller in deepsleep
  }
```

```
// if statements that is called when data has to be send
  if (WSMeasured && CounterStorage >= Cycles && Setup) {
    os_runloop_once();
  }

// if statement that is true when connection to TTN has been made
  if (Connected) {
    NVSStorageReset(); // sets all variables stored in flash memory to zero
    Serial.println("Going to sleep now");
    delay(1000);
    Serial.flush();
    Connected = false;
    esp_deep_sleep_start(); // puts controller to deepsleep
  }
}
```

## A.11 Code for LoRaWAN ESP32

```
void onEvent (ev_t ev) {
    Serial.print(os_getTime());
    Serial.print(": ");
    switch(ev) {
        case EV_SCAN_TIMEOUT:
            Serial.println(F("EV_SCAN_TIMEOUT"));
            break;
        case EV_BEACON_FOUND:
            Serial.println(F("EV_BEACON_FOUND"));
            break;
        case EV_BEACON_MISSED:
            Serial.println(F("EV_BEACON_MISSED"));
            break;
        case EV_BEACON_TRACKED:
            Serial.println(F("EV_BEACON_TRACKED"));
            break;
        case EV_JOINING:
            Serial.println(F("EV_JOINING"));
            break;
        case EV_JOINED:
            Serial.println(F("EV_JOINED"));
            {
              Connected =true;
              u4_t netid = 0;
              devaddr_t devaddr = 0;
              u1_t nwkKey[16];
              u1_t artKey[16];
              LMIC_getSessionKeys(&netid, &devaddr, nwkKey, artKey);
              Serial.print("netid: ");
              Serial.println(netid, DEC);
              Serial.print("devaddr: ");
              Serial.println(devaddr, HEX);
              Serial.print("artKey: ");
              for (int i=0; i<sizeof(artKey); ++i) {
                Serial.print(artKey[i], HEX);
              }
              Serial.println("");
              Serial.print("nwkKey: ");
              for (int i=0; i<sizeof(nwkKey); ++i) {
                Serial.print(nwkKey[i], HEX);
              }
              Serial.println("");
            }
            // Disable link check validation (automatically enabled
```

```
            // during join, but because slow data rates change max TX
        // size, we don't use it in this example.
            LMIC_setLinkCheckMode(0);
            break;
        /*
        || This event is defined but not used in the code. No
        || point in wasting codespace on it.
        ||
        || case EV_RFU1:
        ||     Serial.println(F("EV_RFU1"));
        ||     break;
        */
        case EV_JOIN_FAILED:
            Serial.println(F("EV_JOIN_FAILED"));
            break;
        case EV_REJOIN_FAILED:
            Serial.println(F("EV_REJOIN_FAILED"));
            break;
        case EV_TXCOMPLETE:
            Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
            if (LMIC.txrxFlags & TXRX_ACK)
              Serial.println(F("Received ack"));
            if (LMIC.dataLen) {
              Serial.print(F("Received "));
              Serial.print(LMIC.dataLen);
              Serial.println(F(" bytes of payload"));
            }
            // Schedule next transmission
            os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_INTERVAL), do_send);
            break;
        case EV_LOST_TSYNC:
            Serial.println(F("EV_LOST_TSYNC"));
            break;
        case EV_RESET:
            Serial.println(F("EV_RESET"));
            break;
        case EV_RXCOMPLETE:
            // data received in ping slot
            Serial.println(F("EV_RXCOMPLETE"));
            break;
        case EV_LINK_DEAD:
            Serial.println(F("EV_LINK_DEAD"));
            break;
        case EV_LINK_ALIVE:
            Serial.println(F("EV_LINK_ALIVE"));
            break;
```

```
        /*
        || This event is defined but not used in the code. No
        || point in wasting codespace on it.
        ||
        || case EV_SCAN_FOUND:
        ||    Serial.println(F("EV_SCAN_FOUND"));
        ||    break;
        */
        case EV_TXSTART:
            Serial.println(F("EV_TXSTART"));
            break;
        default:
            Serial.print(F("Unknown event: "));
            Serial.println((unsigned) ev);
            break;
    }
}


void do_send(osjob_t* j) {
  // Check if there is not a current TX/RX job running
  if (LMIC.opmode & OP_TXRXPEND) {
    Serial.println(F("OP_TXRXPEND, not sending"));
  } else {
// calls packet forming function
  PacketString();

    LMIC_setTxData2(1,(uint8_t*)mydata.c_str(), mydata.length(), 0);
    Serial.println(F("Packet queued"));
  }
  // Next TX is scheduled after TX_COMPLETE event.
}
```

# B Python Code

## B.1 Code to get data from TTN and insert in mysql database

```
import time
import ttn
import mysql.connector

# Application ID and access_key to TTN network
app_id = "**************"
access_key = "**************"

#Variables that allow the connection to the mysql database
mydb = mysql.connector.connect(
        host = "*******", #ip of mysql database
        user = "*******",  #user of mysql database
        passwd = "******",  #password of mysql database
        port = "*******", #port of mysql database
    database = "*****" #database name
)

def uplink_callback(msg, client):

  TimeStamp = msg.metadata.time  # sets TimeStemp to current time
  Data = str(msg.payload_fields.Data)  #Sets the payload from TTN to string data
  Counter = int(msg.counter)  #Sets the counter from to TTN to integer counter
  PrototypeNumber = str(msg.dev_id) #Sets the board name to variable PrototypeNumber


  #parses the string with data so all paramaters are set to the right variable
  Split0  = Data.split("B")
  BatteryLevel = float(Split0[1])
  Split1  = Split0[0].split("D")
  WindDirection = float(Split1[1])
  Split2 = Split1[0].split("V")
  WindSpeed = float(Split2[1])
  Split3 = Split2[0].split("S")
  SolarRadiation = float(Split3[1])
  Split4 = Split3[0].split("H")
  Humidity = float(Split4[1])
  Temperature = float(Split4[0][1:])

#prints the parsed data
  print("Wind Direction = ", WindDirection, "Wind Speed = ", WindSpeed, "Solar Radiation = "
 # print(Split2)
#prints message received from TTN
  print("/////////////////////////////////////////////////////////////////")
  print(msg)
```

```
  print("////////////////////////////////////////////////////////////////////")
  print("prototype version = ", PrototypeNumber,"Payload = ", Data,"Counter = ", Counter,"Ti
  print("////////////////////////////////////////////////////////////////////")


# initialises connection with mySQL database
  mycursor = mydb.cursor()
# makes string with sql querry
  sql = "INSERT INTO Node1 (Device_ID, Counter, Temperature, Humidity, SolarRadiation, WindS
#makes string with all the sensor data
  val = (PrototypeNumber, Counter, Temperature, Humidity, SolarRadiation, WindSpeed, WindDir

#enters the data in the mySQL database
  mycursor.execute(sql,val)
  mydb.commit()


handler = ttn.HandlerClient(app_id, access_key)

# using mqtt client from TTN
mqtt_client = handler.data()
mqtt_client.set_uplink_callback(uplink_callback)
mqtt_client.connect()
#defines how long the is going to run
time.sleep(432000)
mqtt_client.close()

# using application manager client
app_client =  handler.application()
my_app = app_client.get()
#print(my_app)
my_devices = app_client.devices()
#print(my_devices)
```

## B.2 Code to get data from mqtt broker and insert in mysql database

```
import paho.mqtt.client as mqtt #import the client1
import time
import mysql.connector
from datetime import datetime

Counter = 1 # sets Counter to starts at 1

#Variables that allow the connection to the mysql database
mydb = mysql.connector.connect(
        host = "*******", #ip of mysql database
        user = "*******",  #user of mysql database
        passwd = "******",  #password of mysql database
        port = "*******", #port of mysql database
    database = "*****" #database name
)


############
def on_message(client, userdata, message):
#sets string Data to message recieved from mqtt broker
    Data = str(message.payload.decode("utf-8"))
    print("message received " ,Data)


#parses the string with data so all paramaters are set to the right variable
    Split0  = Data.split("B")
    BatteryLevel = float(Split0[1])
    Split1  = Split0[0].split("D")
    WindDirection = float(Split1[1])
    Split2 = Split1[0].split("V")
    WindSpeed = float(Split2[1])
    Split3 = Split2[0].split("S")
    SolarRadiation = float(Split3[1])
    Split4 = Split3[0].split("H")
    Humidity = float(Split4[1])
    Temperature = float(Split4[0][1:])
    PrototypeNumber = "WindStrobe22mqtt"
    global Counter



    print("inserting")
        # makes string with sql querry
    sql = "INSERT INTO GPData (Device_ID, Counter, Temperature, Humidity, SolarRadiation, Wi
```

```
#makes string with all the sensor data
   val = (PrototypeNumber, Counter, Temperature, Humidity, SolarRadiation, WindSpeed, WindDi

# initialises connection with mySQL database
    mycursor = mydb.cursor()
    mycursor.execute(sql,val)
    mydb.commit()
    print("inserted")
    print(Counter)
        #makes a string of the current time
    dateTimeObj = datetime.now()
    timeObj = dateTimeObj.time()
    print(timeObj.hour, ':', timeObj.minute, ':' , timeObj.second)
 # adds 1 to the counter when data is inserted in the database
    Counter = Counter + 1

########################################
broker_address="**********"
client = mqtt.Client("P1") #create new instance
client.username_pw_set(username="***********", password="*********")
client.on_message=on_message #attach function to callback

client.connect(broker_address) #connect to broker
client.loop_start() #start the loop
#subscribes to a certain topic in the mqtt broker
client.subscribe("GP/Data")


time.sleep(4) # wait
```

## B.3 Code to input data manually and insert in mysql database

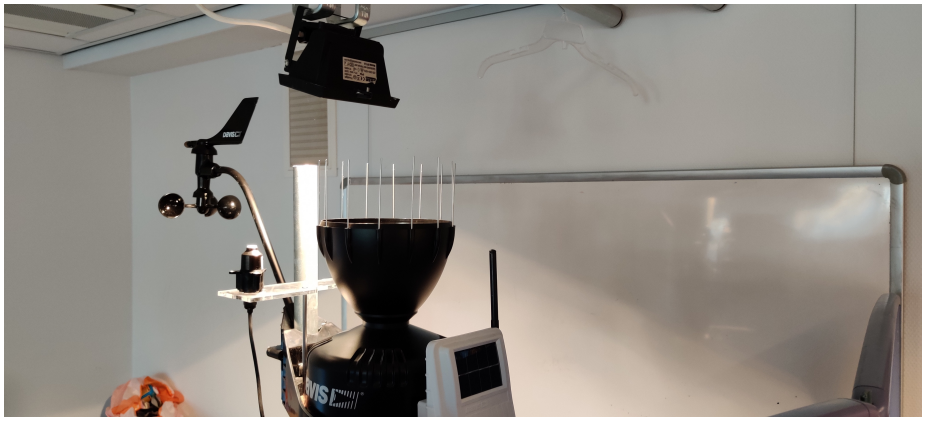
```
import time
import mysql.connector


#Variables that allow the connection to the mysql database
mydb = mysql.connector.connect(
        host = "*******", #ip of mysql database
        user = "*******",  #user of mysql database
        passwd = "******",  #password of mysql database
        port = "*******", #port of mysql database
    database = "*****" #database name
)

#code that makes the console ask all the values
#Values that stay constant are made constants

#BatteryLevel = float(input("What is the Battery level? "))
BatteryLevel = float(0)

#WindDirection = float(input("What is the Wind Direction? "))
WindDirection = float(0)

WindSpeed = float(input("What is the Wind Speed? "))
#WindSpeed = float(0)

#SolarRadiation = float(input("What is the Solar Radiation? "))
SolarRadiation = float(0)

Humidity = float(input("What is the Humidity? "))

Temperature = float(input("What is the Temperature? "))

Counter = int(input("What is the Counter? "))

Time = "2020-02-01 " +str(input("At what time? "))

PrototypeNumber = "WindStrobe22Davis"

#make the sql querry and inserts it in the database
print("inserting")
sql = "INSERT INTO GPData (Device_ID, Counter, Temperature, Humidity, SolarRadiation, WindSp
val = (PrototypeNumber, Counter, Temperature, Humidity, SolarRadiation, WindSpeed, WindDirec
```

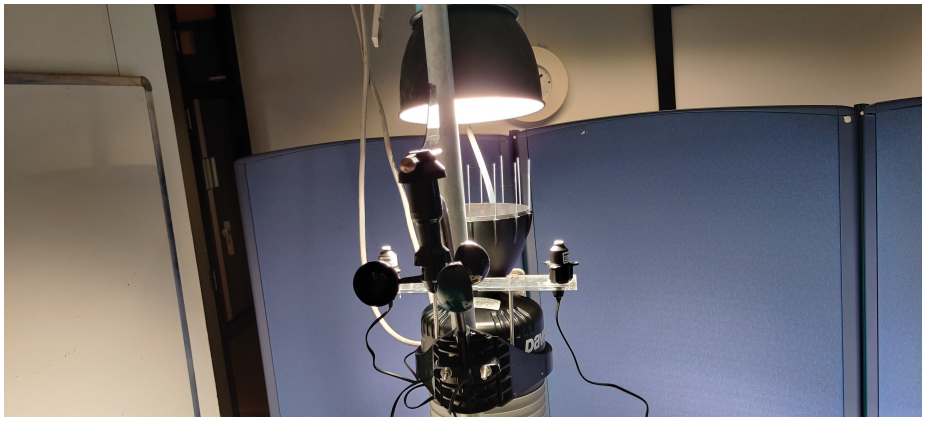
```
# initialises connection with mySQL database
mycursor = mydb.cursor()
mycursor.execute(sql,val)
mydb.commit()

time.sleep(4) # wait
```

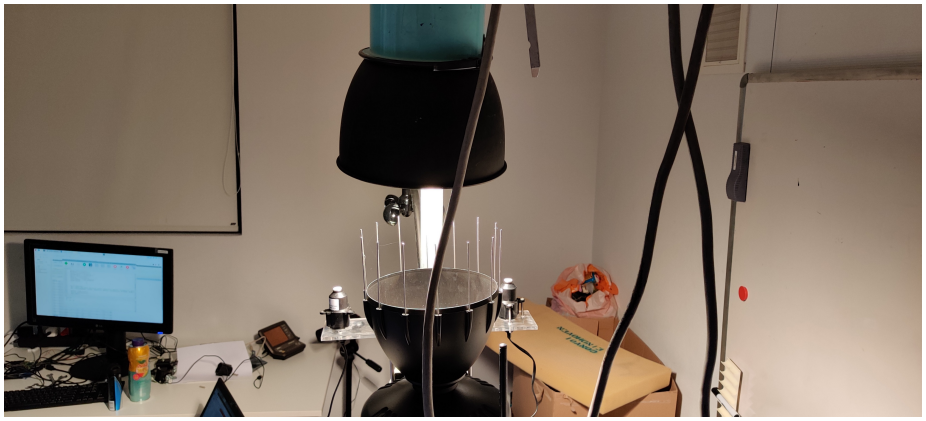
# C Pictures of test setup

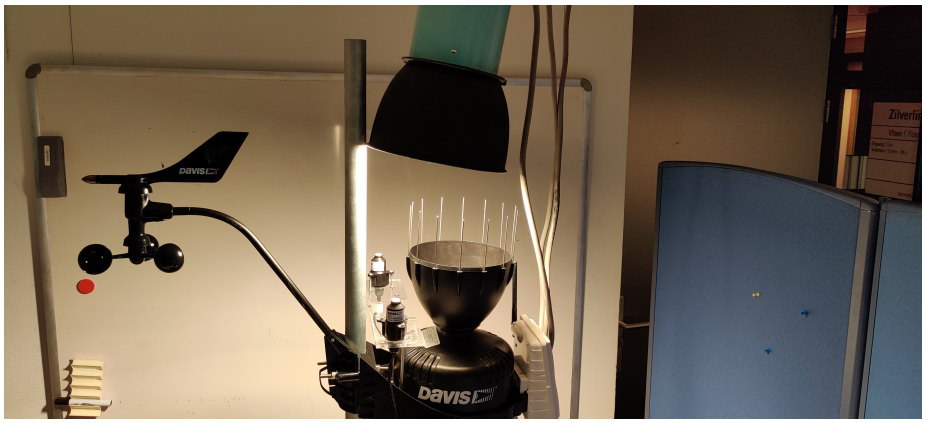
## C.1 Pictures of components test

## C.2 Pictures of light test








## C.3 Pictures of wind test setup