




# MONITORING IRRIGATION ISSUES IN SCOTTISH AGRICULTURE

FINAL REPORT | EPS

ANE GONZÁLEZ – CHRISTIAN KRULL – JACK MILLER – SIAM STREIBL

# Contents

# Abstract

The Scottish Environment Protection Agency frequently monitors the water level of various water bodies across Scotland and has noticed that a large number of those have a water level that is classified as “less than good”, posing environmental concerns. Having a reliable system to measure the water consumption in Scottish farms may avoid future environmental problems and establish a modern way of monitoring water consumption.

This report presents how an autonomous low-cost Long Range (LoRa) device can real-time monitor the water consumption and provide feedback to SEPA. Taking into consideration that the water flow of the pump is constant, a simple way to measure the water consumption is to detect the working time of the pump. This can be solved in two ways; either using sound and vibration sensors or the power supply of the engine’s alternator.

# Acknowledgements

Before we proceed to the report, we would like to say a couple words of thanks to the people that have given us this opportunity to work on our project.

All of this would not have been possible without our sponsor, the Scottish Environmental Protection Agency (SEPA). They have given the European Project Semester of Glasgow Caledonian University a project, in which we were chosen to fulfill and that we are thankful for.

A special thanks to our supervisor, Dr. Tuleen Boutaleb, for guiding us through the project by giving us important feedback as well reassuring the progression of our work.

We would also like to express our gratitude to our mentor, Prof. Andrew Quinn. Our time spent at GCU has been nothing but pleasant thanks to him. He has answered all of our questions that we have asked him, whether it was about administration or about our projects, and he has made sure that we were given accurate information as well as making sure that there was never a dull moment.

All of this wouldn't have been possible without their help.

# List of Figures

# List of Tables

# 1. Introduction

The European Project Semester is a program that involves students from different degrees, countries and universities from across Europe and beyond. This exact EPS module, run by the Glasgow Caledonian University is led by Prof. Andrew Quinn. The aim of this program is to provide the engineering students with the necessary skills to face the challenges of today's world economy with the project and also problem-based learning. Students are divided into interdisciplinary groups of 3 to 6 students.

The main problem is that Scotland lacks a reliable, real-time monitoring system that measures the water consumption of its farms and as a result numerous water bodies have a water level that is deemed as 'less than good'. This team has to develop a system that can measure the water usage of Scottish farms using a relatively new communications technology.

The Scottish Environment Protection Agency (SEPA) is the organisation that strives to protect and improve the Scottish environment. One of SEPA's main responsibilities is to monitor water consumption and gather all the information; it is their job to protect the environment by ensuring the correct consumption of water.

As mentioned before, there is no reliable system available to measure the water used for agricultural irrigation. Therefore, this project will focus on developing a way to real-time monitor the water consumption, following SEPA's requirements:

- to be a low-cost system (less than £200),
- to wirelessly communicate using LoRa, a technology that will be explained later on
- and to be low-maintenance.

This report will explain in detail all the process this group has followed, starting with the literature report about the three main areas (agriculture, communication/technology and hardware) up until the two possible solutions.

Summarizing, the main goal of this team is to prove that an autonomous low-cost Long Range (LoRa) device can be used to real-time monitor abstraction equipment. This will allow a more dynamic approach to managing the water resource across an agricultural catchment.

## 1.1. The Scottish Environment Protection Agency

The Scottish Environment Protection Agency (SEPA) is Scotland's principal environmental regulator, protecting and improving Scotland's environment.
It is a non-departmental public body (NDPB) of the Scottish Government with around 1300 people involved all over Scotland. Their role is to protect the environment and human health by monitoring the air, land and water. The results of the research done by SEPA are shared with the public and the industry to inform about the current situation and help those to understand and comply with their *environmental responsibilities and legislation*.

The organisation does several tasks as for example offering a flood warning system for Scotland and delivering the Scotland's Zero Waste Plan together with the Scottish Government.
Figure 1 shows the method of operating and research method of the SEPA. [1]

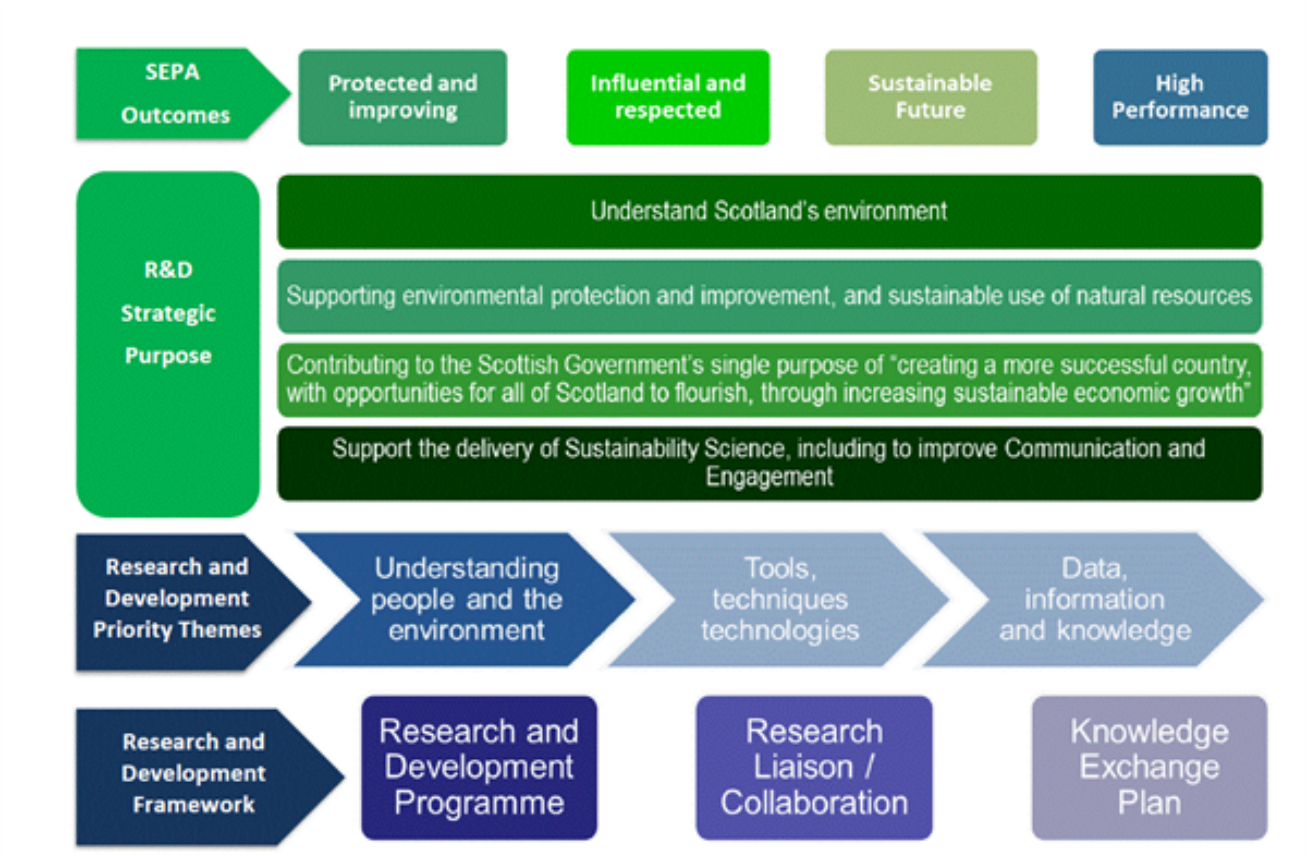



*Figure 1: Method of operation and research of SEPA*

## 1.2. Team members

There are four members taking part on this project: Ane González, Christian Krull, Jack Miller and Siam Streibl.

ANE GONZÁLEZ

- Spain
- Universidad Politécnica de Valencia
- Product Design Engineering (4th year)

- **Skills:**
  - 3D design & printing
  - Product development
  - Graphic design

CHRISTIAN KRULL

- Germany
- University of Applied Sciences Osnabrück
- Mechanical Engineering (4th year)

- **Skills:**
  - Statics
  - Dynamics
  - CAD Modelling
  - Mathematics

JACK MILLER

- Scotland
- Glasgow Caledonian University
- Electrical and Electronic Engineering (3rd Year)

- **Skills:**
  - Programming
  - Electrical design
  - Communications
  - Mathematics

SIAM STREIBL

- France
- École Nationale d'Ingénieurs de Tarbes
- Mechanical Engineering (3rd year)

- **Skills:**
  - CAD Modelling
  - Dynamics
  - Mathematics

## 1.3. Project Plan and Strategy

The project plan is the process of defining the scope, the objectives and the steps that need to be followed to obtain these objectives. The chart below shows the planned development of the project from the 20th of January up until our final presentation on the 11th of May (a total of 15 weeks).

As shown in Figure 2, the research of the project is divided into 3 main areas: Agriculture, Technology and Hardware. In the Agriculture part we will explain first the water measurement system in different countries around the world, and then in Scotland. Regarding the Technology part, we will present several communication technologies in order to explain the decision of choosing LoRa. In the Hardware part we will explain all the hardware that could be used in our project.

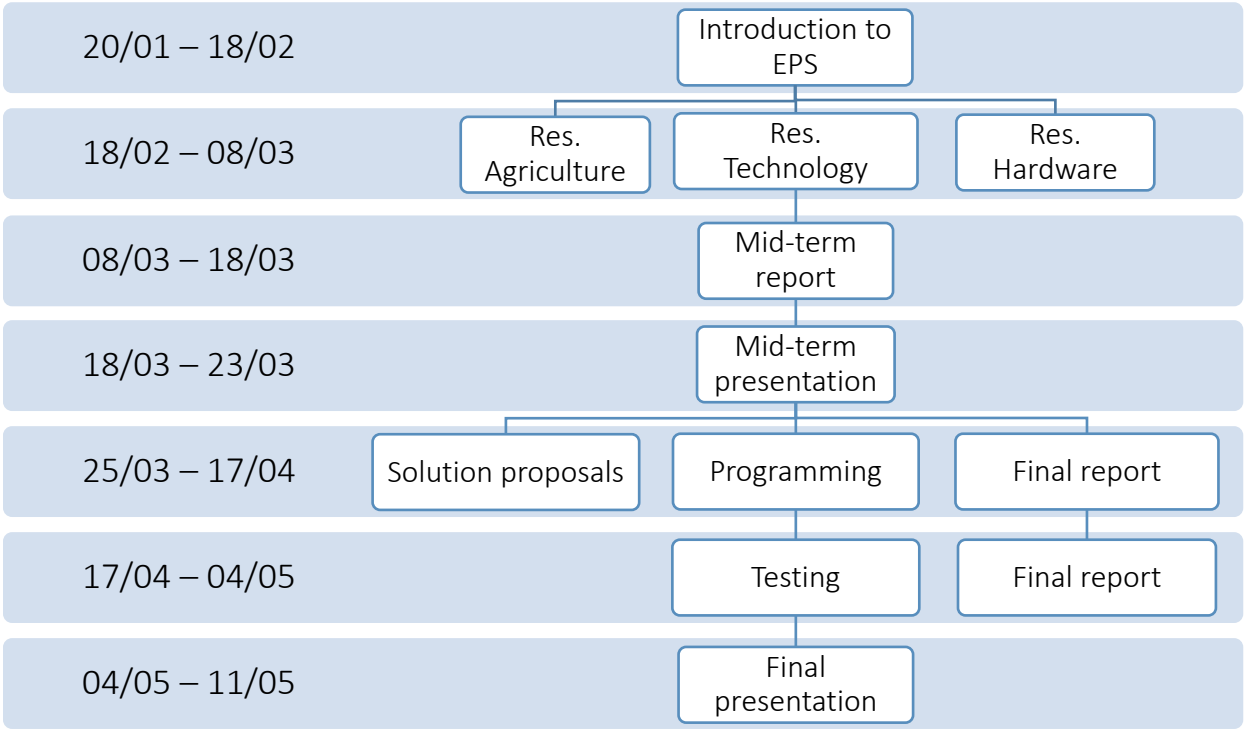



*Figure 2: EPS Project Plan*

Every project is unique, is temporary and accomplishes a goal, and in this case, the goal is to create a reliable, convenient and robust device that measures water consumption in Scottish farms.

Therefore, there are several steps and goals that have to be defined and organised so the project is developed in a correct and logical way.

Figure 3 shows the entire process that the group is following to constantly improve the project; through research, development, testing and refining.

Research is the first step of the strategy, so as to gather new information, once there are enough materials it is possible to start developing a solution, for example developing a prototype. Testing will provide all the feedback necessary to be able to refine the system and proceed in taking the required steps to achieve the main goal.

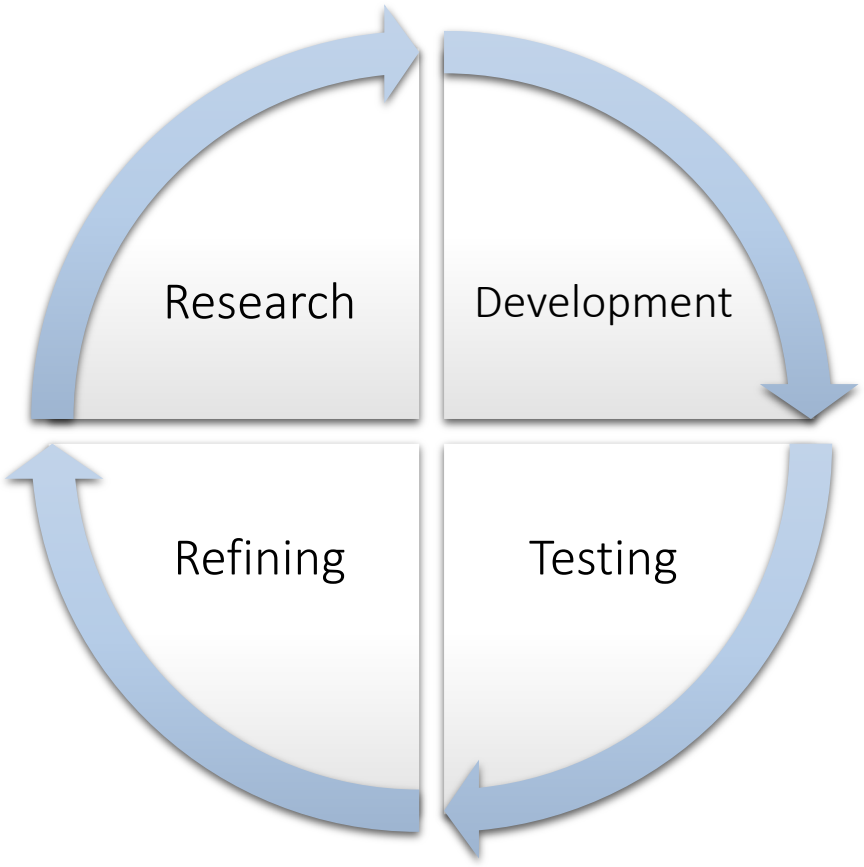



*Figure 3: Strategy of Project Refinement*

# 2. Literature Report

## 2.1. Agriculture in Scotland and water measurement methods

### 2.1.1. Water resources in Scotland

During our Agriculture research, we were searching for a map which shows the vulnerable water bodies in Scotland to define where we have to look at. We looked for areas of Scotland with problems in terms of the water level of lochs and with farms which have a huge water consumption. In general, Scotland is a country with large and sufficient water resources due to its mountainous regions, heavy rainfall and a relatively small population. “In practice, Scotland’s water resources are unevenly distributed and with the majority of the population resident in the central belt of the country, appropriate management of the water resources is essential.” – SEPA [2]

The main agricultural areas in Scotland are located down the east coast. There is the cultivation of crops, especially potatoes but also salad crops, grass and soft fruits. These plantations need a lot of water for the irrigation. The volume of water which is used for the irrigation of the Scottish crops depends on the weather conditions; if it is a dry summer the water consumption will be much higher. In Scotland this value is a bit higher because of the greater importance of the potato crop.

The areas West Peffer Burn in E Lothian and Elliot Water have a low effective runoff depth and a large crop area of potatoes. As a result, the volume of water that is available for the irrigation of the potatoes is very low (not shown in the map in Figure 4). [2]

The project focuses on 4 to 5 farms in these two areas highlighted, the finished solution will be implemented in these farms at first with the intention of having the system in a number of farms all over Scotland.

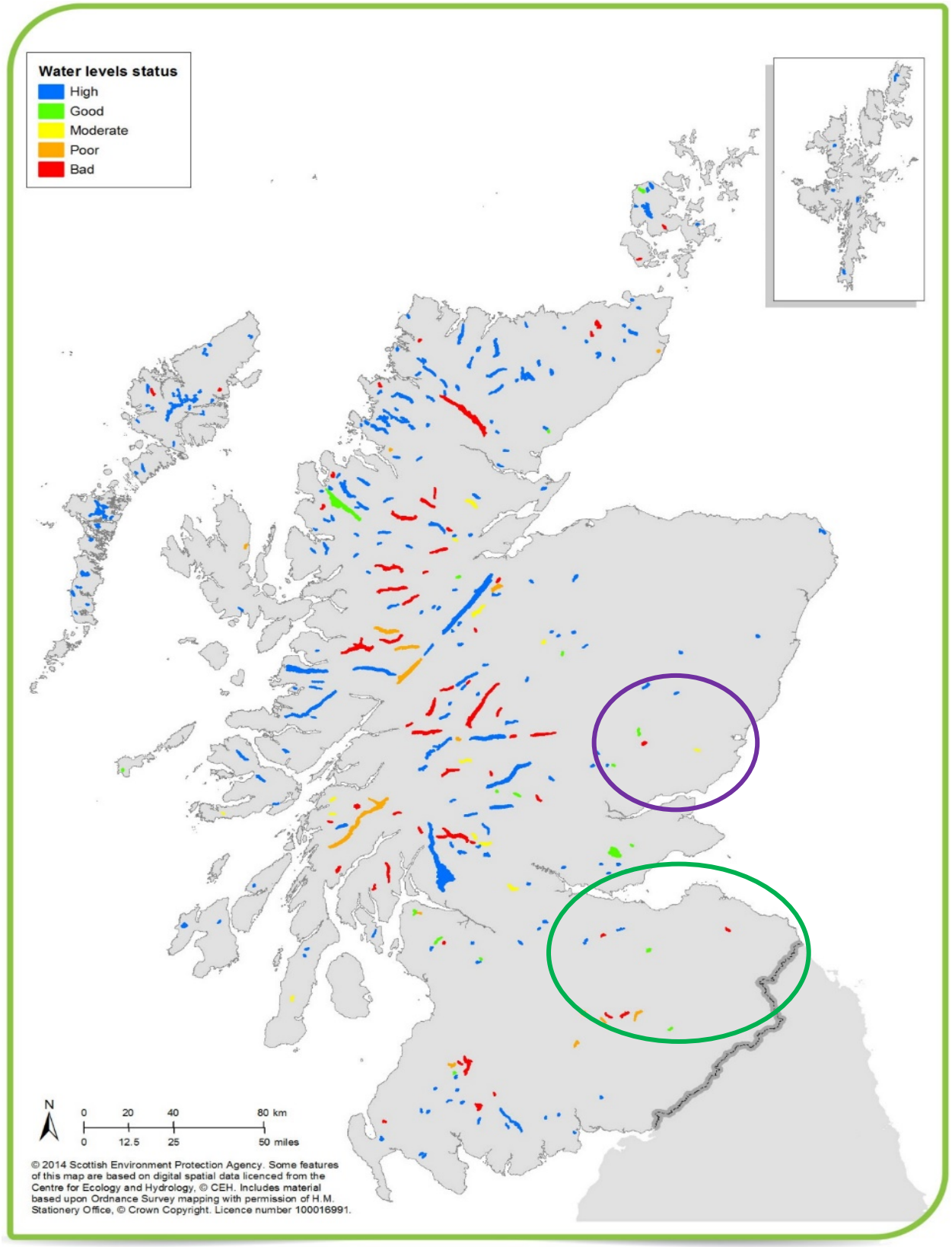



*Figure 4: Map of water levels in Scotland 2012 (source: SEPA)*

### 2.1.2. Water measurement methods

This chapter will go into depth concerning SEPA's main problem. Firstly, we are giving a small overview of different agricultural water monitoring methods in California, Spain, Germany and finally, Scotland.

#### California (US)

In California, 25% of its total land area is under cultivation and around 80% of its water is used by agriculture. [3] [4]

The farmers are using ground and surface water for the irrigation of their crops. A state law published in 2009 dictates that water suppliers who supply water to farms with a land area over 4047 hectares must annually report these farms water consumption. This information solely concerns the amount of water provided by canals. This information includes only the surface water.

California knows even less about the usage of groundwater than surface water. The state estimates the amount of used groundwater by a formula; the estimated water volume that the crops are requiring minus the amount of water which is delivered by the surface.

One existing solution to reduce the water consumption is created by one orange farm. They developed a system which checks the ground moisture of the orange trees and submits a warning signal if the ground is too dry. The moisture levels are then back to normal once the farmers have watered them.

Basically, California does not know enough about the water consumption of the farmers and this state is going to improve the measurement methods in the future. [5]

## Spain

According to the results of a survey conducted by the Ministry of Agriculture of Spain, 22% of the total crop fields of Spain (17 million hectares in total) are irrigated areas [6]. Approximately 51% of those irrigated fields is using a localised irrigation system [7].

This system consists on supplying water, so that it only wets the part of the crop in which the roots are developed. The principal part of this system is the irrigation head, where various mechanisms are situated in order to verify the operating efficiency.

These mechanisms are the following:

- Manometer: a device designed to measure the pressure of the system. It is situated at the entry and the exit of the pump and it measures the loss of power that is taking place.

- Water meter & Flow meter: mechanical or electromechanical devices that indicate the amount of water used in the irrigation, and the amount of water that crosses the head in a certain moment (instant flow).

- Programmer: electronic devices that are able to remember and execute in time the commands that control the operation of the parts of the system (valves, injectors…). [8]

## Germany

There were 48% of the total area of Germany (35 741 000 hectares) used by agriculture in 2015. 700 000 hectares of this area under cultivation (16 731 000 of hectare) were irrigated areas in 2015.

In Germany it is allowed to use ground water and water of surface water for the irrigation of the crops. The farmers are using mobile pumps (such as Scotland) and stationary pumps.

One of the federal states in Germany with the highest water consumption for irrigation due to the large area of potato cropping is Lower Saxony at the North. This federal state is subdivided into different irrigation organisations. The farmers in Lower Saxony are measuring their water consumptions themselves by using a water meter and transmitting the results to the irrigation organisations. These organisations have to transmit this information about the water consumption to the federal state authority, which checks the results. In Lower Saxony the quota of the water used by irrigation for the farmers is 80 l/m² per year and is defined for 7 years. The water

consumption is calculated by the area under cultivation of each irrigation organisation (e.g.: organisation 1 has 1000 hectares -> max. 800 million litre water per year).

Basically, the measurement method in Germany is similar to the method in Scotland. [Source: Family business experience]

## Scotland

73 % of Scotland's total land area are under agricultural usage (5.7 million hectares). [9] In 2003 the size of the total irrigated area was around 18 thousand hectares [10] which corresponds roughly to 0.31 percent of the total land area under agricultural usage.

The current problem of the agency is that there is no real-time monitoring of the water consumption of the farmers. The farmers merely have to measure their water consumption themselves and fill out a schedule, where they write down their yearly water consumption. After this, they need to send this schedule to SEPA. Their main problem is the questionability of those annual reports, as they are manually filled.

## 2.2. Communications system and Technology

### 2.2.1. Communications system

Our project requires us to send data from our devices to the internet. In order to accomplish this, a communications system will be needed.

A communications system, in general, is what makes it possible for our devices, or for us, to communicate. Commonly in the form of technology ranging from Bluetooth to Wi-Fi, other communication technologies exist. The next part will compare all commonly used communications system for the Internet of Things applications, and will be explaining which technology we are going to be using in our project and why.

#### IoT communications systems

This section will be discussing currently existing communications systems that are being used for the Internet of Things applications and go into detail concerning LPWAN technology. It is interesting to note that not one type of communications system fits all applications, as different forms of technology better respond to different requirements. The communications technologies listed below are compared to one another based on 3 different factors: bandwidth, power consumption, range and costs.

Bandwidth corresponds to the amount of data that can be sent. It is a measure of data/time and ranges from a couple of bytes every now and then (minutes) up to several megabytes per second.

Power consumption refers to the amount of power consumed when sending and receiving data as well as when the devices are on stand-by. It will be measured by each device's approximate power consumption (in mA) while using different communications technologies. It is also interesting to note that power consumption is a function of bandwidth, as LPWAN technologies such as LoRa or Sigfox only consume a fraction of power compared to cellular data or Wi-Fi. All data related to power consumption

Range is a measure of distance in which data can be sent. It is also correlated to bandwidth; as higher bandwidth technologies have a smaller range than lower bandwidth technologies. It is typically measured based on tests that present a line of site between the transmitter and the receiver and compared between urban and rural applications.

## Wi-Fi

Wi-Fi is a technology for wireless local area networking or WLAN for short. This form of wireless communications is ubiquitous in our modern world and in 2011, the United Nations said it believes that internet connection is a basic human right [11]. According to a survey conducted in 2017, 90% of households in Great Britain have internet access, an increase from 86% in 2015 and 55% in 2005. Wi-Fi access all around has been progressively on the rise since its introduction and is not expected to cease.

Key points on this technology are listed below:

- Bandwidth: up to 10 GB/s [12]
- Power consumption: 320 mA [13]
- Range [13]:
    - Urban: 50 metres on average
    - Rural: up to 250 metres

## Cellular Data

This form of communication is mostly used by mobile phones to connect to the internet. It isn't the most common communication technology for IoT applications as its cost (subscription costs to mobile service providers) and power consumption outweigh its benefits of being long range and can provide high data rates.

- Bandwidth: up to 20 MB/s for typical download speeds for 3G [14]
- Power consumption: 460 mA for 3G connections [12]
- Range [13] :
    - Rural: up to 8 kilometres
    - Urban: up to 70 kilometres

## LPWAN – LoRa vs Sigfox

Low Power Wide Area Network, or LPWAN for short, is a general term that encompasses many implementations and protocols, proprietary and open source that follows common characteristics as the name suggests:

- Low Power: devices send small amounts of data and can last years on a single battery

- Wide Area: data can be sent from several kilometres in urban areas and up to 70 kilometres in rural areas.

This technology unfortunately possesses a physical limitation to achieve these features: low data rate. Most LPWAN technologies can only send 1000 bytes of data per day with a data rate of 5000 bits per second on average [15].

## Link budget

Range depends on the Link Budget of the communications technology. The link budget is measured in decibels (dB) and dictates how far a signal can be sent. It is a description of all the gains and losses of a transmitter when it is active and is commonly represented with the following equation [16]:

$$\textit{Received Power (dBm)} = \textit{Transmitted Power (dBm)} + \textit{Gains (dB)} - \textit{Losses (dB)}$$

It is important to note that dBm is a measure of power, as the lowercase "m" refers to a milliwatt and is measured on a logarithmic scale. For example, 0 dBm would be equal to 1mW, 10 dBm to 10mW, 20 dBm to 100 mW and so on [17].

When a signal is transmitted, the receiver needs enough energy in order to detect it. When a signal is propagated from the transmitter to the receiver, it loses a certain amount of power as it is travels through obstacles and space. This is where receiver sensitivity comes into play.

LPWAN technologies operate with a link budget ranging from 140 dB to 160 dB. These values are achieved by high receiver sensitivities. Receiver sensitivities superior to -130 dBm are commonly used by LPWAN technologies, compared with the -90 to -110 dBm for other wireless technologies. Technologies with a receiver sensitivity of -130 dBm can detect transmitted signals 10,000 times weaker than those with -90 dBm, making LPWAN technologies unique [18].

## Radio Frequency Bands

LPWAN technologies commonly use unlicensed radio frequency (RF) bands reserved for industrial, scientific and medical purposes (ISM Bands) and each region operates on different RF bands. This makes it easier for any service provider or company to deploy and networks due to not having to

apply for a license or pay monthly fees to use them. This however limits the transmission power (ERP) to 25 mW for Europe and the U.S., for example.

The following part compares LoRaWAN technology to its competitor, Sigfox.

## Sigfox

SigFox is a French company that was founded in 2009 and can be described as the main player in the LPWAN industry due to its strong presence and successful marketing campaigns. It is also collaborating with big companies including Texas Instruments, Silicon Labs, and Axom.

SigFox is a proprietary technology using a slow modulation rate to achieve a longer range in which data can be sent. Due to this, SigFox is an excellent choice for applications where devices only need to send small, infrequent bursts of data.

Possible applications include parking sensors, water meters, or smart garbage cans. However, it also has some downsides. Sending data back to the sensors/devices (downlink capability) is severely limited and signal interference can become an issue.

## LoRa

LoRa is a unique modulation technique developed by Semtech and is mainly operated by the LoRa Alliance. The LoRa Alliance has about 400-member companies throughout the world, working to create a network based on the LoRa technology.

LoRaWAN is the open-standard networking layer governed by the LoRa Alliance. Basically, LoRa is the physical layer: the chip. LoRaWAN is the MAC layer: the software that's put on the chip to enable networking [19].

Its functionality is similar to SigFox in that it's mainly for uplink-only applications (sending data from sensors/devices to a gateway and not the other way around). Instead of using ultra-narrowband transmission like Sigfox (UNB), it spreads out information on different frequency channels and data rates using coded messages (Chirp Spread Spectrum Modulation). That being said, data sent is less likely to be subject to interference and gateways can accommodate a much larger number of connections.

Sigfox and LoRa characteristics [13] can be shown on Table 1:

*Table 1: Comparison between LoRaWAN and Sigfox*

| | | Sigfox (TD1207R) | LoRaWAN (RN2483) |
|---|---|---|---|
| Bandwidth | | 0.3 kb/s | 0.3 – 50 kb/s |
| Range | Urban | 3 – 10 km | 2 – 5 km |
| | Rural | 30 – 50 km | Up to 40 km [20] |
| Power consumption | | 32 – 51 mA | 40 mA |

The main difference separating these two is the fact that LoRa is more open than Sigfox. Sigfox owns all of its technology from the backend data and cloud server to the endpoints software, while LoRa encourages companies and individuals to contribute in developing the LoRaWAN network. Any hardware and or gateway manufacturer can create a module or gateway that conforms with the LoRa specifications. The catch for LoRa is that only one company called Semtech can develop LoRa radios, but announcements about giving its license to other silicon manufacturers in the future have been made.

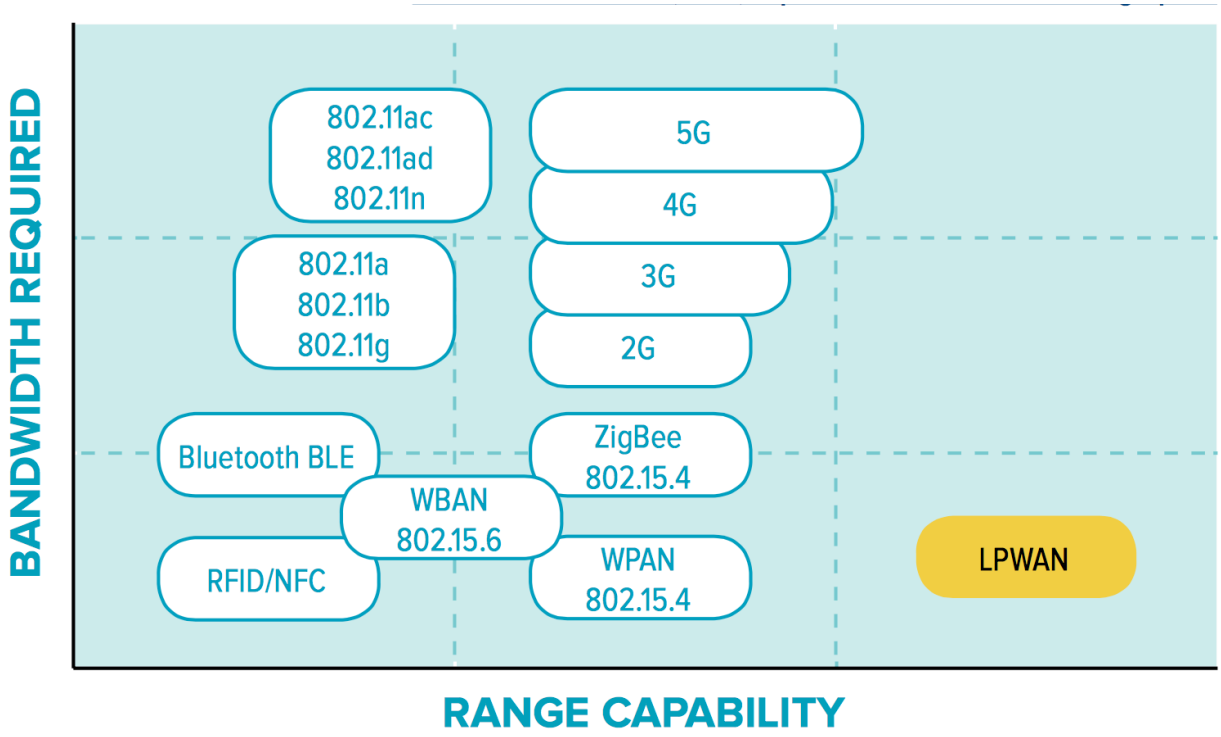



*Figure 5: LPWAN compared to other technologies (Source: Peter R. Egli, 2015)*

Figure 5 shows us where LPWAN technology stands compared to other communication technologies, where range and bandwidth are taken into consideration. Sigfox and LoRa are

considered the main players and rivals when it comes to LPWA networks, but other technologies are hopping on board the LPWAN train such as NB-IoT and LTE-M.

The project consists of a low-powered device hooked to a power source that will be sending small amounts of data received by sensors over long distances. It's clear that LPWAN is the technology for this, as it is cost efficient and low powered compared to cellular data and Wi-Fi (subscription fees and battery life) and provides a much wider range.

For the project, the team was specified to use LoRaWAN as a communications system and more details on this technology can be found in the following section.

### 2.2.2. LoRa

Originally developed by Cycleo (acquired in 2012 by Semtech for $5M), LoRa is a new technology that incorporates chirp spread-spectrum modulation technique, allowing this technology to transmit data through large distances while minimising interference and power consumption. This communications system is optimal for Internet of Things applications such as our project, as it excels in sending small amounts of data up to 40 kilometres away [20].

Basically, in our situation, we would have sensors connected to a LoPy micro-controller fitted with LoRa technology. These sensors would feed information to the micro-controller which would then, through LoRa, send this information to a gateway to afterwards have this information stored in a cloud that can be accessed by devices such as computers or smartphones.

Gateways function as a bridge between data sent with the LoRa technology and the internet. They are basically routers equipped with a LoRa concentrator, allowing it to receive LoRa packets or data. Devices use low power networks such as LoRaWAN to send information to a gateway which uses high bandwidth networks such as Wi-Fi, cellular data or Ethernet to upload the data onto the internet.

To set up a gateway, one would need 3 components which are: a LoRa concentrator board to receive LoRa packets (iMST iC880a board, for example [21]), an antenna to amplify the signal and a computer to access and interact with the data.

One gateway/concentrator has 8 RF channels, meaning it can support 8 IoT devices in parallel at a duty cycle of 100%, which is not a lot considering the millions of IoT devices. However, when duty cycles are reduced, the gateways can support many more devices. The law dictates that devices can have a maximum of 1% for its duty cycle, allowing a much larger number of devices to connect to the gateway.

For example:

- At 100% duty cycle, a gateway can support up to 8 devices
- At 50%, that number goes up to 16
- At 1%, 800 devices can be connected

Figure 6 depicts an example of a city utilizing the LoRa technology in order to receive information from various "things" around the city. These things range from smart street lights (intelLIGHT, the world's first street lighting control situation by Flashnet, a LoRa Alliance member), parking spots, traffic monitoring, etc. The idea of all these connected devices is to receive data such as brightness, temperature, street light status and many more in order to optimize energy use (smart street lamps), effectively calculating certain trajectories to get you from point A to point B the fastest (smart street lamps feeding information to our applications) and so on.

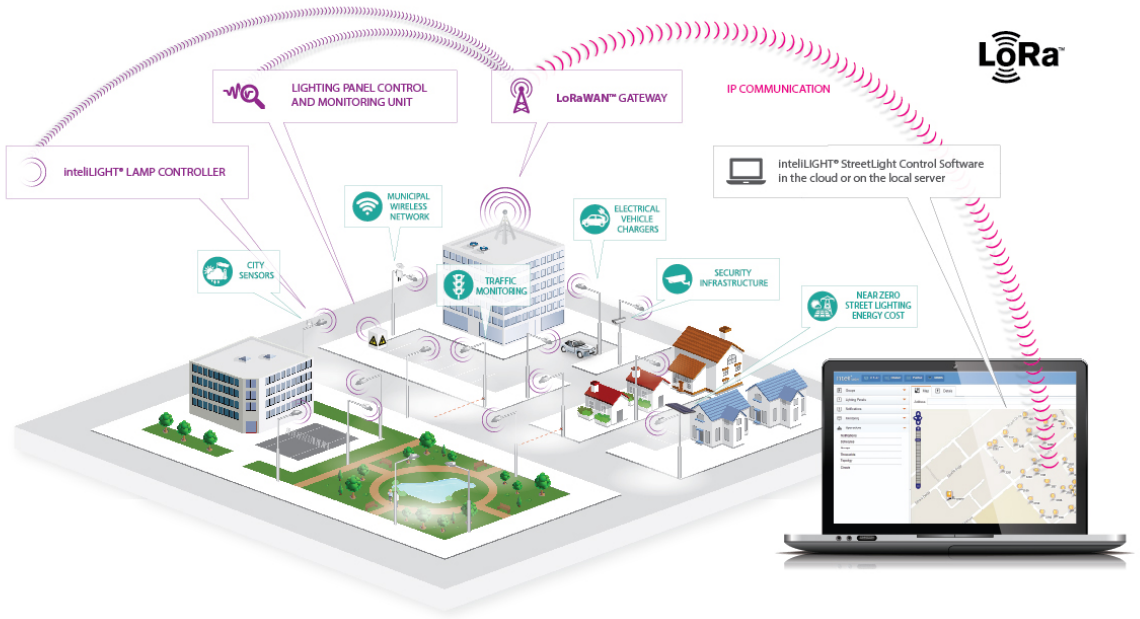



*Figure 6: LoRaWAN example*

Data is transmitted from devices to the internet with 4 main steps, as shown in Figure 7:

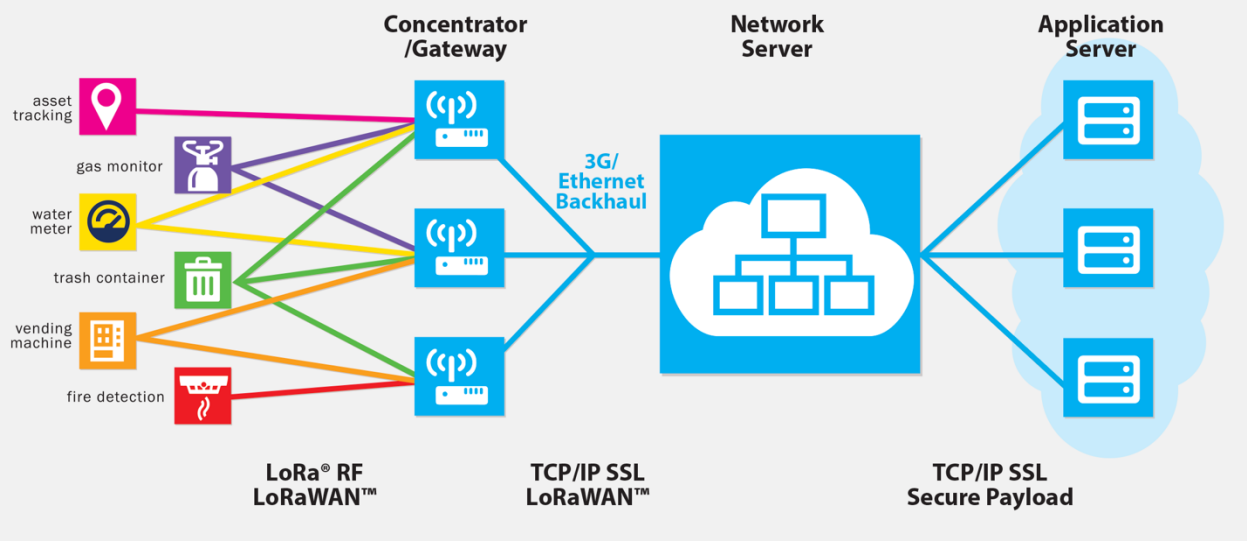



*Figure 7: LoRa data transmision*

The four main steps are:

1. Sensors gather information such as vibration frequency, temperature, acoustic levels, asset tracking, brightness and much more.
2. Information is sent to concentrators or gateways connected to the internet using LoRa technology.
3. The gateways then deliver data onto the cloud (the internet) with the means of Wi-Fi or cellular connectivity.
4. The data is then accessible from devices such as laptops, smartphones and tablets with access to the internet (browsers, programs, applications…)

Several advantages of this technology include:

- Long Range: allows us to monitor data from secluded farms spread over kilometres
- Low-power: minimises maintenance and optimises battery life
- Cost efficient: low-cost sensors and maintenance

## LoRa around the world

LPWA networks enable machine-to-machine (M2M), industrial IoT, community and official applications. LPWA networks have lower costs and energy requirements and longer range than mobile networks. Those advantages mean they can enable a much wider range of M2M and IoT applications, which have been limited by budgets and power issues. Businesses, nations and people are quickly adopting this technology around the world, taking us a step closer to a connected, smarter world.

Ever since the release of the LoRaWAN protocol in June 2015 by the LoRa Alliance, things have been developing rapidly concerning LoRa coverage around the world. The following part will explore its presence in our world with the help of the LoRa Alliance, and then present various projects and examples of regions and countries who have LoRaWAN coverage and promote LoRa technology.

### LoRa Alliance

The LoRa Alliance is an open, non-profit association which is dedicated to promoting the interoperability and standardization of low-power wide area network (LPWAN) technologies to further develop the Internet of Things applications around the world.

*"Our mission is to support and promote global adoption of the LoRaWAN standard by ensuring interoperability of all LoRaWAN products and technologies, to enable the IoT to deliver a sustainable future."* – LoRa Alliance [22]

The LoRa Alliance promotes its LoRaWAN protocol as an open global standard. According to the Alliance, entire countries or cities can be covered by the network with just a few base stations (gateways).

The Alliance has members throughout North America, Europe, Africa and Asia including big telecommunication companies, equipment and hardware manufacturers. Founding members of the LoRa Alliance include IBM, MicroChip, Cisco, Semtech, Bouygues Telecom, Singtel, KPN, Swisscom, Fastnet and Belgacom.

Telecommunications firms in different countries are the main players in promoting LoRa technology and offer LoRa as a cheaper alternative to cellular connectivity when it comes to IoT applications.

These telecommunications companies and projects in different countries are shown below [23]:

## a. Europe

- KPN (The Netherlands), which announced a nationwide LoRa network in The Netherlands at the end of June, 2016.
- Proximus (Belgium), the incumbent operator which was known as Belgacom promises nationwide LoRaWAN coverage in the country by the end of 2016.
- Orange (France), which started with 18 urban areas in the first quarter of 2016 and is expected to offer nationwide coverage soon as well.
- Bouygues Telecom (France), also promising nationwide LoRa coverage by end 2016 and offering IoT services with its daughter company: Objenious.
- Unidata, an Italian telecom company which announced the implementation of a LoRaWAN network on June 14$^{th}$. This will initially be done in Rome with a promise for fully covering the city by 2017.
- Netzikon, a LoRaWAN network which is present already in Stuttgart and will provide nationwide coverage for Germany by 2018 according to an October 4 press release by LPWAN provider Actility.

## b. North America (US)

Philadelphia and San Francisco have been selected for trials in IoT applications such as asset tracking, utility metering and environmental monitoring late 2016. If the trials are successful within the next 18-30 months another 28 'markets' should follow.

Senet offers the first public Low Power Wide Area Network (LPWAN) for Internet of Things applications, using LoRaWAN in North America with coverage in more than 100 US cities since June 2016 and planned to cover twice as many in 2017.

## c. Asia

With North America and Europe hopping on board with the technology, Asia has been a key focus point in the deployment of LPWA networks and improving LoRa coverage.

- Several initiatives are being deployed in Asia, with LoRa tech companies such as Semtech being regularly present at conferences and meetings in Asia.
- On September 14th 2016, Telecom Asia covered the plans of Japanese operator SoftBank to deploy a LPWAN network using the LoRaWAN protocol in its current fiscal year.
- Tata Communications already announced end 2015 that it would deploy a LoRaWAN network which had been tested in Mumbai, Delhi and Bangalore.
- On July 12th 2016, Semtech released a statement saying that the nationwide deployment of a LoRaWAN IoT network in South Korea by local telecom provider SK Telecom covered 99 percent of the population.

With LoRa technology, IoT applications have been met with new limits and is quickly gaining popularity with countries around the world. It offers new cost-saving solutions and ways of gathering data.

### 2.2.3. The Internet of Things

As shown in Figure 8, the Internet of Things refers to a network of physical devices that communicate and transmit data from objects all around us.

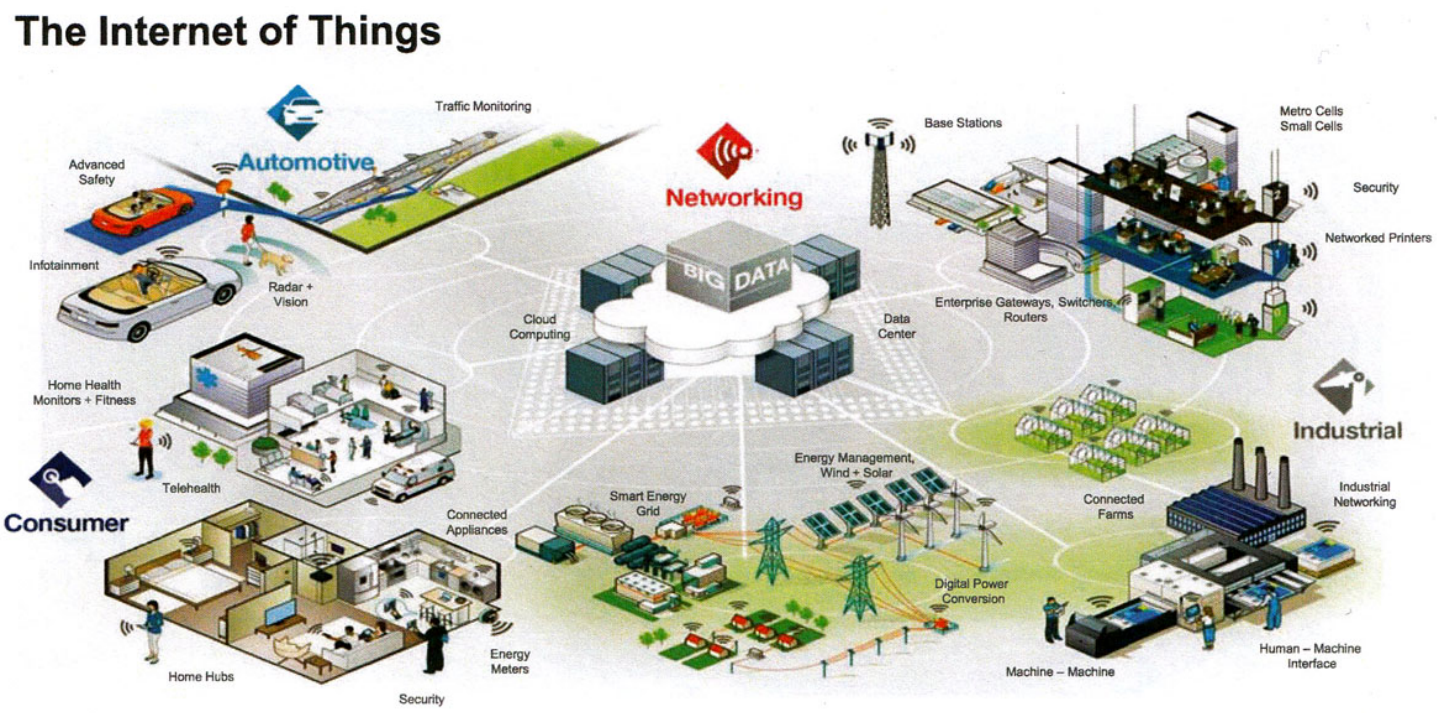



*Figure 8: Illustration of The Internet of Things*

One can already observe it being directly used, as it is used today to, for example, monitor the number of free parking spaces as well as where they are located. A good example of this tech being used would be the example of the Nest thermostat. Using a smartphone, one would be able to activate, control and then monitor the temperature in different rooms. After a certain time, your smartphone would be able to pick off your habits and automatically control your devices and allows you to manage your time in a more efficient way.

The applications of this technology are tremendous. We are entering a new era, where millions of devices are inter-connected. Smart cities, cities with everyday objects connected between each other through LoRa or different communications systems constitutes the Internet of Things.

In these cities, everything will be controlled and monitored by a multitude of sensors with endless applications. Water usage, air quality monitoring, fire detection, door locks, smart refrigerators, the list goes on. Everybody will be able to receive and evaluate real-time information that can be accessed through our smartphones.

## 2.3. Hardware & Software

### 2.3.1. Initial statement on Microcontrollers

Microcontrollers, although the name suggests different, they can be thought of as computers. A desktop PC may be good at doing everything and can be thought of as a 'general purpose computer'. Microcontrollers can be thought more of as 'special purpose computers', as they tend to do just one function very well. [24]

They are normally embedded inside another device so that they can control a specific feature or to perform certain actions of a product. Microcontrollers are dedicated to doing one task and run one specific program, the program will be stored in memory and in general, will not change. They are often low-power and rather inexpensive, it is because all these advantages and more, they are found in almost every electrical appliance or product.

### 2.3.2. Programming Language

For the programming of our project we used the programming language MicroPython. This open source Python programming language interpreter runs on small embedded development boards. The programming process is easy and simple because of this it is perfect to use for beginners in the world of programming.

MicroPython has many libraries with different examples of programming codes such as controlling LED strips or tiny OLED displays but these libraries are not as big as the libraries of the larger Python programming language. This programming language is able to control hardware and connect devices. The MicroPython is not as fast but it is useful for standard applications and the user have to know that this language is predominantly created for small boards with a smaller memory. [24]

# 3. The system and implementation

## 3.1. The Pump

The farms where we have to focus on are using the same type of mobile pumps. Therefore, it makes it easier for us to develop a system due to the similarity of the pumps. The pumps are made by Jones Engineering from Doncaster and using JCB engines and Caprari manufactured pumps. [25] The pump and the engine are encased by a large blue metal cover (Figure 9).

*Figure 9: Jones Engineering's Pump*

This locked cover protects the engine and the pump from environmental influences and vandalism. It easy to get access to the pump due to the lockable flaps on both sides of the machine as shown on Figure 10.
This facilitates the implementation of our system inside the blue cover.

*Figure 10: Jones Engineering's Pump (open)*

In our case it is also important that the machine is fitted with an electrical system which includes an alternator and a battery to supply our device with voltage. To use the pumps, the farmers are putting a tube or a pipe of the pump in a river or loch. At that point the position of the pump is near to the source of water. Then they have to connect the pump with the irrigation system. After that they need to start the engine and the pump works instantly because of the direct connection between pump and engine. The time till the machine pumps water depends mainly on the distance of the pump to the water body and the altitude difference between water source and pump.

In our case we do not know exactly the type of the used pump but to calculate the water consumption we are using the type *'PMXT HIGH PRESSURE MULTISTAGE HORIZONTAL PUMPS'* (Figure 11) with a capacity of 160 litres per second and 9600 litres per minute. [26] This information about the capacity of the pump is necessary to calculate the water flow of the pump using the measured working time.

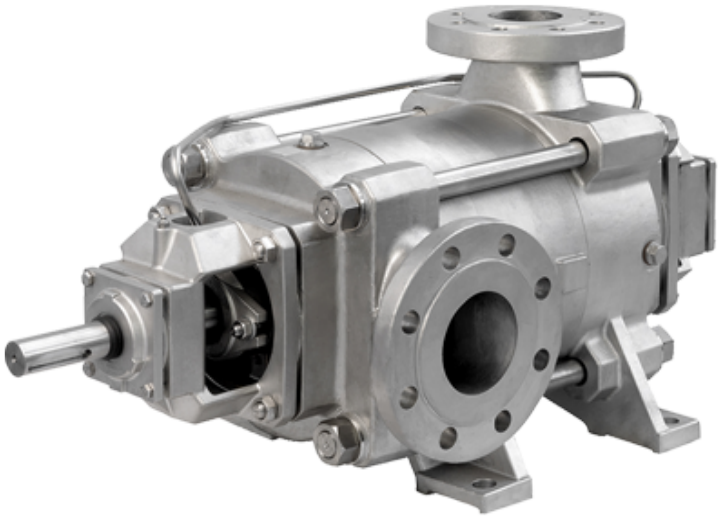


*Figure 11: Caprari pump PMXT*

## 3.2. The implementation: Pump Operation Detection Methods (Hardware Selection)

During our thinking process of how to detect the working time of the pump, we came up with four possible solutions. In this part of the report, we are going to explain each of our ideas with a short introduction and then we continue with the advantages and disadvantages of each idea.

### 3.2.1. Vibration

The first idea was to measure the vibration of the engine which drives the pump. To implement this idea, we have to install a vibration sensor on the engine housing or on other parts of the engine with a lot of vibrations. The vibrations of the engine depend on different factors:

- Engine speed
- Engine mount
- Number of cylinder of the engine

In our case we have an engine with four cylinders and the engine is fixed with four rubber buffers. During the working time, the engine works with around 1600 revolutions per minute. This is the optimal rotation speed of a diesel engine, meaning that the vibration intensity of the pump is at its minimum. [Source: family business experience].

We were looking for plenty of possible vibration sensors we can use. We mainly focused on their temperature resistance as a criterion due to the fact that we have to install them near the engine, where temperatures are relatively high. Furthermore, we also focused on the connectivity of the sensors with a Pycom microcontroller.

We came up with the following vibration sensors:

## a. Piezo Vibration Sensor - Large with Mass [27] (Figure 12)

- Dimensions: 15 x 30 mm
- Sensitivity increases mass to motion
- Wide dynamic range
- Operation temperature: 0°C to 85°C
- Applications: vibration sensing in washing machines, body movement ...
- Solder tab connection
- Price: £2.50

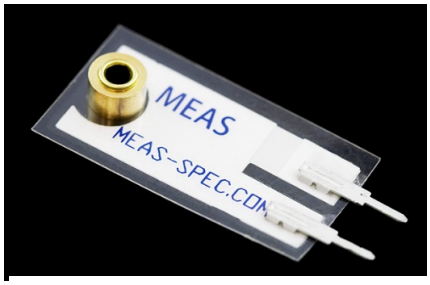



*Figure 12: Piezo Vibration Sensor*

## b. Assemtech Vibration Sensor 250 mA [28] (Figure 13)

- Dimensions: 18 x 4.6 mm
- Operating temperature: -20°C to 85°C
- Activation levels: 2G/ 5G/ 6G
- Resistant to several environmental conditions
- Price: £3

*Figure 13: Assemtech Vibration Sensor 250 mA*

## c. Adafruit ADXL345 - Triple-Axis Accelerometer [29] (Figure 14)

- Dimensions: (25x19x3.14) mm
- Operating with any 3V or 5V microcontroller (Pycom!)
- Sensitivity level of 2, 4, 8 or 16 G
- Three axes of measurements (X Y Z)
- Operation temperature: - 40°C to 85°C
- Price: around £12.50

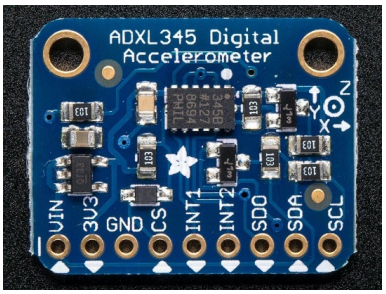



*Figure 14: Adafruit ADXL345 - Triple-Axis Accelerometer*

### 3.2.2. Microphone

The second idea we came up with was to detect the working time of the pump with a microphone. The engine makes a lot of sound during the working time and because of this it is easy to use a microphone as a detection method. Basically, the sound level depends on the engine speed and the area around the engine such as a large cover. Usually the engine works with 1600 rpm and the sound level is relatively high.

To implement this idea, it is necessary to install the microphone as near as possible to the engine to achieve the best measuring results.

We have chosen the following sensors to measure the sound of the engine:

a. SparkFun MEMS Microphone Breakout - INMP401 (ADMP401) [30] (Figure 15)

- Dimensions: (4.72 x 3.76 x 1.0) mm
- Applications: tablets, smartphones, …
- Max. sound level: 70dB
- Price: around £7

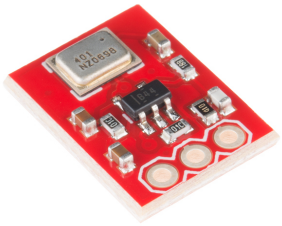


*Figure 15: SparkFun MEMS Microphone Breakout*

b. SparkFun Sound Detector [31] (Figure 16)

- Dimensions:
- Operating temperature: -40°C to +125°C
- Applications: Cellular phones, MP3 players, audio applications, …
- price: around £8

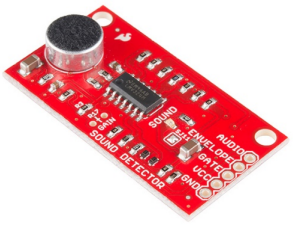


*Figure 16: SparkFun*

c. Adafruit Silicon MEMS Microphone Breakout - SPW2430 [32] (Figure 17)

- Dimensions: (15.8 x 14.1 x 2.9) mm
- Operating supply voltage: 3.3V to 5V
- price: around £5


*Figure 17: Adafruit Silicon MEMS Microphone Breakout*

### 3.2.3. Alternator (energy supply)

Our third idea we had was to detect the working time of the engine by using the voltage which is applied by the alternator of the engine. To implement this idea, we don't need any sensor to detect the working time of the engine and the pump.

The engine is connected with an alternator which is charging the battery of the system. The battery is mainly used to start the engine with the electrical starter motor.

To implement this idea, we are using just the voltage of the alternator which is applied only if the engine and the alternator are working. This implies that the Pycom board is switched on by the voltage of the alternator and sends data.

### 3.2.4. Vibration and Microphone

We came up with the idea to use a microphone and a vibration sensor together to remove some disadvantages of each system. For the implementation of this idea we just have to install a microphone and a vibration sensor as said below (first and second ideas).

The main reason why we were thinking about to connect two ideas was that we wanted to make the detection of the working time freer from errors such as the movement of the mobile pump during the transport on the street. In this case the vibration sensor is not able to differentiate between the working time of the engine and the movement on the street where the engine also vibrates.
For this idea both of the sensors have to measure sound or vibration to activate the data transmission of the Pycom microcontroller.

### 3.2.5 Advantages and disadvantages of the ideas

Table 2 shows the advantages and the disadvantages of our different ideas.

*Table 2: Advantages and disadvantages of the ideas*

| TYPE | ADVANTAGES (+) | DISADVANTAGES (-) |
|---|---|---|
| 1.Vibration | cheap sensor | cannot differentiate between movement on street and engine vibration due to the working time |
| | easy to install | extra case for sensor needed |
| | | needs voltage permanently (circuit breaker or switch to battery needed) and may require a battery |
| 2. Microphone | cheap sensor | needs voltage permanently (circuit breaker or switch to battery needed) |
| | easy to install | cannot differentiate between other surrounding sounds and working time |
| 3.Voltage (Energy supply) | no switch needed (energy supply) | perhaps unsteady voltage (12.5 – 14.5 V) |
| | safe detection of the engine working time | connection with the alternator can be difficult (depends on alternator) |
| | programming is easier | |
| | cheap (no sensor needed!) | |
| 4. Vibration and Microphone | safe detection of the engine working time | Difficult programming |
| | two sensors needed | difficult to install |
| | expensive | Battery and extra case for vibration sensor needed |

*Table 3: Key of rating for Table 3*

| Rating | Meaning |
|---|---|
| 1 | very bad |
| 2 | |
| 3 | |
| 4 | |
| 5 | perfect |

Based on Table 3, Table 4 shows the rating of the four ideas to detect the working time of the engine. We used points from 1 to 5 to rate each category. The most important categories for us were the measuring certainty with a weighting of 50% and the difficulty of the hardware installation with a weighting of 30%.

*Table 4: Rating of four ideas*

| Types | Hardware installation (30%) | Programming (10%) | Costs (10%) | Measurement certainty/ reliability (50%) | Total |
|---|---|---|---|---|---|
| 1. Vibration | 3 | 3 | 4 | 3 | 3.1 |
| 2. Microphone | 4 | 2 | 4 | 2 | 2.8 |
| 3. Alternator (Energy Supply) | 3 | 5 | 5 | 5 | **4.4** |
| 4. Vibration and Microphone | 3 | 2 | 3 | 4 | **3.4** |

After this rating we made our decision to use the alternator and the combination of the vibration sensor and microphone as the final solutions. It was not possible to test the final solution on a real pump due to the fact that the farmers only use their pumps during the summer season. That is why we decided to use the vibration and microphone idea as a second final solution.

Main reasons for the choice of the alternator idea as our primary solution were its accuracy to detect when the pump is being used and the fact that we do not need any sensor, thus cutting costs. Furthermore, one important point was the to use the alternator as a power supply, eliminating the need for a battery. The technical details about our final ideas will be explained in the following part of this report.

# 4. Solution/Methodology

## 4.1. Implementation of our solutions

In the following part the two final solutions will be explained in detail: the usage of the Alternator (power supply) and the combination of vibration sensor and microphone. The reason behind the selection of two ideas as final solutions is that we had no chance to test our device in reality. Originally, we wanted to test our ideas with a real pump to decide which idea will be the best solution. We are providing two different options to make sure that one of them will work under real conditions.

### 4.1.1. Alternator (power supply)

#### Description

As above-mentioned, our primary solution is to use the alternator as the pump operation detection method. In this case we are using the applied voltage of the alternator to detect if the engine and the pump work. The microprocessor is hooked onto the pump's alternator as an energy source. Simply put, when the pump is turned on, the Pycom microprocessor boots up and starts sending data until the pump is turned off.

Basically, the pump's alternator works the same way as a vehicle's alternator. When the engine starts, the alternator applies a voltage providing the system with an electrical current. In our pump the electrical system is primarily used to start the engine by the electrical starter motor.

To generate the power for our microcontroller we are using the connectors D+ or WL (Warning Lamp shown in Figure 18) of the alternator. There we have a voltage only if the engine works, whereas other power ports on the pump provide power constantly. Usually, the main task of this connector is to control the charging and working process of the alternator. If the alternator does not work correctly, the plus-potential changes to a ground potential. This then turns on the lamp because it gets a plus-potential from the other side; the battery. If the alternator works correctly the control lamp gets two plus- potential from both sides and is not able to shine. [33]

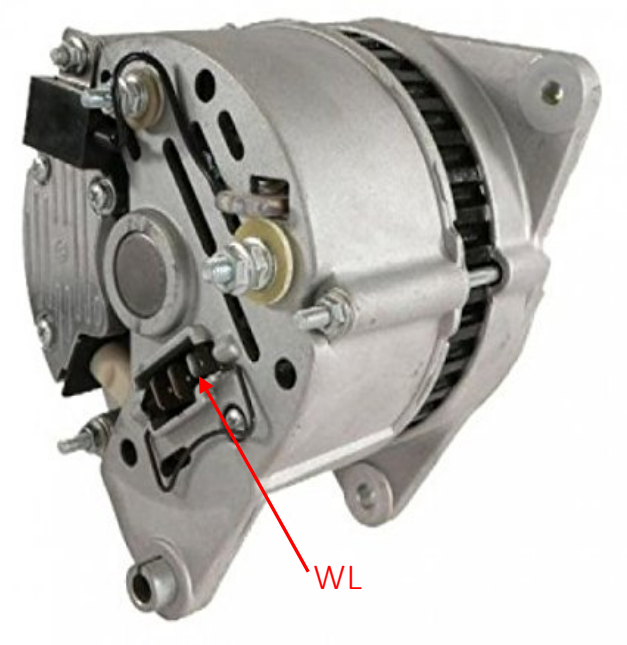



*Figure 18: Alternator of the engine*

## Hardware installation in the pump

For the installation of our idea we were thinking about the place where we can install our device. We thought about different requirements for the implementation:

- Theft protection of the device
- Protection against environmental influences
- To be far away to the engine (due to the engine heat)
- Easily removable
- Antenna outside of the large cover around the pump and engine (due to the connection quality)

There is a grid on the top of the large blue cover (Figure 19). Due to these requirements above, we decided to place our device under pump's top cover with a magnet, where there is enough space and can have the antenna pop out through the grid.

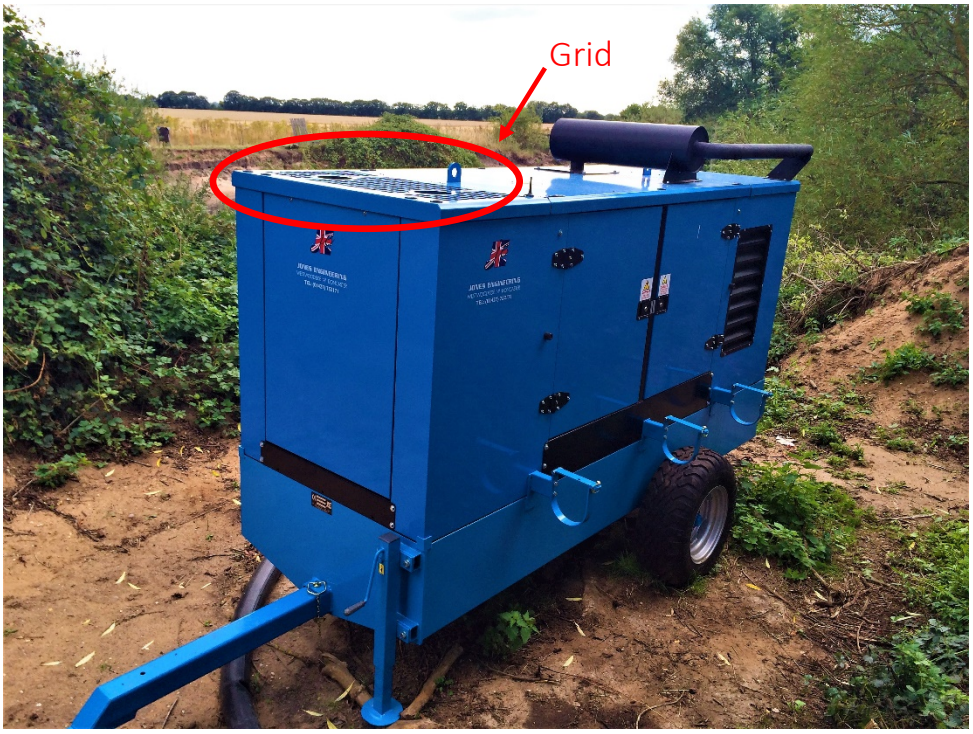



*Figure 19: Grid on the top of the Pump*

Figure 20 shows the possible implementation of our device in the mobile pump of the farmers, with different components coloured according to Table 5. Due to not being able to carry out a site visit, we will be assuming that this is how the pump's components are placed (engine placed at the rear due to the presence of the ventilation grid as shown in Figure 19.)

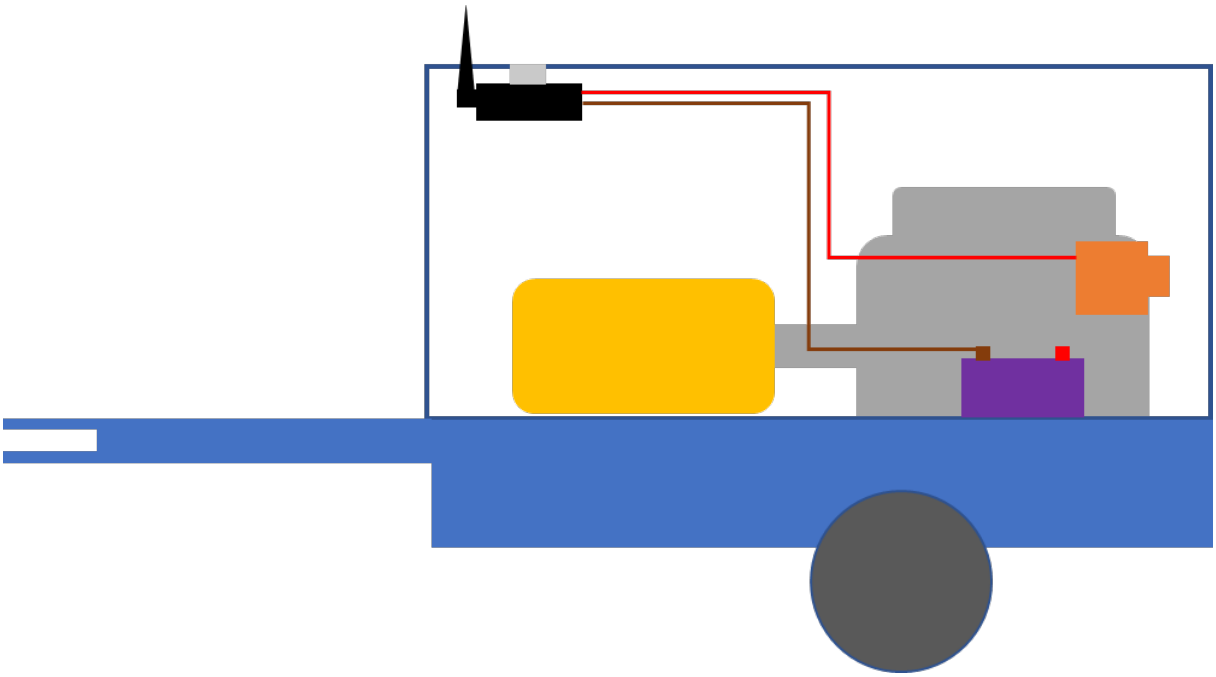


*Figure 20: Possible implementation of the device in the Pump*

*Table 5: Key of colours of Figure 14*

| **KEY – Alternator (power supply)** | |
|---|---|
| **Black** = Pycom microcontroller with case | **Orange** = alternator |
| **Light grey** = magnet | **Yellow** = pump |
| **Dark grey** = engine | **Brown cable** = ground (-) |
| **Purple** = battery | **Red cable** = plus-potential (+) |

## Additional hardware (power supply) [34]

The applied voltage by the alternator (when the pump is on) is around 13 V to 14.5 V, but our board can only support from 3.5 V to 5 V. That being said, we have to install a transformer in our system to reduce the voltage for our Pycom board.

We are suggesting the usage of the DC buck module sowed in Figure 21 (model: 2203ADJ) for the following reasons:

- Small dimensions: (34 x 25 x 20) mm
- Input voltage: DC 8 – 22V (12 – 14.5 V needed)
- Output voltage: 1 – 15V (3.5 – 5 V needed)
- Operating temperature: -20 to +60
- Short circuit protection
- Over-current protection
- Over-temperature protection
- Shockproof, moisture-proof, dust-proof
- Price: 7.80£ (amazon UK)
- Voltage is adjustable
- Aluminium case


*Figure 21: DC buck module*

## Wiring diagram – Alternator (power supply)

Figure 22 shows the wiring of our first solution with the alternator. This diagram does not show the wiring of other components like the charging lamp, which is also wired to the connector WL of the alternator. The brown ground wire can also be connected to another place instead of the battery, such as the engine housing or a part of the alternator.

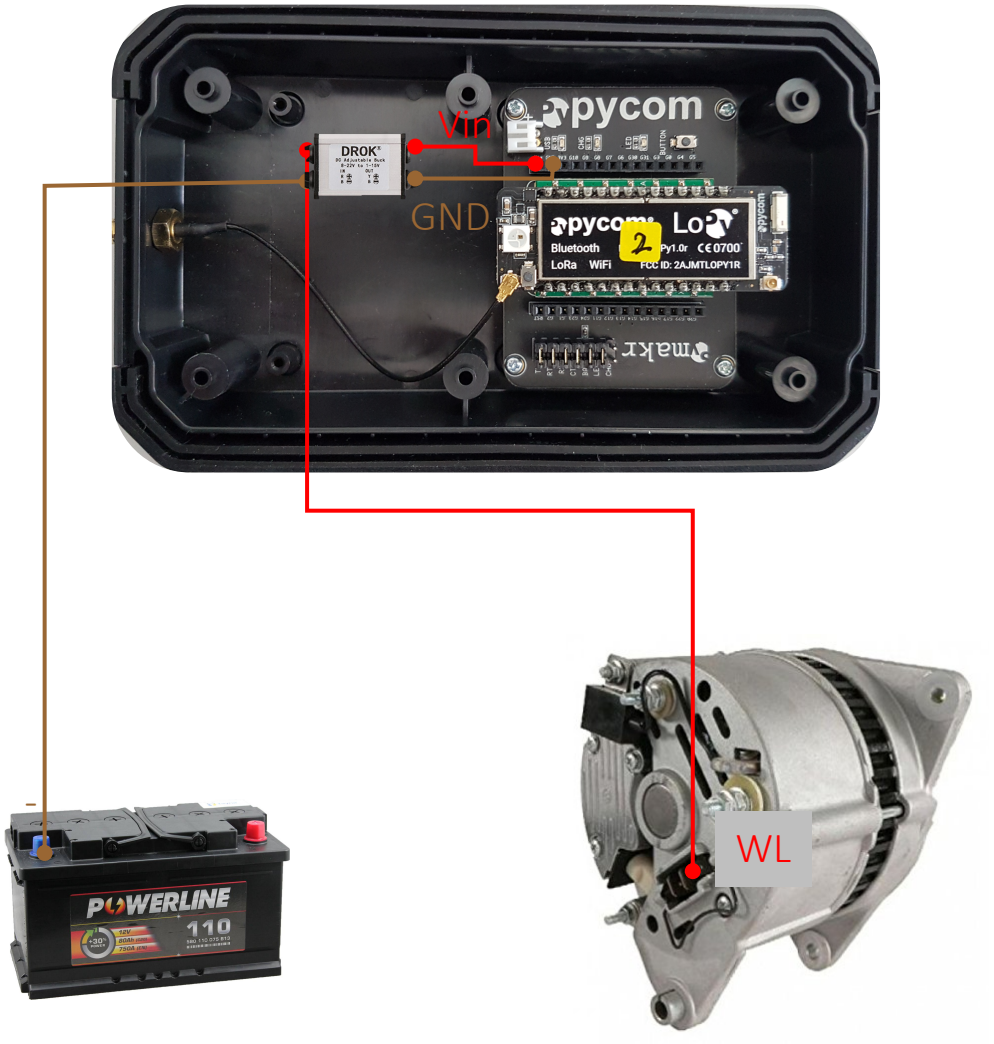



*Figure 22: Wiring diagram*

### 4.1.2. Microphone and Vibration

#### Description

Our second solution is to combine the vibration sensor with the microphone. To detect the working time of the pump both sensors have to measure movement or sounds. Basically it means that the Pycom microcontroller sends data only if both sensors are sending signals to the microcontroller. This way, we make sure that the detection of the pump's working time will not be influenced by other sounds or movements of the engine, such as the sound of another vehicle.

For the implementation of this idea we have to use a vibration sensor (accelerometer) and a microphone. The accelerometer measures the vibration of the engine and the microphone measures the sound of the engine.

Both sensors (accelerometer and microphone) are connected and soldered to the Pycom microcontroller via cables.

#### Hardware installation in the pump

These are the criterions that has been taken into consideration when installing the device:

- Theft protection
- Protection against environmental influences
- To be far away to the engine (due to the engine's heat)
- Easily removable
- Antenna outside of the large cover around the pump and engine (due to the connection quality)

For the two sensors, these are the criterions taken into consideration:

- Has to be installed near the engine
- Needs extra case (should be splash waterproof)
- To consider the temperature resistance (max. 85 °C!)
- Easy removable

The device is placed at the same position as the first solution, whereas this time two sensors are involved. In order to keep the sensors safe, two cases have been designed. There are two ways of installing the cases:

- To use the small magnet placed on the case and fix it to the engine
- To screw the cases to the Pycom case.

Both cases will be waterproof, easily removable and temperature resistant. The case for the vibration sensor can be placed on a vibrating part of the engine where the temperature is not to high; for example, on the valve cover.

Figures 23 and 24 show the 3D-printable cases. The blue one is for the accelerometer and the green one for the microphone.

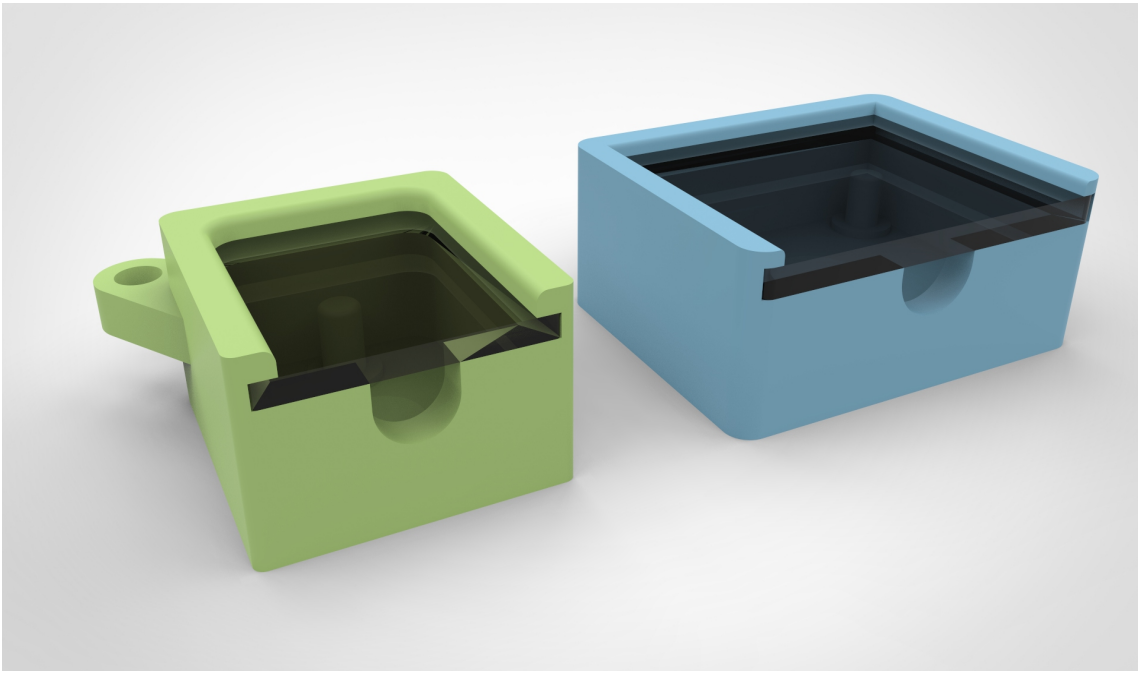


*Figure 23: Cases for microphone and accelerometer (closed)*

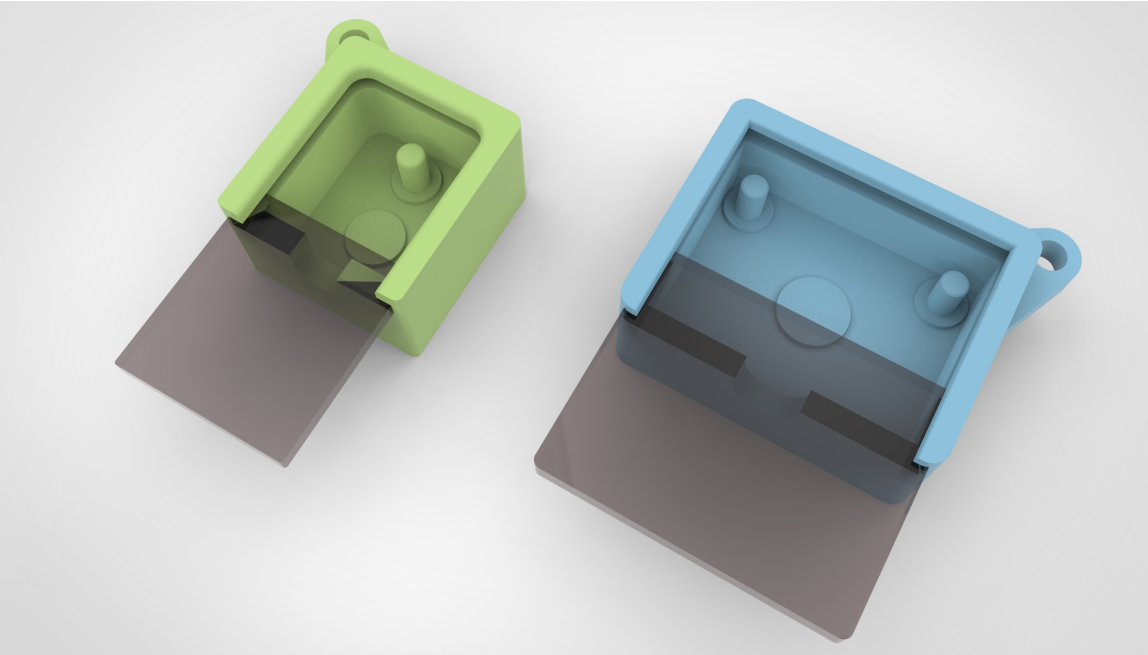


*Figure 24: Cases for microphone and accelerometer (opened)*

Figure 25 shows the possible implementation of our second solution in the pump. As mentioned earlier, the system can be implemented with both sensors inside or outside the Pycom case. It depends on the results of the subsequent testing period as we have to make sure that the two sensors can accurately detect when the pump is working.

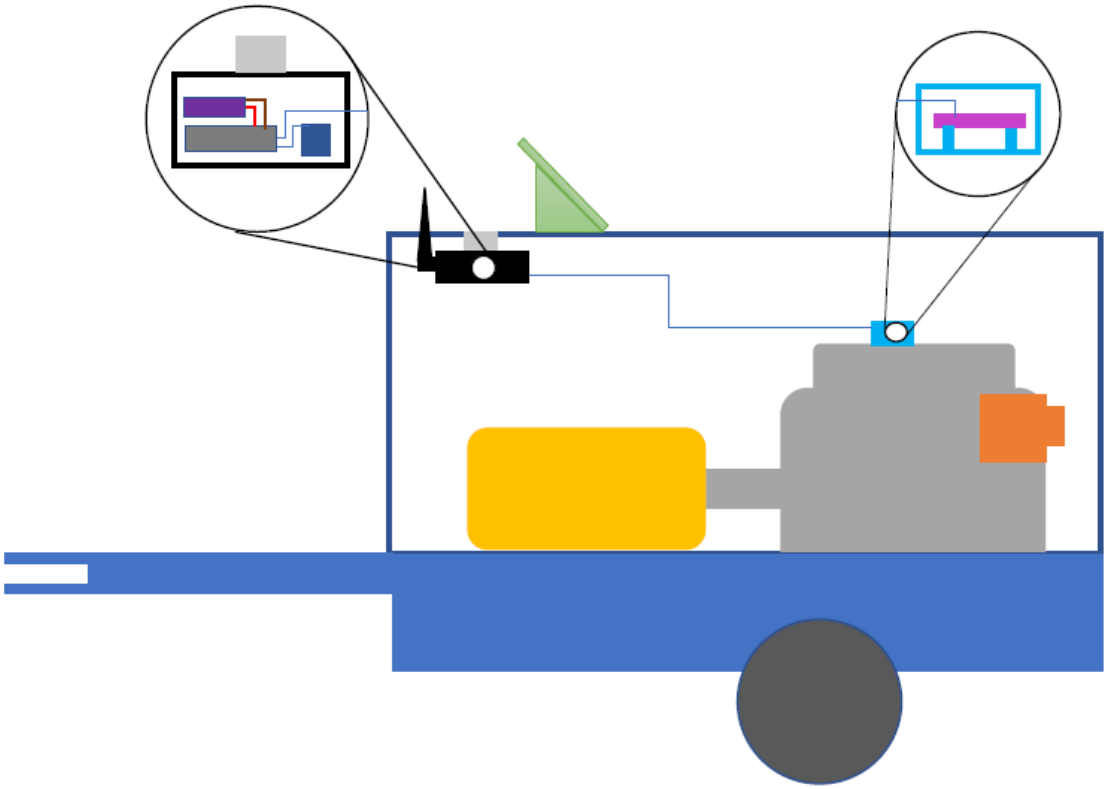


*Figure 25: Possible installation of the device with the two sensors*

*Table 6: Key of colours for Figure 25*

| KEY – Microphone and vibration | |
|---|---|
| **Black** = Pycom microcontroller with case | **Red cable** = plus-potential (+) |
| **Light grey** = magnet | **Light blue** = case |
| **Dark grey** = engine | **Pink** = vibration sensor |
| **Purple** = Lipo battery | **Dark blue** = microphone |
| **Orange** = alternator | **Darker grey** = Pycom microcontroller |
| **Yellow** = pump | **Green** = solar panel (additional) |
| **Brown cable** = ground (-) | |

## Energy supply: The battery and solar panel

As shown in Figure 25 there is a battery and a solar panel (optional) needed for the power supply. The suggestion for an appropriate battery is a Lipo (Lithium polymer) battery. This type of battery can store a huge amount of energy and has small dimensions, which makes it easier to implement in the Pycom case. These batteries (Figure 26) are made to be used by microcontrollers and are fitted with a JST connector (the same connector as used in the Pycom microcontroller). [35]

Technical data:

- Dimensions: 62 x 38.5 x 8.3mm
- 3.7V nominal voltage
- 100mm JST terminated lead
- Short circuit protection
- Nominal capacity: typical 2000 mAh
- Price: 13.50 £


*Figure 26: Lithium Polymer battery*

## Wiring scheme – Microphone and Vibration sensor

```
#ADXL345 -> LoPy
#GND -> GND
#Vin -> 3.3v
#SDA -> G16
#SCL -> G17

#SPW2430 -> LoPy
#GND -> GND
#VIN -> 3V3
#DC  -> G3
```

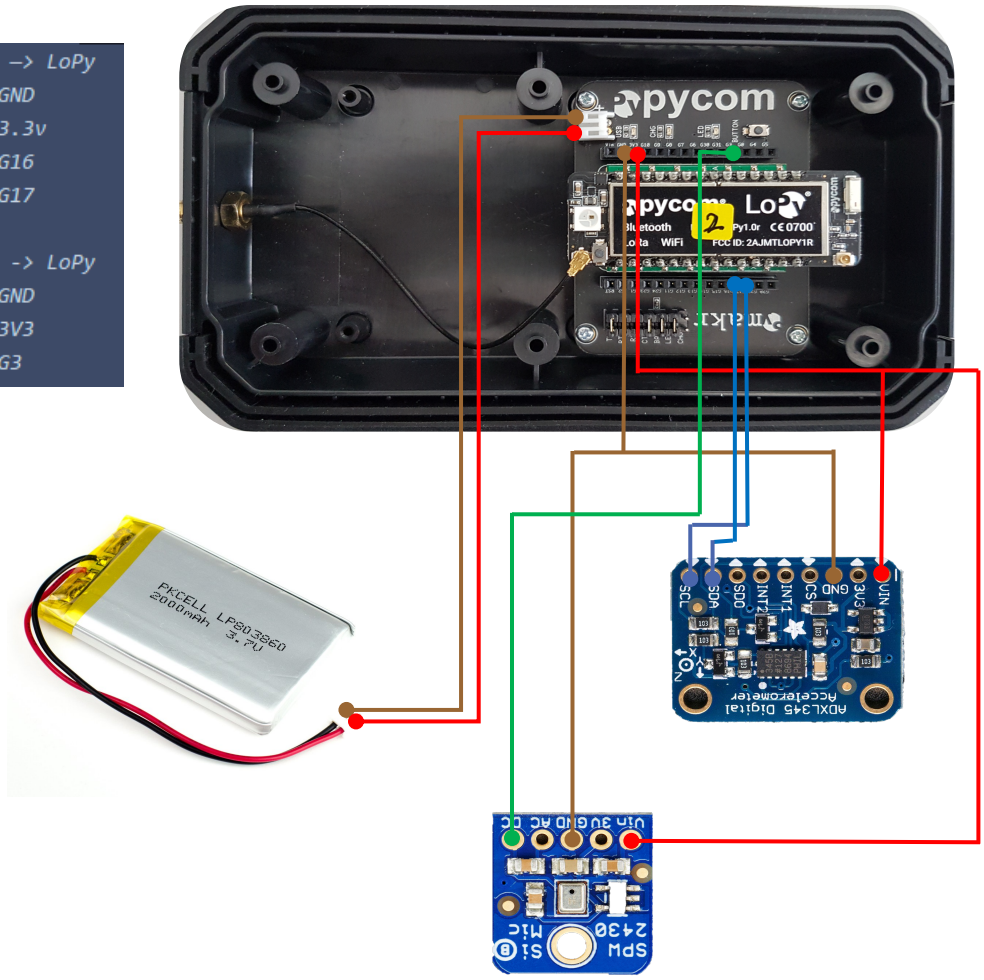



*Figure 27: Wiring diagram for microphone and vibration sensor*

### 4.1.3. Example

The following example how the water measurement is calculated after a measured working time of the pump (the alternator solution is used).

Sunday, 10.06.18

- at 14:00:00; engine starts and Pycom switches on
- at 14:00:30; machine pumps water with 9600 l/min
- at 20:00:00; engine stops and Pycom switches off

#### Result

The Pycom was sending real time data continuously to the gateway during the engine and the pump were working

- total working time: 05:59:30 hours = 359.5 min
- total water consumption = 359.5 min x 9600 l/min = 3,451,200 l/min = 3,451.2 $m^3$

The first 30 seconds after the engine start is the expected time the pump needs to get water and the Pycom microcontroller to switch on. After this time the controller starts sending data and counting the water consumption.

## 4.2. Hardware

### 4.2.1. Sensors

#### Accelerometer

There were a few different sensors that we could have used in our system to sense vibrations and sound. To sense vibrations a suitable accelerometer would need to be used, we decided on going with the ADXL345 triple-axis accelerometer from Analog Devices [36], shown in Figure 28. It is small and thin with it being only 3mm x 5mm x 1mm in size making it perfect for small enclosures. With it also being low-power only consuming as low as 23µA in measurement mode and 0.1µA in standby mode, it is perfect for mobile applications like this. Its full resolution, where resolution increases with *g* range, up to 13-bit resolution at ±16*g* enables measurement of inclination change of less than 1.0°, making it an excellent sensor to detect vibrations. The accelerometer would come from Analog Devices in a form of a chip, however the accelerometer we developed the system with was already preassembled and tested in the form of a mini breakout board from a company called Adafruit. The preassembled package comes with digital $I^2C$ interface and Serial Peripheral Interface (SPI) breakout, an onboard 3.3V regulator and logic-level shifting circuitry, making it a perfect choice for interfacing with any 3V or 5V microcontroller.

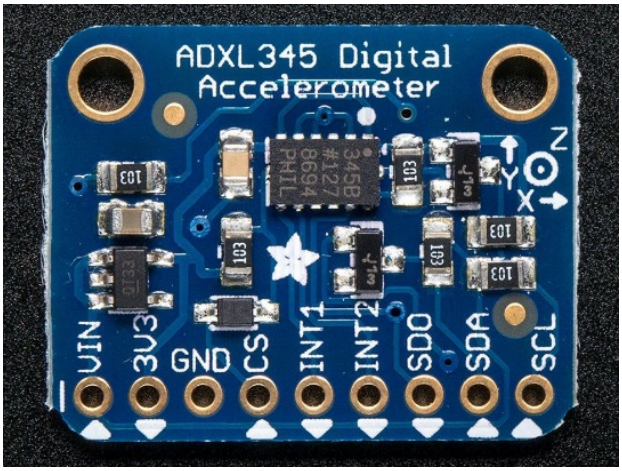



*Figure 28: ADCL345 3-axis accelerometer*

The sensor is deemed triple-axis because it has three axes of measurements, in the X, Y and Z planes. The sensitivity level can be set to either ±2g, ±4g, ±8g, or ±16g; the lower range gives more resolution for slow movements, the higher range is good for high speed tracking. It is for these features just mentioned that the use of the ADXL345 would be more than suitable for the vibration sensing application in the system. The breakout board which is shown above is larger than just having the Chip itself, however is still just 25mm x 19mm x 3.14mm and weighing only 1.27 grams and has some mounting holes for easy attachment.

## Microphone

The microphone going to be used is the SPW2430 [37], shown in Figure 29. Again, the company that takes the chip and provides an already assembled and pre-tested breakout board that makes it easier to implement to the system. It is a really small MEMS (Microelectro-Mechanical Systems) microphone, that instead acquiring sound like a typical microphone, they detect sound and convert it to voltage, making it easier to read and also assemble because it does not require an amplifier or bias resistor.

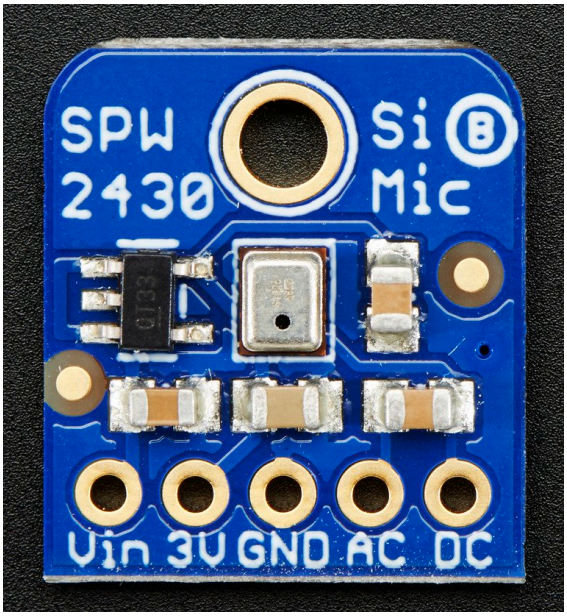



*Figure 29: Adafruit Silicon MEMS Microphone Breakout*

The board provides both DC and AC coupled outputs to suit whatever audio sensing purpose, for the use with microcontrollers analogue input, the DC coupled output would be used. With being even smaller than the accelerometer at 15.8mm x 14.1mm x 2.9mm, having the required outputs and having a frequency response of 100Hz to 10,000kHz, it is perfect for the intended purpose of detecting engine noise in our system.

### 4.2.2. Node/gateway

The sensors must have somewhere to connect to that would both provide power and process the data from the sensors. The system will also need to be able to transmit data using LoRa communications. To tackle this, the system is going to use the LoPy microcontroller from Pycom. LoPy is a triple bearer MicroPython enabled microcontroller, triple bearer meaning that it can transmit data using three communication methods, those being LoRa, Wifi and Bluetooth making it a perfect Internet of Things platform.

For a small microcontroller – at only 55mm x 20mm x 3.5mm in dimensions – the LoPy has some very impressive specifications. It has powerful processing power with Espressif ESP32 chipset which is a dual-core system with two Harvard Architecture Xtensa LX6 CPUs, one acts as a network procecessor to handle the WiFi connectivity and the IPv6 stack, which isn't going to be used for the systems main function but is a useful thing to have, the other processor will be used to run the user application. The chipset has an extra ultra-low power coprocessor that can monitor GPIOs, the ADC channels and control most of the internal peripherals during a deep-sleep mode which only consumes 25µA. It has 24 GPIO pins and 8 12-bit ADC channels making it perfect to connect sensors to.

Other features of the LoPy (Figure 30) include that it has 512KB of RAM and external flash of 4MB, which does not sound like a lot however in terms of microcontrollers, the size of memory is more than adequate for storing and running programs as they are typically only a few kilobytes. [38]

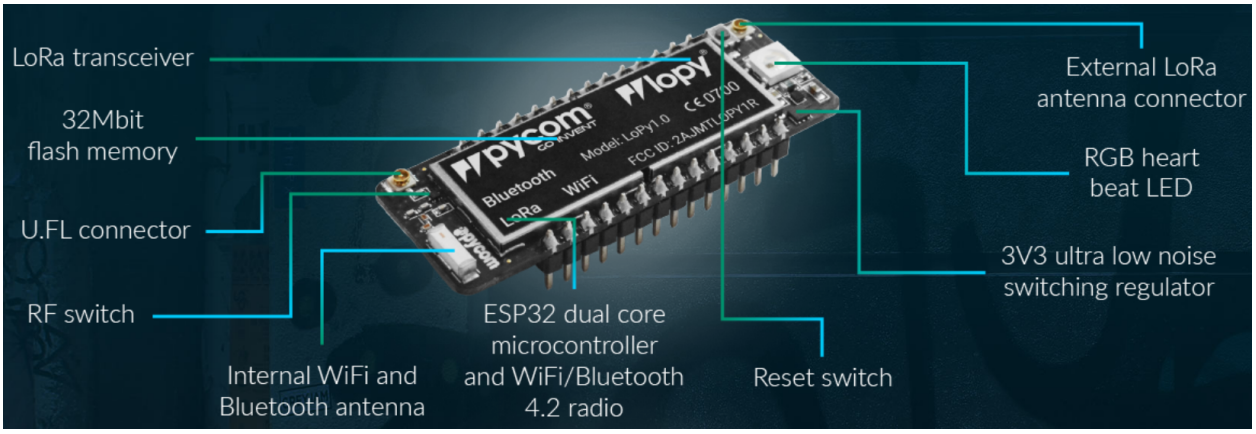



*Figure 30: LoPy microcontroller[38]*

The main feature that made the decision easier to implement the LoPy into our system rather than other microcontrollers or micro PCs is that it can transmit data using LoRa straight out the box.

The LoPy can be set up in such a fashion that is could be used as a LoRa node or nano-gateway which is useful because a dedicated gateway would not be required which could be really expensive. The device has a Semtech LoRa transceiver SX1272 which is capable of communicating as a node of upto 40km and as a nano-gateway, if the LoPy is set up as a nano-gateway iy has a capacity of upto 100 nodes. Although the LoRa transceiver has impressive specifications it only draws 15mA while active and only 1µA in standby.

The LoPy will be connected to an expansion board which adds some functionality and makes it easier to get started.  It is the Expansion Board 2.0 from Pycom (Figure 31) and is compatable with most of their microcontrollers. It allows the LoPy to be USB and LiPo battery powered, the USB port has a USB to serial converter meaning that programs and data can be transferred to the LoPy straight from a computer. It has a MicroSD card socket which could be used to expand the memory of the LoPy. The microcontroller's 24 GPO pins are also easily accessible with jumper connections which is good for testing and prototyping quickly without the need to solder. The LoPy will connect directly into the expansion board's female headers. [39]

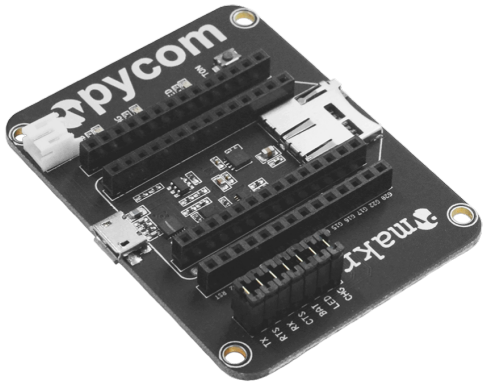



*Figure 31: Pycom Expansion Board 2.0*

The expansion board is not necessarily needed however provides some very useful features such as having a USB to serial converter and a battery connection.

So that the LoPy can send and receive LoRa signals an external antenna is needed. The antenna used in this system comes as part of a kit from Pycom (Figure 32). It is a universal LoRa and Sigfox antenna kit that can be used with most of Pycom's microcontrollers that support these communication methods. An antenna is essential for LoRa communication on the LoPy because trying to use LoRa without an antenna could severely damage the LoPy's chipset. [40]

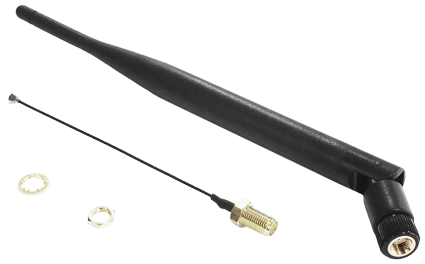


*Figure 32: Antenna Kit*

## 4.3. Software

To get the final working solution it is not quite as simple as connecting a power source and sensors to start determining if the pump is on. To get closer to the final solution, a program that will be loaded into the LoPy's memory is required.

The program to get both the microphone and accelerometer to detect sound and vibrations from the engine or motor from the pump will need to take the signals from the sensors and determine if the pump is active or not.

### 4.3.1. Vibration and Sound Sensing

The main aim is to conclude if the pump is on by determining if there is vibrations and sound being generated by the motor or engine. How the program is going to determine this is shown by the flow chart illustrated below.

First of all, the accelerometer will check for vibrations by determining if there is any sudden movement. If there has been a sufficient number of vibrations in a n second period then the microphone will go onto checking for any sound, if not then it will keep checking for any vibrations. If there has been enough sound in the following n second period, then it will send data to the gateway stating that the pump is on. If there hasn't been enough sound, then the program will go back to checking for vibrations and not send anything to the gateway.

The way that the program should work in practice is if the time period it will sense for vibrations and sound is set to 10 seconds, then the accelerometer will determine if there have been vibrations in a 10 second period, if there has then the microphone will determine if there has been enough sound generated by the engine or motor in the following 10 second period. These steps are shown in Figure 33. So therefore, if the pump is on the LoPy should transmit data stating that the pump is on every 20 seconds.

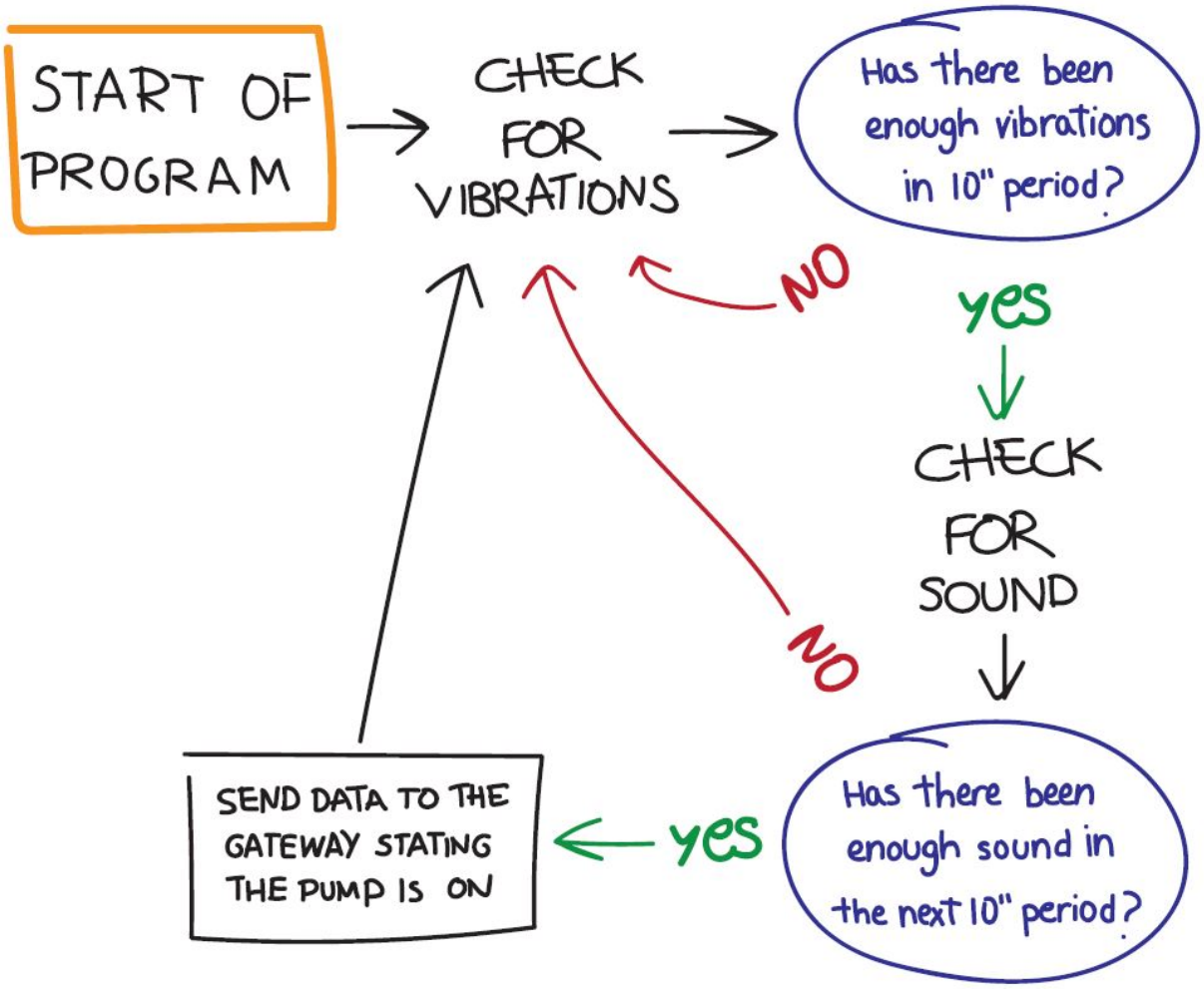



*Figure 33: Diagram describing the code for the sensor solution*

There are three files that are contained within the LoPy's memory for this system and are all essential for the sensing of vibrations and sound from the accelerometer and microphone respectively. The three files are:

- boot.py – the code inside the file is executed first when the LoPy boots up. It normally sets up various configuration options
- main.py – the code in main.py is the main script that will contain the LoPy's main program
- ADXL345.py – this is a library file that is essentially a driver for the ADXL345 accelerometer and is needed for it to work as it should. The microphone does not need a driver file to work.

The first file that runs when the LoPy boots up is boot.py, it is just a short file to initialize the UART connection, this is essentially getting serial access to the LoPy using USB, this is useful for analysis and testing of what is happening. This is shown in Figure 34 and can be found in Appendix B1.

```
1  import os
2
3  #initiates the UART (USB) connection
4  uart = machine.UART(0, 115200)
5  os.dupterm(uart)
```

*Figure 34 - boot.py of the system*

main.py as stated contains the main program. To start there are comments stating the connections from the sensors to the expansion board that the LoPy is connected to, this is shown in Figure 35. (The full code is in Appendix B2). This section will go into detail about what each part of the program achieves.

```
1  #ADXL345 -> LoPy
2  #GND -> GND
3  #Vin -> 3.3v
4  #SDA -> G16
5  #SCL -> G17
6
7  #SPW2430 -> LoPy
8  #GND -> GND
9  #VIN -> 3V3
10 #DC  -> G3
```

*Figure 35 - Comments showing the connections between LoPy and sensors*

Next the libraries that will be used in the program are declared for the code to run and also for the LoRa to communicate, these libraries are shown in Figure 36.

```
12  #Libraries needed
13  import time
14  import machine
15  import pycom
16  import adxl345
17  # LoRa libraries
18  from network import LoRa
19  import socket
20  import binascii
21  import struct
```

*Figure 36 - Labraries included in main.py*

Figure 37 shows how the program starts, first by turning off the LoPy's flashing LED, also known as the heartbeat. This is always on by default and is used to let the user know the LoPy is in working order. It is turned off on this occasion to save power.

The I2C and Pin variables are set to machine, the machine module contains functions related to hardware. Then the microphone variables are created with integer values to be used when detecting sound.

An ADC object is created, this is when an analogue to conversion process takes place, i.e. when an analogue sound is turned in to digital signals that the microcontroller can process. This object is used to create an analogue pin on 'P16' on the LoPy, which is pin 'G3' on the expansion board.

```
23  pycom.heartbeat(False)  #save power
24
25  #Set the I2C and Pin to machine.
26  I2C = machine.I2C
27  Pin = machine.Pin
28
29  #microphone variables
30  noise_samples = 2000
31  noise_damping = 30
32  noiseCount = 0
33  stateNoise = False
34
35  adc = machine.ADC()             # create an ADC object
36  noise_pin = adc.channel(pin='P16')   # create an analog pin on P16
```

*Figure 37 - How the main body of main.py starts*

A timer will be used in the program to determine when to sense for vibrations or sound and to determine when to send data. Figure 38 shows how the variables for the timer are created and a 'timeThreshold' is also created, this can be changed to affect how long the LoPy senses for vibration and sound.

The $I^2C$ bus is initialised which is used for the connection between the LoPy and the accelerometer. 'x', 'y' and 'z' variables are created which will be used to check if accelerometer has detected any movement.

```
#Timer variables
Timer = machine.Timer
chrono = Timer.Chrono()
timeThreshold = 10  #affects how long senses for vibration and sound

#initialize the I2C bus
i2c = I2C(0, I2C.MASTER, baudrate=100000)

#x,y,z variables for accelerometer to compare against its current x,y,z values
oldx = 0
oldy = 0
oldz = 0
```

*Figure 38 - Time variables and i2c bus initialised*

Next, the variables that will be used for the vibration detection are declared. The LoPy indication LED is set to be used, this would not be included in the final program however it will be used for testing.

The LoRa communication is initiated, then the parameters for ABP are created. As described, ABP stands for Authentication by Personalisation, and in order for the LoPy to communicate with the gateway, the ABP parameters that are associated with the gateway are needed. These are the device address, the network switch key and the application switch key and are created by the gateway.
The program then attempts to join the network using the three ABP parameters. This full process is shown in Figure 39.

```
#Set this to false to turn off indication lights
lightVar = True
if lightVar == True:
    print('Turn lights on')

#Initiates lora communication
print("Initialiing LoRa")
lora = LoRa(mode=LoRa.LORAWAN) #RCZ1/RCZ3 Europe / Japan / Korea

# create ABP authentication params
# ABP stands for Authentication By Personalisation. No checks!
# device address, network and application keys for gateway
dev_addr = struct.unpack(">l", binascii.unhexlify('26 01 1D 59'.replace(' ','')))[0]
nwk_swkey = binascii.unhexlify('1E 89 96 93 9B 2E A3 9C FA 1E EC 5A A9 99 99 3C'.replace(' ',''))
app_swkey = binascii.unhexlify('60 1D B1 EA 11 C5 BC 1B B5 67 E6 A7 11 5F F3 2B'.replace(' ',''))
# join a network using ABP (Activation By Personalization)
lora.join(activation=LoRa.ABP, auth=(dev_addr, nwk_swkey, app_swkey))
```

*Figure 39 - How the connection to the gateway is setup*

A LoRa socket is created, this will be used to send data using LoRa and the data rate is set to 5 bytes per transmission, which is more than enough for the program's function because it will send only small amounts of data. This can be seen in Figure 40.

The LoRa settings are printed, this is just for testing and will not be included in the final system's program.

```
# create a LoRa socket
s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)
# set the LoRa data rate
s.setsockopt(socket.SOL_LORA, socket.SO_DR, 5)

#check LoRa settings
print("LoRa BW:")
print(lora.bandwidth())
print("LoRa freq")
print(lora.frequency())
print("LoRa coding rate")
print(lora.coding_rate())
print("LoRa preamble symbols")
print(lora.preamble())
print("LoRa spreading factor")
print(lora.sf())
print("Last received packet stats")
print(lora.stats())
print("LoRa MAC")
print(lora.mac())
```

*Figure 40 - LoRa socket created*

The function that will be used to get noise values from the microphone is defined, as shown in Figure 41. The global variables used for taking the number of samples and the noise damping are

declared in the function. The integers ‘vmin’ is set to 10000 and ‘vmax’ to 0, these will be used to determine the noise level. The for loop runs from ‘I’=0 to the value for ‘noise_samples’, which in this case is 2000. The reason for this is to create a time buffer between obtaining noise values and also to get a reliable value.

A variable ‘val’ is created and gets its value from the microphone which was set up as the noise pin. ‘vmax’ is then determined by comparing and assigning as the maximum value between the previous value for ‘vmax’ and the value from the microphone. ‘vmin’ is determined in a similar way, however is assigned to the minimum value between the previous value and the value from the microphone.

```
98  #get noise function
99  def getNoise():
100     global noise_samples
101     global noise_damping
102
103     vmin = 10000
104     vmax = 0
105
106     for i in range(0,  noise_samples):  #0to2000
107         val = noise_pin()               # read an analog value
108         vmax = max(vmax,  val)
109         vmin = min(vmin,  val)
110
111     noise1 = vmax - vmin    #noise is measured in voltage form
112     level = int(noise1 / noise_damping)
113
114     return level
```

*Figure 41 - The 'getNoise' function*

The noise level is calculated by finding the difference between the ‘vmax’ and ‘vmin’ values and then is divided by the noise damping value of 30 to get a more reasonable number. An example of this is if the value received from the microphone is 1200 then ‘vmax’ would be the 1200 and ‘vmin’ 1000, therefore the noise level would be 200/30 which is 6.67, as an integer it would be 7.

Next, the function that will be used to detect any vibrations coming from the pump system is declared. It first pulls the global variables that have been created outside the function. The variables used to get the x, y and z axes values is created as ‘axes’, it uses the $I^2C$ connection. They are then used to create variables for each axis and get a value from the accelerometer for the x, y and z axes.

An if statement is used to compare the ‘x’, ‘y’ and ‘z’ values to the old values to check if they have marginally changed, if they have then it will mean the accelerometer has sensed vibration. The other if statement simply acts as a time buffer to prevent the program from getting results in quick succession. If there has been a vibration, then the program will start the timer if it has not already and add to the ‘hitCount’ which will be used to determine if there has been a sufficient amount of vibration. It prints the hit count and the time of the hits for testing purposes. The old x, y and z variables are updated with the current ‘x’, ‘y’ and ‘z’ values after the comparison is done.
This full function is shown in Figure 42.

```
116 #get vibration function
117 def getVibration():
118     global hitCount, measureCounter, threshold, measureThreshold, stateMotion, oldx, oldy, oldz
119 
120     data = adxl345.ADXL345(i2c)
121     axes = data.getAxes(True)       #function using i2c to get value of accelerometer axes
122 
123     x = axes['x']   #get x,y,z axes values
124     x = abs(x)
125     y = axes['y']
126     y = abs(y)
127     z = axes['z']
128     z = abs(z)
129     measureCounter += 1
130     #check if any axes values have changed
131     if (x <= oldx-0.1) or (x >= oldx+0.1) or (y <= oldy-0.1) or (y >= oldy+0.1) or (z <= oldz-0.1) or (z >= oldz+0.1):
132         if measureThreshold <= measureCounter:      #acts as time buffer
133             chrono.start()
134             hitCount = hitCount + 1     #increase hitcount
135             print('I have been hitten')
136             print('My count is ')
137             print(hitCount)         #print hitcount for testing
138             print(chrono.read(), '\n')  #print time for testing
139             measureCounter = 0      #reset time buffer
140 
141     oldx = x    #the x,y,z variables used for comparison become the axis values just obtained
142     oldy = y
143     oldz = z
```

*Figure 42 - The 'getVibration' function*

An infinite loop is used to execute the program and to use the ‘getSound’ and ‘getVibration’ functions and use them to determine if the LoPy should send data to the gateway to state the pump is on. It starts by calling the ‘getVibration’ which, as explained, will determine if there have been vibrations and then add to the ‘hitCount’.
Then the program determines if there have been three or more vibrations after a 10 second period, this can be changed by altering the time threshold. In a 10 second period the maximum number of vibrations that can be recorded is six, so setting the minimum ‘hitCount’ to three will determine if the pump is on or if the accelerometer picked up vibrations from elsewhere. If there has been a

sufficient number of vibrations, then the 'stateMotion' Boolean will change to true. If there have not been enough vibrations, then the timer will reset, and the program will sense for vibrations again, shown in Figure 43.

```
while True:

    getVibration()  #function to detect if there has been vibrations to begin with
    if timeThreshold<=chrono.read() and hitCount >= 3:  #if there has after 10 seconds then move on
        totnoise = 0
        stateMotion = True
    elif timeThreshold<=chrono.read() and hitCount < 3:#if there hasn't reset and try again
        hitCount = 0
        chrono.reset()
        chrono.stop()
        stateMotion = False

```

*Figure 43 - Start of the infinite loop*

If there has been a sufficient amount of vibrations, then the program will then get noise from the microphone, shown in Figure 44. The 'getNoise' function is inside a for loop that counts up to 10 and adds the noise values of noise up to slow down the rate that noise is being received and to also add accuracy, this is because it is easier to determine if the pump is on or not by adding up several slightly larger noise values to get one large value instead of determining if it is on from one slightly large noise value.

The value that the added noise values has to be at least 60, if it is over 60 then the program will print "on" for testing purposes and add one to the 'noiseCount'. The 'noiseCount' is used to determine if the LoPy should send data to the gateway or not. If the 'noiseCount' is 2 or greater in the next 10 second period then the Boolean 'stateNoise' will become true, it will be used to determine if the LoPy should send data to the gateway or not.

```
#if time is less than 20 seconds and there has been vibrations detect if there is noise
    if 2*timeThreshold>=chrono.read() and stateMotion == True:
        totnoise = 0
        for j in range(10):     #for loop to slow down the rate noise is been recieved and for accuracy
            noise = getNoise()  #gets noise from function
            totnoise += noise
        print(totnoise)
        print(chrono.read())        #print time for testing purposes
        if totnoise >= 60:      #if there is no noise then the value should be less than 60
            print('on')
            noiseCount = noiseCount + 1
        if noiseCount >= 2:
            stateNoise = True
```

*Figure 44 - How the LoPy determines if there has been enough sound*

The next part, shown in Figure 45, uses the 'stateNoise' Boolean to determine whether the timer should reset and start looking for vibrations again or continue. If there has been enough noise, i.e. 'stateNoise' is true then the 'stateLora' Boolean becomes true.

```
174     #This function checks if it is allowed to send a message via lora (after 20 seconds of first vibration)
175     if  2*timeThreshold <= chrono.read() and stateNoise == True :
176         stateLora = True
177         chrono.stop()
178     elif 2*timeThreshold <= chrono.read() and stateNoise == False:
179         stateMotion = False
180         hitCount = 0
181         pycom.rgbled(0x000000) #light turned off
182         chrono.reset()
183         chrono.stop()       #timer reset
```

*Figure 45 - How the program resets if there is no vibrations*

The final part is sending the data to the gateway, it uses the Boolean 'stateLora' to determine this. The program gets the LoPy to send the 'hitCount', 'noiseCount' and then just simply "on". The only data that would be sent in the final program would just simply be "on" or even just simply "1", sending the 'hitCount' and 'noiseCount' is just for testing purposes only.

At the end, the 'stateLora', 'stateNoise' and 'stateVibration' Booleans as well as the 'hitCount', 'noiseCount' and timer are reset. This will mean that the infinite loop will restart and the 'getVibration' function will run. This is shown below in Figure 46.

```
185     #Then it checks if it is allowed to send a message
186     if stateLora == True:
187         #Send hit count to lora gateway
188         print('I am going to send this hit count ')
189         print(hitCount)
190         print(chrono.read(), '\n')      #print time for testing purposes
191         #Send the hitcount to lora
192         hitCount = str(hitCount)
193         s.send("Hit" + hitCount)    #send hitcount to gateway
194 
195         print('I am going to send the noise count')
196         print(noiseCount)
197         print(chrono.read(), '\n')      #print time for testing purposes
198         #Send the noisecount to lora
199         noiseCount = str(noiseCount)
200         s.send("Noise count" + noiseCount)  #send noisecount to gateway
201 
202         print('The pump is on')
203         s.send("on")
204         stateLora = False
205         stateMotion = False
206         stateNoise = False
207         hitCount = 0
208         noiseCount = 0          #reset lora, vibration, sound indications
209         chrono.reset()
210         chrono.stop()           #reset timer
```

*Figure 46 - How the data to the gateway is sent if there has been vibrations and sound*

At the very end, the LED is controlled depending on what the Boolean values are, or in other words what is currently being run by the LoPy. This is shown in Figure 47, and useful for testing purposes.

```
212     if lightVar == True:            #change led colours for testing purposes
213         if (stateMotion == True) and (stateNoise == False) and (stateLora == False):
214             #There has been motion
215             pycom.rgbled(0x007f00) #green
216         elif (stateMotion == True) and (stateNoise == True) and (stateLora == False):
217             #There has been noise
218             pycom.rgbled(0x7f7f00) #yellow
219         elif (stateMotion == True) and (stateNoise == True) and (stateLora == True):
220             #Lora can send message
221             pycom.rgbled(0x7f0000) #red
```

*Figure 47 - LED control for testing*

The library file ADXL345.py is provided in Apendix B4. It was premade and was available online [36], and was very useful to get the accelerometer to return values in the program.

### 4.3.2. Line in

As stated, the second solution involves having a connection coming from a power outlet on the pump, like the alternator, straight to the LoPy which would both power the microcontroller and also provide an indication that the pump is in use as there is only power generated when the pump is on.

The program is more straight forward than the microphone and accelerometer solution program as for this solution the LoPy will only be connected to power. The program will get the LoPy to send data to the gateway every so often stating that it is on, i.e. every 20 seconds. The flow chart below shows this. These steps are shown in Figure 48.

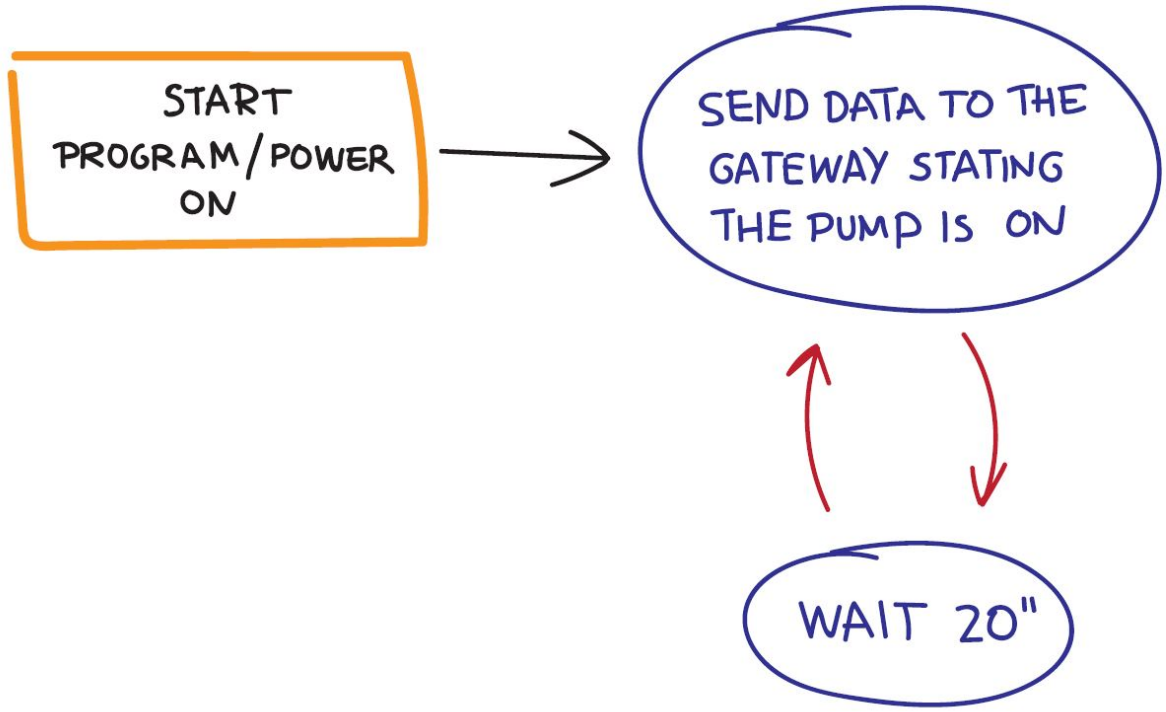



*Figure 48: Diagram for Accelerometer solution*

There are three files that are contained within the LoPy's memory for this system. These are:

- Boot.py – provides the same purpose and is the same as boot.py in the other system
- Main.py – will send data to the gateway every 20 seconds

The libraries that are included in main.py are not as extensive as the other system, there is only a need for time keeping and LoRa related libraries. As shown below in Figure 49. The full program can be found in Appendix B3.

```
1  import time
2  import machine
3  # LoRa things
4  from network import LoRa
5  import socket
6  import binascii
7  import struct
```

*Figure 49 - libraries on the line-in main.py*

The LoRa initialisation and gateway set-up are exactly the same as it is in the accelerometer and microphone system, this is shown in Figure 50.

```
#Initiates lora communication
lora = LoRa(mode=LoRa.LORAWAN) #RCZ1/RCZ3 Europe / Japan / Korea
# create an ABP authentication params
# ABP stands for Authentication By Personalisation. No checks!
# device address, network and application keys for gateway
dev_addr = struct.unpack(">l", binascii.unhexlify('26 01 19 AF'.replace(' ','')))[0]
nwk_swkey = binascii.unhexlify('65 A6 43 92 DA 40 65 7D 7E 34 EE 7E E2 F9 2D 9B'.replace(' ',''))
app_swkey = binascii.unhexlify('66 5D A2 D7 9D F0 A5 4C 41 28 EF 72 01 91 10 12'.replace(' ',''))
# join a network using ABP (Activation By Personalization)
lora.join(activation=LoRa.ABP, auth=(dev_addr, nwk_swkey, app_swkey))
# if lora.has_joined(): this is always true in ABP, no checking
# create a LoRa socket
s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)
# set the LoRaWAN data rate
s.setsockopt(socket.SOL_LORA, socket.SO_DR, 5)


#check LoRa settings
print("LoRa BW:")
print(lora.bandwidth())
print("LoRa freq")
print(lora.frequency())
print("LoRa coding rate")
print(lora.coding_rate())
print("LoRa preamble symbols")
print(lora.preamble())
print("LoRa spreading factor")
print(lora.sf())
print("Last received packet stats")
print(lora.stats())
print("LoRa MAC")
print(lora.mac())
```

*Figure 50 - LoRa initialisation and gateway set-up*

The program's main function starts by initialising a timer using the appropriate libraries and then starting the timer. The 'timeThreshold', measured in seconds, is set to 10.

An infinite loop is used to send data every n seconds. To determine when to send data, an If statement is used to run code when the timer reaches the 'timeThreshold', which is 10 seconds. The LoPy then prints "on" and sends "Pump is on" to the gateway. In reality, there will not be a need to send as much data for the data is irrelevant, the gateway will only want to know if the pump is on or not, and if the LoPy is transmitting data then it will assume it is on. So ,in practice, the LoPy could send data as simple as "1" or just "on". The timer resets back to 0 and counts back up to the 'Time threshold'. This is all shown in Figure 51.

```
44  #initialise and start timer
45  Timer = machine.Timer
46  chrono = Timer.Chrono()
47  chrono.start()
48  timeThreshold = 10
49
50  while True:
51      #sends data every 10 seconds
52      if  timeThreshold <= chrono.read():
53          print('on')
54          s.send("Pump is on")
55          chrono.reset()
```

*Figure 51 - LoPy sending data to gateway every 10 seconds*

## 4.4. IoT Platform: The Things Network

The Things Network (Figure 52) is a website/network we use to receive and examine data sent from our device. It's a community-based network that serves to help people all around the world regarding IoT applications that use LoRaWAN technology [37].


*Figure 52: The Things Network logo*

Initiated in June 2015 by Wienke Giezeman, The Things Network (or TTN for short) is the first open-source, decentralized infrastructure for the Internet of Things and now serves over 38 thousand developers, has a network of about 3600 gateways and has been used to deploy over 21 thousand IoT applications all around the world [37].

The network is composed of a number of gateways around the world which have their own coverage areas. LoRa equipped devices within these areas can send data to nearby gateways, which will send that data onto The Things Network platform, all for free.

The Things Network is also a member of the LoRa Alliance, an association dedicated to promoting LPWAN technology regarding its standardization and interoperability as previously mentioned.

There are two ways to connect to the network: Over-The-Air-Activation (OTAA) and Activation By Personalization (ABP). Security and their ease of implementation is what differentiates the two.

*Table 7: Advantages and disadvantages of OTAA and ABP*

| | ADVANTAGES | DISADVANTAGES |
|---|---|---|
| OTAA | The network generates and sends encryption keys, having better network security compared to ABP. | More complex to set up |
| ABP | Easy and quick to connect to the network | The encryption keys used to communicate with the network are preconfigured in the device |

In the context of our project, we have chosen to use ABP due to the fact that we're merely coming up with a prototype. It makes sense to use the easiest method to test our device, and security isn't our main concern during this semester.

We interact with The Things Network to analyse data sent from our device. This is done through the gateway that our supervisor Dr. Tuleen Boutaleb has installed at the university.

## 4.5. System Cost

One of the requirements from SEPA was for the system to be low-cost, the price of the components was a large factor when deciding what ones to implement in the system.
For both of the two systems, they had four of the same components, these were:

- Pycom LoPy - £40 [43]
- LoRa antenna - £10.50 [44]
- LoPy case - £13.30 [46]
- Pycom expansion board - £22.50 [47]

The total of these four components is £86.30 and is the total price for the 'line-in' system as there are no other sensors required.
For the vibration sound sensing method, the price of the two sensors are:

- ADXL345 accelerometer - £17.50 [48]
- SPW2430 microphone - £5 [49]

The price of the two sensors brings the price up to £108.80. Adding a possible portable power source such as a LiPo battery and small solar panel, the price would still be less than £200 which was the aim.

# 5. Tests & Results

To test the program works how it should, and that there are no bugs in the code. The Pymakr package for the text editor Atom is used to communicate with the LoPy, allowing to run files straight from a PC and for the LoPy to communicate back and forth with the PC using the built in command line REPL.

It is through the use of the 'print' functions in the program's code that allows the LoPy to print text on the REPL, which helps for testing.

At the start of each solution, after initialising LoRa and setting up the gateway, the LoRa settings are printed. This is shown below in Figure 53, it is not used for any practical purpose, just for checking LoRa is working.

```
Turn lights on
Initialiing LoRa
LoRa BW:
0
LoRa freq
868000000
LoRa coding rate
1
LoRa preamble symbols
8
LoRa spreading factor
7
Last received packet stats
(rx_timestamp=0, rssi=0, snr=0.0, sfrx=0, sftx=0, tx_trials=0, tx_power=14, tx_time_on_air=0, tx_counter=0, tx_frequency
=0)
LoRa MAC
b'p\xb3\xd5I\x9a\xcb\x9a\xf6'
```

*Figure 53 - LoRa settings printed on REPL*

Focusing on the microphone and accelerometer solution, the LoPy will first use the accelerometer to sense for vibrations. While sensing for vibrations, the indication LED on the LoPy is green, as shown in Figure 54. If there are no vibrations, then the REPL will remain blank.

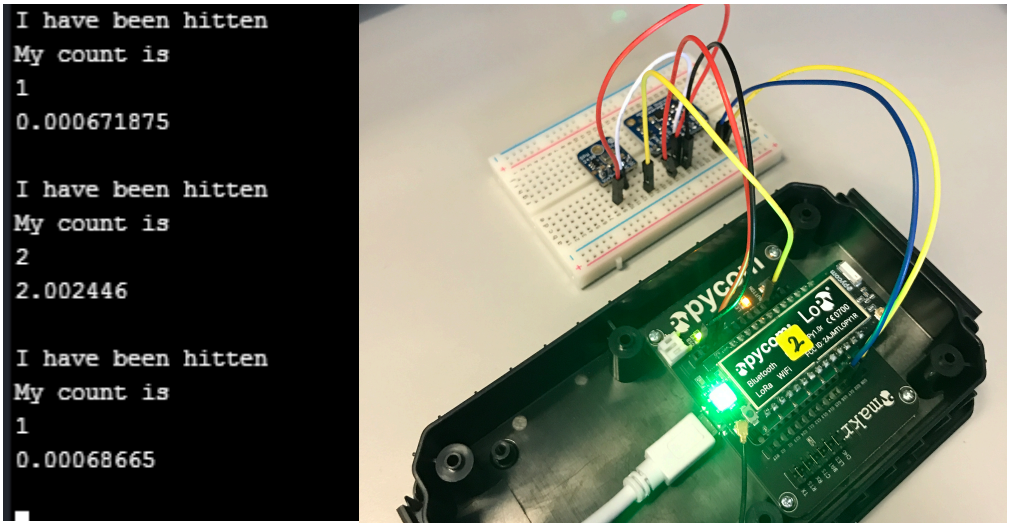



*Figure 54 - Green LED when sensing for vibrations*

If it senses vibrations, however not enough in a 10 second period, which is when the hit count is less than 3, then it will clear the hit count and timer and start sensing for vibrations again. The timer value is shown in the bottom line every time the hit count is incremented.

Figure 46 shows the transition of sensing for vibrations to sensing for sound when the timer reaches 10 seconds and the hit count is 3 or more. When the LoPy starts sensing sound from the microphone, the indication LED turns yellow. This is shown in Figure 47.

```
2.002731

I have been hitten
My count is
3
4.001016

I have been hitten
My count is
4
6.01299

I have been hitten
My count is
5
8.022422

52
11.4136
```

*Figure 56- Hit count incrementing whilst sensing vibrations*

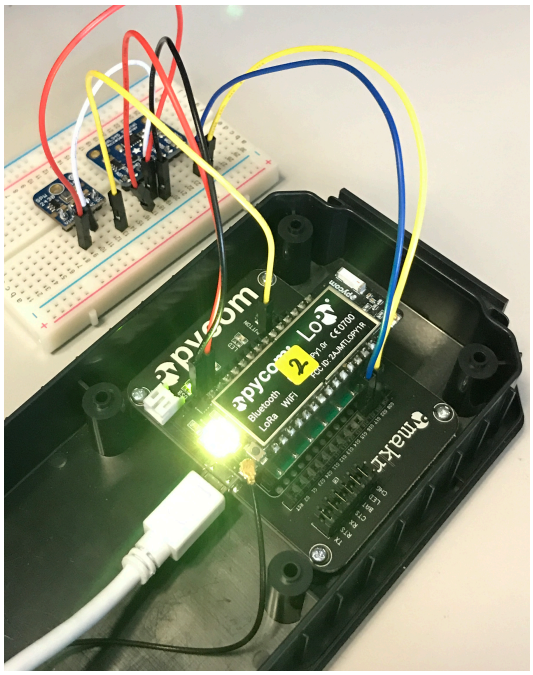


*Figure 55 - LED turns yellow to when sensing sound*

In the next part when sensing for sound and there is not enough sound in the following 10 second period (when the noise count is less than 2), then it will return back to sensing for vibrations, showed in Figure 47. Then the LED will return back to being green as seen in Figure 45.

```
51
11.41481
54
12.83255
48
14.24987
54
15.6669
52
17.08421
54
18.50139
52
19.92216
54
21.33937
I have been hitten
My count is
1
```

*Figure 57 - Resetting after sensing no sound*

If the LoPy senses enough sound from the microphone, then as in Figure it will send both the hit count and noise count, aswell as sending a message stating the pump is on. The LED momentarily turns red, as shown in Figure before turning green again and sensing for vibrations.

```
14.24987
on
77
15.66767
on
51
17.0855
117
18.50268
on
58
19.92044
49
21.33759
I am going to send this hit count
6
21.33784

I am going to send the noise count
4
21.33784

The pump is on
```

*Figure 58 - Sending data to the gateway after sensing sufffiicient amount of vibrations and sound*

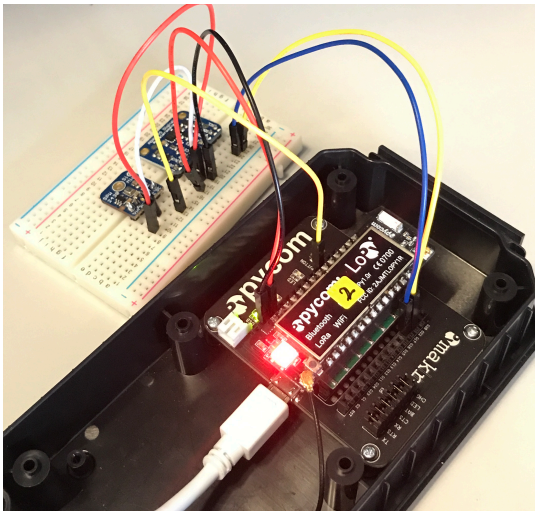


*Figure 59 - LED turns red when sending data*

Table 8 shows the test results with what is happening at each stage and what happens after each possible action

*Table 8 - Test results for the vibration and sound sensing solution*

| Stage | Action | Next Step | LED colour | Next LED colour | Time period |
|---|---|---|---|---|---|
| 1.Vibration sensing | Less than 3 hit count | 1.Continue sensing for vibrations | Green | Green | 0 to 10 seconds |
| 1.Vibration sensing | 3 or more hit count | 2.Sound Sensing | Green | Yellow | 0 to 10 seconds |
| 2.Sound sensing | Less than 2 noise count | 1.Go back to Vibration sensing | Yellow | Green | 10 to 20 seconds |
| 2.Sound sensing | 2 or more noise count | 3.Send data using LoRa to gateway | Yellow | Red | 10 to 20 seconds |
| 3.Send data using LoRa to gateway | None | 1.Go back to Vibration Sensing | Red | Green | 20 to 23 seconds |

Moving onto the power line in program, where the LoPy will send data every so often if the LoPy is powered. It is a very simple program and does not require a lot of testing.

Figure shows how the REPL looks when there is power to the LoPy, with “on” getting printed every 20 seconds and “Pump is on” being sent to the gateway at the same time.


*Figure 60 - REPL showing "on" getting displayed every 20 seconds*

The Things Network, as mentioned, is the platform that will be used to interpret the data. Figure shows the screenshot of what The Things Network console looks like when there is data being sent to the gateway. The payload is the data that the gateway receives and is in hexadecimal. The payload in Figure 61 is the hexadecimal equivalent of “Pump is on”.

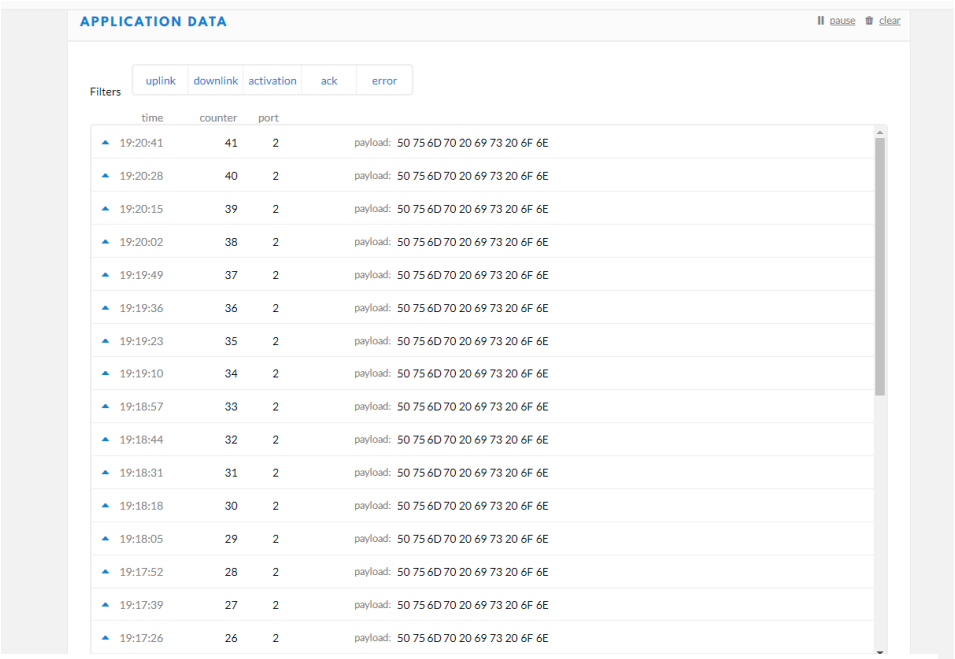



*Figure 61 - Screenshot of TTN when data is being sent to the gateway*

# 6. Conclusions and Future Work

The main aim of the project was to provide a solution to the lack of a real-time water usage monitoring system in Scottish Agriculture. During this project, a working prototype comprising of a range of electronics was developed while taking into consideration the requirements that the Scottish Environment Protection Agency desired.

Two variations of the prototype have been developed; one comprising of a microphone and accelerometer, and the other utilising the power generated by the engine of the pump. The two solutions have a common goal of detecting when and for how long the water pump is being used in the extraction of water from a source. The working time of the pump will be used to determine the amount of water extracted as its water flow is constant.

After testing the prototype in the university, the next step would be to prepare the hardware in order to test the system under real conditions; such as 3D printing the sensor cases, permanent wiring and installation of the components.
After this, an ample testing of the system under real conditions inside the pump is needed to make sure that everything works correctly. Due to the large number of advantages, the solution that uses the power generated by the engine should be tested first, followed by the solution that uses the sensors. If the testing process is successful, the solution will be ready to be implemented in the farms that are in collaboration with SEPA.

Although this system would solve the main problem, there are some improvements that could be implemented. In some cases, farmers move their pumps in order to use different water sources; it would be useful to implement a GPS module to track what water source they are extracting water from. Improvements could also be made to the program to minimise power consumption and increase the reliability of our system.

The project has been educational on multiple levels and has given us a valuable experience that we are more likely to use or interact with in the future. It has been a challenge to come up with a low-cost, optimal solution that can effectively measure agricultural water consumption in Scotland. Being able to work on an innovative and relatively new technology has been interesting and insightful.

# References


[1] “Scottish Environment Protection Agency,” [Online]. Available: https://www.sepa.org.uk/about-us/.

[2] “Modelling the influence of irrigation abstractions on Scotland's water resources,” [Online]. Available: https://search-proquest-com.gcu.idm.oclc.org/docview/1943407344?accountid=15977&rfr_id=info%3Axri%2Fsid%3Aprimo.

[3] “Geography in California,” [Online]. Available: https://www.thoughtco.com/geography-of-california-1435723.

[4] “Agriculture is 80 percent of water use in California. Why aren’t farmers being forced to cut back?,” [Online]. Available: ttps://search-proquest-com.gcu.idm.oclc.org/docview/1943407344?accountid=15977&rfr_id=info%3Axri%2Fsid%3Aprimo..

[5] “California Farms Use How Much Water? Nobody Really Knows,” [Online]. Available: http://www.circleofblue.org/2015/world/california-farms-use-how-much-water-nobody-really-knows/.

[6] “El País (Spanish newspaper),” [Online]. Available: https://elpais.com/economia/2018/03/11/actualidad/1520793945_819539.html.

[7] “Fertirrigación – Marco A. Oltra Cámara,” [Online]. Available: http://www.fertirrigacion.com/sistemas-de-riego-en-espana/.

[8] ““El sistema de riego localizado” – Moisés Mario Fernándes De Sousa & Guillermo García González de Lena,” [Online]. Available: http://www.serida.org/pdfs/6003.pdf.

[9] “Agricultural land use in Scotland,” [Online]. Available: http://www.gov.scot/Topics/Statistics/Browse/Agriculture-Fisheries/agritopics/LandUseAll.

[10] “Global Map of Irrigation Areas (GMIA),” [Online]. Available: http://www.fao.org/nr/water/aquastat/irrigationmap/GBR/.

[11] “U.N declares internet access a human right,” [Online]. Available: https://www.wired.com/2011/06/internet-a-human-right/.

[12] “World's fastest ISPs,” [Online]. Available: http://vintaytime.com/worlds-fastest-internet/.

[13] “IoT: a comparison of communication technologies,” [Online]. Available: https://blog.montem.io/2017/03/10/internet-of-things-a-comparison-of-communication-technologies/.

[14] “How fast is 4G?,” [Online]. Available: https://www.4g.co.uk/how-fast-is-4g/.

[15] “LPWAN Benefits,” [Online]. Available: https://www.iotforall.com/lpwan-benefits-vs-iot-connectivity-options/.

[16] “RF Sensitivity,” [Online]. Available: https://www.link-labs.com/blog/rf-sensitivity-for-m2m.

[17] “RF Calculator,” [Online]. Available: https://www.everythingrf.com/rf-calculators/dbm-to-watts.

[18] “LPWAN Technology,” [Online]. Available: http://cdn2.hubspot.net/hubfs/427771/LPWAN-Brochure-Interactive.pdf?__hstc=&__hssc=&hsCtaTracking=046055b9-7e1c-47d1-a077-dbf7f6b3ba00%7Cc45caeb9-8be0-436c-a0ab-419e760eb414.

[19] “What is LoRa?,” [Online]. Available: https://www.link-labs.com/blog/what-is-lora.

[20] “LoRa IoT,” [Online]. Available: https://www.kerlink.com/vision-technologies/lora-iot/.

[21] “iC880A - LoRaWAN® Concentrator 868MHz,” [Online]. Available: https://wireless-solutions.de/products/radiomodules/ic880a.html.

[22] “About LoRa Alliance,” [Online]. Available: https://lora-alliance.org/about-lora-alliance.

[23] “LoRaWAN accross the globe,” [Online]. Available: https://www.i-scoop.eu/internet-of-things-guide/iot-network-lora-lorawan/#LoRa_from_a_leading_LPWAN_enabler_in_Europe_to_increasing_presences_across_the_globe.

[24] “Overview MicroPython,” [Online]. Available: https://learn.adafruit.com/micropython-basics-what-is-micropython/overview.

[25] “Pumps - Jones Engineering,” [Online]. Available: http://www.jonesengineering.co.uk/new-equipment/irrigation/pumps/.

[26] “PMXT - High pessure multistage horizontal pumps,” [Online]. Available: http://www.caprari.com/products/surface/high-pressure-multistage-horizontal-pumps-2.

[27] “Piezo Vibration Sensor,” [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/LDT_Series.pdf.

[28] “Assemtech Vibration Sensor,” [Online]. Available: https://uk.rs-online.com/web/p/vibration-sensors/4553665/.

[29] “Adafruit ADXL345,” [Online]. Available: https://www.adafruit.com/product/1231.

[30] “SparkFun MEMS Microphone Breakout,” [Online]. Available: https://www.sparkfun.com/products/9868.

[31] “SparkFun Sound Detector,” [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Sound/LMV324.pdf.

[32] “Adafruit,” [Online]. Available: https://www.adafruit.com/product/2716.

[33] “Troubleshooting the ignition warning light,” [Online]. Available: https://www.howacarworks.com/ignition-system/troubleshooting-the-ignition-warning-light.

[34] “Buck Voltage Regulator,” [Online]. Available: http://www.droking.com/DC-to-DC-8-22V-to-1-15V-12V-5V-Buck-Voltage-Regulator-Mini-Adjustable-Step-Down-Converter-Module?search=Buck%20Converter%20Step%20Down%20adjustable&description=true.

[35] “Lipo battery pack,” [Online]. Available: https://shop.pimoroni.com/products/lipo-battery-pack?utm_medium=cpc&utm_source=googlepla&variant=20429082183&gclid=EAIaIQobChMIsNf43rnp2gIVA5ztCh3-IAgOEAQYBCABEgJGlfD_BwE.

[36] “ADXL345 library,” [Online]. Available: https://www.hackster.io/lokefar/knock-knock-who-s-sending-to-sigfox-a26eed.

[37] [Online]. Available: https://www.thethingsnetwork.org.

[38] [Online]. Available: https://www.sepa.org.uk/about-us/how-we-work/our-research/.

[39] [Online]. Available: https://www.pmi.org/about/learn-about-pmi/what-is-project-management.

[40] [Online]. Available: http://www.crew.ac.uk/sites/default/files/sites/default/files/publication/Climate_change_and_water_demand-supply.pdf.

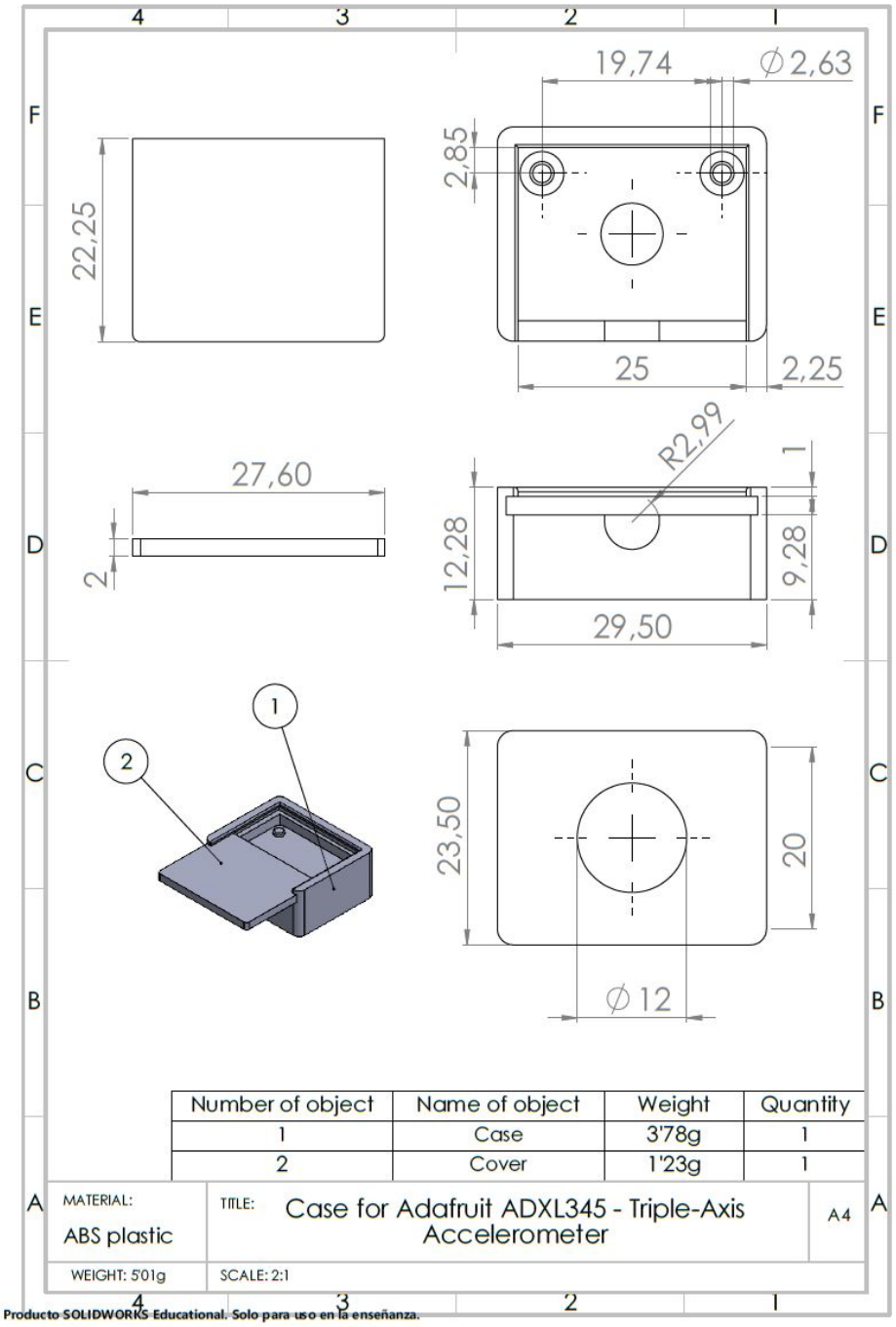
[41] [Online]. Available: https://www.sepa.org.uk/.

[42] “Agricultural land use in Scotland,” [Online]. Available: http://www.gov.scot/Topics/Statistics/Browse/Agriculture-Fisheries/agritopics/LandUseAll.

[43] “Pycom datasheet,” [Online]. Available: https://pycom.io/wp-content/uploads/2018/03/Pycom_Specsheet_LoPy.pdf.

# Appendix

## Appendix A1 – Accelerometer case technical drawing




| Number of object | Name of object | Weight | Quantity |
|---|---|---|---|
| 1 | Case | 3'78g | 1 |
| 2 | Cover | 1'23g | 1 |

| | | |
|---|---|---|
| MATERIAL: ABS plastic | TITLE: Case for Adafruit ADXL345 - Triple-Axis Accelerometer | A4 |
| WEIGHT: 501g | SCALE: 2:1 | |

## Appendix B1 – boot.py of both solutions

```
import os
import machine

#initiates the UART (USB) connection
uart = machine.UART(0, 115200)
os.dupterm(uart)
```

# Appendix B2 – main.py of accelerometer and microphone solution

```
#ADXL345 —> LoPy
#GND —> GND
#Vin —> 3.3v
#SDA —> G16
#SCL —> G17

#SPW2430 -> LoPy
#GND -> GND
#VIN -> 3V3
#DC  -> G3

#libraries needed
import time
import machine
import pycom
import adxl345
# LoRa libraries
from network import LoRa
import socket
import binascii
import struct

pycom.heartbeat(False)              #save power

#Set the I2C and Pin to machine.
I2C = machine.I2C
Pin = machine.Pin

#microphone variables
noise_samples = 2000
noise_damping = 30
noiseCount = 0
stateNoise = False

adc = machine.ADC()           # create an ADC object
noise_pin = adc.channel(pin='P16')   # create an analog pin on P16

#Timer variables
Timer = machine.Timer
chrono = Timer.Chrono()
timeThreshold = 10     #affects how long senses for vibration and sound

#initialize the I2C bus
i2c = I2C(0, I2C.MASTER, baudrate=100000)

#x,y,z variables for accelerometer to compare against its current x,y,z values
oldx = 0
oldy = 0
oldz = 0

#accelerometer variables
threshold = 1                       #threshold is measued in g for accelerometer
stateMotion = False
hitCount = 0
stateLora = False
measureCounter = 500
measureThreshold = 500

#Set this to false to turn off indication lights
lightVar = True
if lightVar == True:
        print('Turn lights on')
```

```
#Initiates lora communication
print("Initialiing LoRa")
lora = LoRa(mode=LoRa.LORAWAN) #RCZ1/RCZ3 Europe / Japan / Korea

# create ABP authentication params
# ABP stands for Authentication By Personalisation. No checks!
# device address, network and application keys for gateway
dev_addr = struct.unpack(">l", binascii.unhexlify('26 01 19 AF'.replace(' ','')))[0]
nwk_swkey = binascii.unhexlify('65 A6 43 92 DA 40 65 7D 7E 34 EE 7E E2 F9 2D 9B'.replace(' ',''))
app_swkey = binascii.unhexlify('66 5D A2 D7 9D F0 A5 4C 41 28 EF 72 01 91 10 12'.replace(' ',''))
# join a network using ABP (Activation By Personalization)
lora.join(activation=LoRa.ABP, auth=(dev_addr, nwk_swkey, app_swkey))
# if lora.has_joined(): this is always true in ABP, no checking
# create a LoRa socket
s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)
# set the LoRa data rate
s.setsockopt(socket.SOL_LORA, socket.SO_DR, 5)

#check LoRa settings
print("LoRa BW:")
print(lora.bandwidth())
print("LoRa freq")
print(lora.frequency())
print("LoRa coding rate")
print(lora.coding_rate())
print("LoRa preamble symbols")
print(lora.preamble())
print("LoRa spreading factor")
print(lora.sf())
print("Last received packet stats")
print(lora.stats())
print("LoRa MAC")
print(lora.mac())

#get noise function
def getNoise():
  global noise_samples
  global noise_damping

  vmin = 10000
  vmax = 0

  for i in range(0,  noise_samples):              #0to2000
    val = noise_pin()            # read an analog value
    vmax = max(vmax,  val)
    vmin = min(vmin,  val)

  noise1 = vmax - vmin              #noise is measured in voltage form
  level = int(noise1 / noise_damping)

  return level

#get vibration function
def getVibration():
        global hitCount, measureCounter, threshold, measureThreshold, stateMotion, oldx, oldy, oldz

        data = adxl345.ADXL345(i2c)
        axes = data.getAxes(True)                       #function using i2c to get value of accelerometer axes

        x = axes['x']               #get x,y,z axes values
        x = abs(x)
        y = axes['y']
        y = abs(y)
        z = axes['z']
        z = abs(z)
        measureCounter += 1
```

```
        #check if any axes values have changed
        if (x <= oldx-0.1) or (x >= oldx+0.1) or (y <= oldy-0.1) or (y >= oldy+0.1) or (z <= oldz-0.1) or (z >= oldz+0.1):
                if measureThreshold <= measureCounter:          #acts as time buffer
                        chrono.start()
                        hitCount = hitCount + 1                 #increase hitcount
                        print('I have been hitten')
                        print('My count is ')
                        print(hitCount)                         #print hitcount for testing
                        print(chrono.read(), '\n')          #print time for testing
                        measureCounter = 0                  #reset time buffer

        oldx = x    #the x,y,z variables used for comparison become the axis values just obtained
        oldy = y
        oldz = z


print('Starting the loop')

while True:

        getVibration()          #function to detect if there has been vibrations to begin with
        if timeThreshold<=chrono.read() and hitCount >= 3:      #if there has after 10 seconds then move on
                totnoise = 0
                stateMotion = True
        elif timeThreshold<=chrono.read() and hitCount < 3:#if there hasn't reset and try again
                hitCount = 0
                chrono.reset()
                chrono.stop()
                stateMotion = False

#if time is less than 20 seconds and there has been vibrations detect if there is noise
        if 2*timeThreshold>=chrono.read() and stateMotion == True:
                totnoise = 0
                for j in range(10):                 #for loop to slow down the rate noise is been recieved and for accuracy
                        noise = getNoise()      #gets noise from function
                        totnoise += noise
                print(totnoise)
                print(chrono.read())                #print time for testing purposes
                if totnoise >= 60:                  #if there is no noise then the value should be less than 60
                        print('on')
                        noiseCount = noiseCount + 1
                if noiseCount >= 2:
                        stateNoise = True

        #This function checks if it is allowed to send a message via lora (after 20 seconds of first vibration)
        if  2*timeThreshold <= chrono.read() and stateNoise == True :
                stateLora = True
                chrono.stop()
        elif 2*timeThreshold <= chrono.read() and stateNoise == False:
                stateMotion = False
                hitCount = 0
                pycom.rgbled(0x000000) #light turned off
                chrono.reset()
                chrono.stop()                       #timer reset

        #Then it checks if it is allowed to send a message
        if stateLora == True:
                #Send hit count to lora gateway
                print('I am going to send this hit count ')
                print(hitCount)
                print(chrono.read(), '\n')                  #print time for testing purposes
                #Send the hitcount to lora
                hitCount = str(hitCount)
                s.send("Hit" + hitCount)            #send hitcount to gateway

                print('I am going to send the noise count')
```

```
            print(noiseCount)
            print(chrono.read(), '\n')                  #print time for testing purposes
            #Send the noisecount to lora
            noiseCount = str(noiseCount)
            s.send("Noise count" + noiseCount)          #send noisecount to gateway

            print('The pump is on')
            s.send("on")
            stateLora = False
            stateMotion = False
            stateNoise = False
            hitCount = 0
            noiseCount = 0                              #reset lora, vibration, sound indications
            chrono.reset()
            chrono.stop()                               #reset timer

if lightVar == True:                                    #change led colours for testing purposes
            if (stateMotion == False) and (stateNoise == False) and (stateLora == False):
                        #There has been motion
                        pycom.rgbled(0x007f00) #green
            elif (stateMotion == True) and (stateNoise == False) and (stateLora == False):
                        #There has been noise
                        pycom.rgbled(0x7f7f00) #yellow
            elif (stateMotion == True) and (stateNoise == True):
                        #Lora can send message
                        pycom.rgbled(0x7f0000) #red
```

# Appendix B3 – main.py of line in program

```
import time
import machine
# LoRa things
from network import LoRa
import socket
import binascii
import struct

print("Initialiing LoRa")

#Initiates lora communication
lora = LoRa(mode=LoRa.LORAWAN) #RCZ1/RCZ3 Europe / Japan / Korea
# create an ABP authentication params
# ABP stands for Authentication By Personalisation. No checks!
# device address, network and application keys for gateway
dev_addr = struct.unpack(">l", binascii.unhexlify('26 01 19 AF'.replace(' ','')))[0]
nwk_swkey = binascii.unhexlify('65 A6 43 92 DA 40 65 7D 7E 34 EE 7E E2 F9 2D 9B'.replace(' ',''))
app_swkey = binascii.unhexlify('66 5D A2 D7 9D F0 A5 4C 41 28 EF 72 01 91 10 12'.replace(' ',''))
# join a network using ABP (Activation By Personalization)
lora.join(activation=LoRa.ABP, auth=(dev_addr, nwk_swkey, app_swkey))
# if lora.has_joined(): this is always true in ABP, no checking
# create a LoRa socket
s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)
# set the LoRaWAN data rate
s.setsockopt(socket.SOL_LORA, socket.SO_DR, 5)


#check LoRa settings
print("LoRa BW:")
print(lora.bandwidth())
print("LoRa freq")
print(lora.frequency())
print("LoRa coding rate")
print(lora.coding_rate())
print("LoRa preamble symbols")
print(lora.preamble())
print("LoRa spreading factor")
print(lora.sf())
print("Last received packet stats")
print(lora.stats())
print("LoRa MAC")
print(lora.mac())

#initialise and start timer
Timer = machine.Timer
chrono = Timer.Chrono()
chrono.start()
timeThreshold = 10

while True:
   #sends data every 10 seconds
   if  timeThreshold <= chrono.read():
      print('on')
      s.send("Pump is on")
      chrono.reset()
```

# Appendix B4 – ADXL345.py

```
#The address of the ADXL345 given in the datasheet
ADXL345_ADDR = 0x53


#The bytes for making the ADXL345 send at 100Hz output data rate
BW_RATE_100HZ = 0x0B

#The address for making changes to POWER_CTL
POWER_CTL = 0x2D
#The byte "code" for starting the measurements
MEASURE = 0x08

#The address for changing the DATA_FORMAT. This is used together with the ranges
DATA_FORMAT = 0x31

#The address where the measurement data starts from. Each axis has two bytes for the given value
AXES_DATA = 0x32

#The address for accessing and setting the bandwidth rate
BW_RATE = 0x2C

#Decide the range of measurements ie the precision. Possible options
#2G
RANGE_2G = 0x08
#4G
RANGE_4G = 0x09
#8G
RANGE_8G = 0x2A
#16G
RANGE_16G = 0x0F

SCALE_MULTIPLIER = 0.004

#Standard gravity constant for going from G-force to m/s^2
EARTH_GRAVITY_MS2 = 9.80665



class ADXL345:

        def __init__(self, i2c):
                self.i2c = i2c
                self.addr = ADXL345_ADDR
                self.setBandwidthRate(BW_RATE_100HZ)
                self.setRange(RANGE_8G)
                self.enableMeasurement()

        def enableMeasurement(self):
                self.i2c.writeto_mem(self.addr, POWER_CTL, bytes([MEASURE]))

        def setBandwidthRate(self, rate_flag):
                self.i2c.writeto_mem(self.addr, BW_RATE, bytes([rate_flag]))

        def setRange(self, range_flag):
                self.i2c.writeto_mem(self.addr, DATA_FORMAT, bytes([range_flag]))

        def getAxes(self, gforce = False):
                bytes = self.i2c.readfrom_mem(self.addr, AXES_DATA, 6)
                x = bytes[0] | (bytes[1] << 8)
                if(x & (1 << 16 - 1)):
                        x = x - (1<<16)
```

```
y = bytes[2] | (bytes[3] << 8)
if(y & (1 << 16 - 1)):
        y = y - (1<<16)

z = bytes[4] | (bytes[5] << 8 )
if(z & (1 << 16 - 1)):
        z = z - (1<<16)

x = x * SCALE_MULTIPLIER
y = y * SCALE_MULTIPLIER
z = z * SCALE_MULTIPLIER

if gforce == False:
        x = x * EARTH_GRAVITY_MS2
        y = y * EARTH_GRAVITY_MS2
        z = z * EARTH_GRAVITY_MS2

x = round(x,4)
y = round(y,4)
z = round(z,4)

return {"x": x, "y": y, "z": z}
```