INF-3910-3-3 - COMPUTER SCIENCE
SEMINAR: IOT SERVICES WITH LORAWAN
NETWORK AND COMPATIBLE EMBEDDED
DEVICES AND SENSORS

UNIVERSITY OF TROMSØ

Animal tracker project

Thomas Bye Nilsen
Valter Berg

May 9, 2018

Contents

1 Introduction **2**
- *1.1 Problem statement* . . . 2
- *1.2 Background* . . . 3
- *1.3 Member roles* . . . 4

2 Milestone 2 **5**
- *2.1 Design decisions* . . . 5
- *2.2 Overall flow and states* . . . 7
- *2.3 Estimated time for work* . . . 7
- *2.4 Technical design* . . . 7

3 Milestone 3 **9**
- *3.1 Sensor side* . . . 9
- *3.2 Telenor MIC Cloud* . . . 11
- *3.3 Front-end* . . . 11

4 Milestone 4 **12**
- *4.1 Sensor side* . . . 12
- *4.2 Analysis of battery consumption* . . . 15
- *4.3 Front-end and server* . . . 16

5 Conclusion **17**
- *5.1 Ending state* . . . 17
- *5.2 Future work* . . . 17
- *5.3 Relevant work* . . . 18

Appendices **19**

List of Figures

1 Pytrack data sheet . . . 3
2 DS18B20 data sheet . . . 3
3 Architecture . . . 7
4 Temperatue sensor connection data sheet . . . 13
5 Temperatue sensor connection data sheet . . . 14
6 Final result of Pytrack . . . 15
.1 The box with a Pytrack and battery package strapped onto it . . . 19
.2 View on how the widgets in MIC displayed observational data. 20
.3 View of website showing history path and latest position of one device. . . . 21

1 Introduction

This report describes the work that lays the foundation for our project. The project is about tracking certain animals in their natural habitat without hindering their way of living. The motivation for our project is the usefulness of the services provided by the system we develop. Animal keepers can track down and survey individuals as they are in movement in their natural habitat.

1.1 Problem statement

The problem we want to solve is to gather and present data regarding the environment and movement patterns of the animals using IoT devices. We want to make the IoT devices fit as many species as possible. We will track animals by emitting GPS coordinates, accelerometer data, battery voltage and temperature from the IoT devices. All data will be delivered to a front-end system through the LoRaWAN network.

We will primarily focus on gathering the four attributes from an arbitrary number of observational units; we will not focus on creating a mesh network of IoT devices.

The complexity of the project is not too high for a practical completion. The technology needed is available and there seems to be not too much work to achieve it. After one week we will design the architecture of the system and make a list of equipment we need. We expect that we will acquire enough low-level details of the technical requirements after a few weeks to implement most of the core functionalities by milestone 3. The remaining functionalities will be postponed to milestone 4, if any. Soldering of electronics will take about one day. Designing and printing the box will take about one day as well. Designing and implementing the front-end system will take between one and two weeks. Bridging the transmission between the different components in system will take a few days. The time estimates include testing and debugging.

In addition, we have been in contact with Rovdata, a company that tracks wild animals for the purpose of population control and positional tracking. We also contacted some farmers who work with sheep herding to see if they could contribute with any opinions on the concept and design. The farmers were positive to our design and concept, but they were concerned about the total cost of implementing such a system on all their sheep, as they have 300 to 600 individuals grassing each summer. That is why we also intend to keep the cost as low as possible, which also should be an advantage with using cheap LoPy devices and a low-cost network such as LoRa.

1.2 Background

Hardware

For the processing device we use a Lopy, a MicroPython enabled microcontroller, with LoRa connectivity as one of its features. The LoPy is mounted on a Pytrack extension board which provide a GNSS and Glonass GPS system, and a three axis accelerometer. The temperature sensor is a DS18B20 one-wire digital temperature sensor. It reports degrees in Celsius in the range -55C to 125C (+/-0.5C)[1]. Figure 1 displays the data sheet for a Pytrack and figure 2 displays the data sheet for DS18B20.

Figure 1: Pytrack data sheet

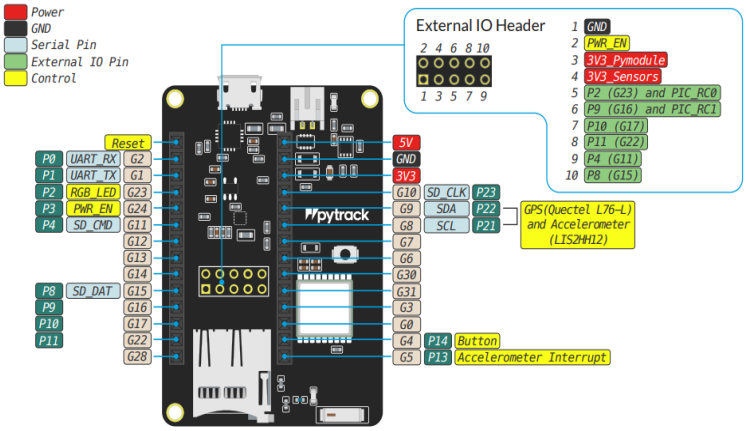
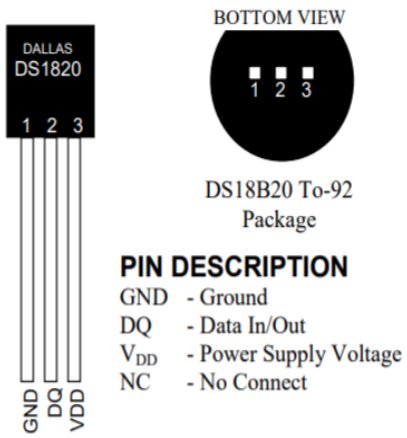


Figure 2: DS18B20 data sheet



LoRaWAN LoRaWAN is a long range wide area network protocol designed to wirelessly connect small battery powered devices to the internet. It targets important internet of things features such as bi-directional communication, end-to-end security and mobility. LoRaWAN fits in the second and third layer of the OSI model, implemented on top of LoRa or FSK modulation in industrial, scientific and medical radio bands[2].

Mqtt and Managed IoT Cloud
MQTT, or Message Queue Telemetry Transport, is a machine to machine/internet of things connectivity protocol based on publish-subscribe messaging. This protocol works on top of the TCP/IP protocols and enables communication with remote locations where small footprints are required or the bandwidth is limited. To do the publish-subscribe messaging, there is a message broker which distributes messages to connected clients based on the topic of the message[3]. The Managed Iot Cloud (MIC), provided by Telenor, uses MQTT as its main protocol in the underlying AWS Iot Message broker for thing-to-backend communication. The Managed IoT Cloud platform enables managing of the IoT broker and data is routed through exposed MQTT topics. The MIC cloud platforms provides a user interface to view reported date from Lopy and gives you the opportunity to format the data sent to the Mqtt broker[4].

1.3 Member roles

The team members are Valter Berg and Thomas Bye Nilsen. The member agree on the tasks that are to be solved. Below is a table of features that we will be working on and who will be working on it. For some features, mainly in the beginning of the project, both members conduct. This approach seems convenient to give both members a better understanding of the technical details, e.g how to emit simple GPS data from lopy device to IoT cloud. Both members will update each other on the state of the feature they work on and provide each other with input.

Feature	**Member**
Contact farmers	Valter
Contact Rovdata	Thomas
Test GPS and accelerometer, write basic script code for LoPy	Both
Solder temperature sensor, write test code, perform testing	Thomas
Solder battery package	Thomas
Design and print box. Test it in realistic envrionment	Thomas
Research front-end API and framework	Both
Login on front-end. Research database system for users	Thomas
Design login view	Valter
Setup remote database for users	Valter

Query historic data and prepare data for view on a map	Thomas
Find map provider and write API for displaying positions	Valter
Write server code for MQTT connection and live updates from Telenor Managed IoT Cloud and listen on all devices	Valter
Make front-end view for live or last update, toggle option for previous positions	Valter
Make GUI and back-end functionalities to edit properties with Telenor Cloud API. Simple GUI to edit update rate and save changes in a database	Valter
Implement fault-tolerance and deep sleep on Lopy, perform tests	Valter
Setup server instance at Heroku for hosting	Valter
Test features and system behvaiour(e.g. battery life, signal strengths)	Both
Measure power consumption in deep sleep and work mode	Thomas
Generate dummy sensor data with script, used for testing of server and front-end	Valter

The rest of the report is laid out as milestone chapters, starting from milestone 2. The milestones are laid out on a timeline that incrementally build a solution to the problem. Milestone 2 includes the project design, a high-level priority list of features to implement, technical design and outlining of risks, as well as how they will be mitigated. Milestone 3 describes the state of the project after implementing the core functionalities and the path to achieve it, as well as any observations made. Milestone 4 describes the remaining implementation choices, the path the project took from milestone 3 and end-state of the project. Finally, there will be an analysis of some of the aspects in the project. At last, we will draw a conclusion.

2 Milestone 2

2.1 Design decisions

As of 19th of February, all design decisions are clarified. To gather relevant data, we learn that two Pytrack expansion boards and two temperature sensor are suitable for our use because they provide all the features we need. The relevant components are one GPS transmitter, accelerometer sensor and a DS18B20 one-wire digital temperature sensor on each board. The GPS transmitter and the accelerometer sensor are embedded in the Pytrack. We need to focus on the power consumption because animals are in their natural habitat for months at a time, depending on the species. We would like to make the sensors as low-powered as possible because it takes time to replace the battery packages. Because of the LoRaWAN's bandwidth limitations and battery life limitation, we need to pay attention to the size of the data that are sent. We also need to focus on the design of the boxes. An important aspect is that they must not bother

the animals. In addition, it must be waterproof and able to withstand nudges and that they don't fall off the wearer. Finally, we need to securely present the data that are meaningful and intuitive to the end-user.

We prioritize the order in which the features are implemented. The features that must be included are:

- Gather GPS coordinates, accelerometer data, battery voltage and temperature readings on the senors
- Send the gathered data to a MIC Cloud instance provided by Telenor
- Send data from MIC Cloud to a front-end system
- Implement a front-end system with login sessions
- Show the GPS coordinates in a map, along with time stamps and temperature readings

These requirements are needed to make up a minimal standard for the project's result.

The features that should be included:

- Two-way communication with MQTT, being able to change state at lopy, e.g. update rate.
- Show info and previous positions for each device
- Upon no movement registered with gps, or lack of gps signal, automatically use accelerometer to verify that the animal is still alive and moving. Give notification in GUI of result.
- Notify if some individuals are in unwanted positions, such as close to village, roads or similar places.

The features that could be included are:

- Notify subscribed users on email or phone if interesting events happen
- Implement a smart phone application that interfaces with the system
- Comprehensive info about each animal.
- Show any grassing areas that are beneficial to weight conditions or other factors

The feature that we will not include is

- Direct communication between the observational units

2.2 Overall flow and states

So far we have a direction and goal for the project. We have a list for all the sensors we need, an architecture for the system and we also know how to design boxes that will be 3D-printed. In addition, we have setup a development environment and tested the LoPy device by sending some arbitrary data to MIC Cloud. We have also looked into popular development frameworks for implementing a back-end system for presenting the data.

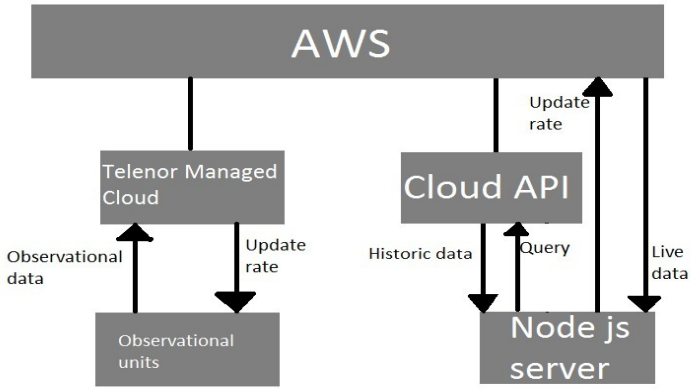
2.3 Estimated time for work

Taking the current state of the project into consideration, we believe we are on schedule. By the time of milestone 3, we aim to have implemented most of the features on the server and on the observational units. We think that between milestone 3 and milestone 4 we implement all the remaining features.

2.4 Technical design

We begin by describing the architecture of the system, the choices made and why they were made. The architecture is visualized in figure 3.

Figure 3: Architecture



This is a distributed system with observational units on the edge, a cloud instance and a server with a front-end system. The AWS box in the figure is both the message broker for Mqtt transactions and were thing properties and historic data is stored in buckets. The MIC cloud handles LoRa connection with the Lopy devices and publishes formated data on a specific topic to the Mqtt broker at AWS. The node js server subscribes on the topic for the observation data from the AWS broker. The observation data is stored in buckets at AWS and are reached through a cloud API provided by telenor. The cloud API is

also used for updating thing properties such as name and descriptions. The node js server also interact with a MongoDb database, although it is only used for login credentials and other small properties concerning our website(Not in the figure) .

There is an arbitrary number of homogeneous observational units on the edge, each of which gathers data. Their homogeneity comes from the fact that they perform the same tasks in the same order. However, they might do their tasks at different points in time. Each observational unit will gather temperature in Celsius, GPS coordinates, accelerometer data and battery voltage. They will regularly send the observational data to the Telenor MIC Cloud using the LoRaWAN protocol. The data is transmitted over LoRa network because it is important to have long range connections and low power consumption. It is also within the scope of the assignment to use LoRaWAN to connect with the Lopy devices.

We need a server with a front-end that displays the data because we can fine-tune the presentation of the data. It can also be more user-friendly than the provided MIC cloud GUI from Telenor. In addition, we would like to emulate a real-world user's point of view from this front-end system. To enforce security measurements on the front-end component, we want a login feature. We want to make it possible for the end-user to manage the sensors from the front-end. This feature simplifies the process of naming sensors and managing their properties.

We need to show the positions in a map because knowing the position of the devices is the essential of this project. For each data point in the time interval T_1 to T_2, the longitude coordinate, latitude coordinate, temperature and battery voltage are displayed. The time is given in the ISO date. The longitude and latitude coordinates are marked on a two-dimensional map as a single point. The temperature, accelerometer and battery are also displayed at given points in time. This is so the user can access vital information on the current state of each device, specially upon abnormal events. These abnormal events can be too low temperatures where an animal keeper may want to track down the individuals and possibly move the pack. Also, researchers can look at the relation between geographical areas in which individuals are located and the temperatures in the areas. The battery voltage is needed so that an observational unit is able to alert the end-user of low battery life.

The accelerometer data is used as a indicator along with the GPS data to see if a individual is likely to be deceased. If an individual has not changed its GPS position, it should be possible to see if the accelerometer data within a given time frame has changed. It may strengthen or weaken the suspicion of a deceased individual. If there is no GPS signal, the other observational data points are still sent. The accelerometer data can indicate whether or not an individual is still alive even though there is no GPS coverage.The time frame

has yet to be decided through parameter adjustments. Because it is possible that the IoT device sends data at the time an individual sleeps, the time frame must be adjusted. For example, if an observational unit transmits data once a night, the animal might be at rest at a position it favours. The animal is still alive, but the sensor readings tell that it has not moved. However, if the update rate is rapidly (e.g. once every tenth minute) and the GPS coordinates and the accelerometer data don't change over some period of time, it is likely that the animal is deceased. This should be recognized by the system itself, either server side or at the device, and then perform tests with increasing the update rate for reading sensor data. The state should then be presented with an alert level according to the adversity of the results.

A two-way communication is needed so that the end-user is able to adjust the rate at which a particular observational unit sends data. The update rate should be multi-casted; a broadcast is not needed because there might be situation where only one individual needs to be tracked down with higher update rate. However, a broadcast is possible by toggling how many observational units that are to be updated. A situation where two-way communication is needed is when a wolf is in its natural habitat the update rate can be as low as one transmit per day. When the same wolf is to be tracked down for research purposes, the end-user can increase the update rate to several times per minute. This feature enables the end-user to prolong the battery life to the fullest extent. This feature makes the system more agile with respect to battery life. A two-way communication is also needed when checking a suspicion of a deceased animal server side, where the server might have to adjust the interval of the device emit rate.

3 Milestone 3

As of 19th of March, we have implemented most of the core functionalities on the edge and the front-end. We have made progress on all components of the system. The equipments that we use are two Pytrack expansion boards and two DS18B20 one-wire temperature sensors. From the technical specifications drafted in milestone 2, we have decided for most of the low-level details. An important point to make is that observational data is transferred from the edge to the front-end system. We have made several discoveries since previous milestone that need to be addressed.

3.1 Sensor side

We have managed to gather GPS data, battery voltage and accelerometer data from the Lopy device. We implement the Pytrack boards using MicroPython. We successfully send the gathered data with LoRaWAN to a Managed IoT Cloud instance. We use a 3.7V battery 600 mAh battery to power the sensor.

When we deploy the sensor in a real-world environment in lower temperatures, we will use a bigger battery.

We tested GPS data with carrying the device around on our self and checked in the MIC cloud GUI that they were correct. We also had them with us in our cars when driving about. What we did notice was that that we lacked many points on the route we have taken and we suspected a lack of GPS signal. When checking the historic observational data stored in AWS buckets, it was revealed that we were not having proper LoRa connection every where we went and therefore lacked the positions. This is reasonable as the LoRa network in the area is only at a test level with some antennas, meaning its coverage will be limited to certain areas. We did have observational data that were lacking GPS coordinates, but they were mostly from testing the device inside a building. It is reasonable to think that there are areas in the habitat of the animals that may also lack GPS signals, e.g. close to buildings or cliffs. This problem is hard to cope with as it is decided of factors we don't influence, therefore it seems that the GPS suits it purpose as far as our testing goes. The device is set to wait for GPS signals for 20 seconds. If there is a chance of signals, this amount of wait time appeared to be decent during testing. Waiting any longer implies that the signals are very poor and the device might need to be moved somewhere else to retrieve signals. As the device runs in a loop when waiting on signals it uses much power and it is a trade-of between power consumption and the probability of getting GPS signals. We think it is better they for the device to go to sleep and try again next time it awakes when there is a chance the animal has moved to a more reachable location.

We choose still to send temperature readings, accelerometer data and battery voltage if there is no GPS coverage because the data being sent are still useful. For example, if a sheep wears the observational unit and the end-user sees that the temperature drops to very low temperatures, the end-user can still take actions. The limitation, however, is the fact that the end-user is not aware of the sheep's location. The battery voltage reading still serves its purpose of telling how long the observational unit has to live. The accelerometer data tells if there is any movement and therefore reveal if the animal is likely to be alive or not.

When testing the accelerometer, we observe that the accelerometer values fluctuate in all three dimensions by about +/- 0.001g, even when the accelerometer is standing completely still. We think we can introduce an interval between which the values can fluctuate so that the system can ignore negligible fluctuations. Since we are interested in any movement regardless of its direction, the values for movement in x, y and z direction is summed together before they are sent. The summation of the data is reduced to three significant figures, as a more accurate precision is useless because of the fluctuation in the values. This also reduces the data size that has to be emitted. The system guarding these values still has to consider some fluctuation when checking the movement values. From testing we estimated that consecutive values inside a interval of

+/-0.02g can be considered as no movement, but this should be tested a to a further extent in a more realistic environment before it is confirmed.

The format of the data that we send is <longitude, latitude, battery voltage, temperature, accelerometer>. As of now, we only generate data for temperature since the actual sensor is still to be mounted on the Lopy device. The size of the payload we send is either 24 bytes or 34 bytes.The size depends on whether or not there is GPS coverage in the area. The values in the payload are separated by whitespaces. If there isn't GPS coverage, we choose to send empty default values for longitude and latitude coordinates.

The sleep time is a value meant to be set as it suits the conditions and environment the device is operating in. We have mentioned an update-rate meaning how often the device emits data, which in practice is how long the device sleeps between each measuring and transmit. We have yet to implement deep-sleep on the device.

3.2 Telenor MIC Cloud

We read observational data on Telenor MIC Cloud by specifying the format of the observational data received from the Lopy. We successfully visualize the received data with the MIC cloud GUI widgets. See Appendix figure 2. This is very useful when testing the devices on connection and content of the data. Sending the data back to the observational units is still an open issue.

3.3 Front-end

We have implemented a website with html, css and javascript for front-end and using the Node JS server environment for back-end. We are somewhat familiar with Node JS and we know it has suitable packages since it is used in all the tutorials about MQTT connection and cloud API. To display map we use the open source library Leaflet which provide a user friendly interface to display maps. We use maps from NorgesKart because it provides a detailed and much descriptive map of Norway at no cost. It is especially useful for displaying points, given longitude and latitude coordinates, which is what we need. We are able to subscribe on topics at the Mwtt broker at AWS and receives the published observational data from the Lopys. We also do queries for older observation data with use of the cloud api and display things positions on the map.

We made feature for editing things in the Managed IoT Cloud with use of the Cloud API. This feature provides possibility to edit the thing name and description in the frontend website. This feature is implemented since it is likely that it is more convenient to have other information on a animal then just an ID nummer. For example you could note number of lambs for a sheep, weights or other info about a individ that could be useful to know. This info should then

be connected with thigns shown on the map in an intuitive way. As of now they can be viewed in the editing table.

We choose to protect the front-end with password authentication because we believe that security is an important aspect of this project. Although the project is not planned to be shipped into production, it is worth applying best practices. After authentication, the user manages the front-end as it sees fit.

To test the front-end and its behavior with more data and units, we wrote a script for dummy units publishing generated observational data to the Mqtt broker.

4 Milestone 4

As of 18th of April we have made progress on several aspects of the project. We have refined some of our approaches on the features since milestone 3 and implemented some new features as well.

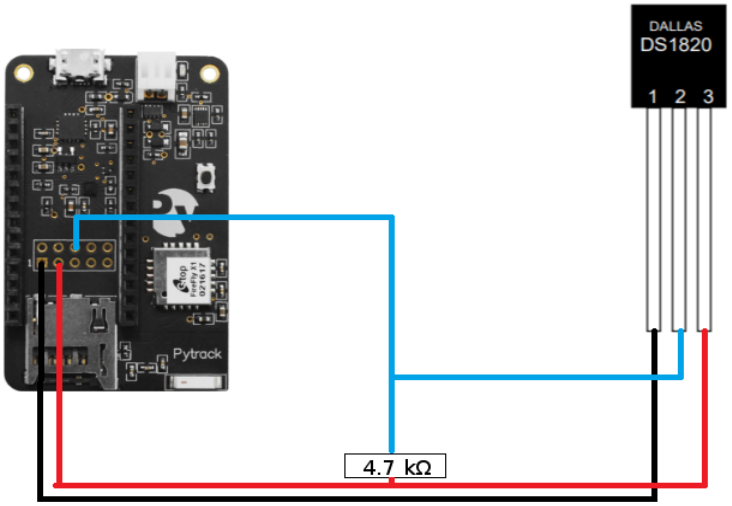
4.1 Sensor side

There is now implemented deep-sleep at the Lopy where the sleep time is set static in the code. The deep-sleep is initiated when the all the sensor data is read and sent further over the LoRa connection. When the device awakes from the sleep it should fetch and send sensor data, before it falls into deep-sleep again. Therefore the sleep time will decide the interval on the rate that observational data is transmitted and is the value that should be set when changing the update rate. The decision to use deep-sleep is made upon the interest of reducing power consumption by reducing the processing time for the unit. There occurred some challenges since the device reboots after each time it awakes meaning that the previous state of the device is forgotten. This includes the already established connection with LoRa, which made the device use a significant time on reestablishing the connection on each awakening. To reduce the processing time, the established connection is saved to a none-volatile memory on the unit where it is read from upon a awakening. To check if a connection is stored, a validation number is also saved each time an connection successfully is established. The decision of saving the connection was made since it is in our interest to reduce the processing time and power consumption. The effects on the battery lifetime with implementing deep-sleep, is left out for a own section.

We will be soldering a DS18B20 temperature sensor on one of our Pytrack boards. The basis for soldering the temperature are figure 1 and figure 2. In particular, the external IO header connectors are relevant. Initially, we connect the temperature sensor to the Pytrack via a breadboard to see the data being read. We begin by temporarily attach 10 female sockets on the Pytrack board

during the experiment because it makes testing the temperature sensor more practical. We also used three male cables between the breadboard and the Pytrack. The diagram for connecting the temperature sensor to the Pytrack is displayed in figure 3. Connecting the temperature sensor to the breadboard is displayed in figure 4. Notice the use of a 4.7 kΩ resistor.

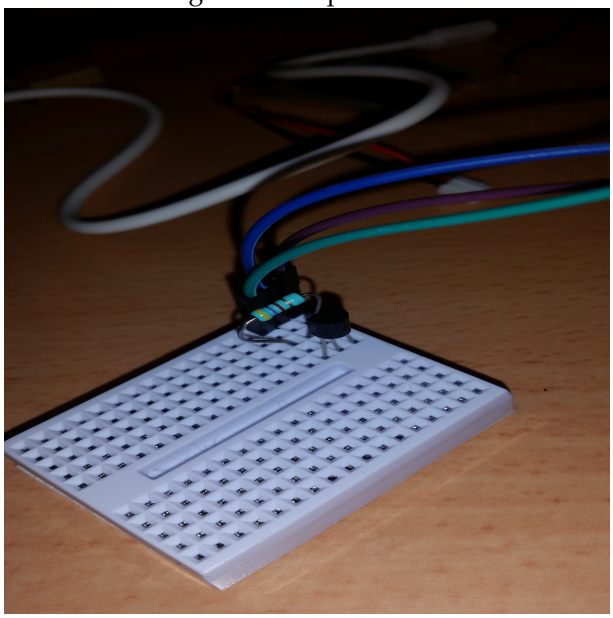
Figure 4: Temperatue sensor connection data sheet



This setup proves to be successful. We observe temperature readings from the sensor at Telenor Cloud. Prior to successfully setting up the wiring, Thomas misread the datasheet once, which resulted in a temperature sensor getting destroyed. Now that we know the sensor works and the temperature readings seem to make sense, we need to attach the temperature sensor to cables. We use two stands to keep the components still while soldering. However, we realize that the Pytrack board is too thick with the cable pointing outwards from the microcontroller board. It is important to make the box as small as possible, which depends on the micro controller that is to be put inside.

We remove the connectors underneath the board and instead use connectors that are bent 90 degrees. In addition, the cables can be hidden underneath the LoPy component, decreasing the thickness even more. We solder the bent connectors to the Pytrack board. The result is displayed in figure 5. Notice the black LoRa antenna to the right and the three temperature sensor cables to the left coming from down under the LoPy component.

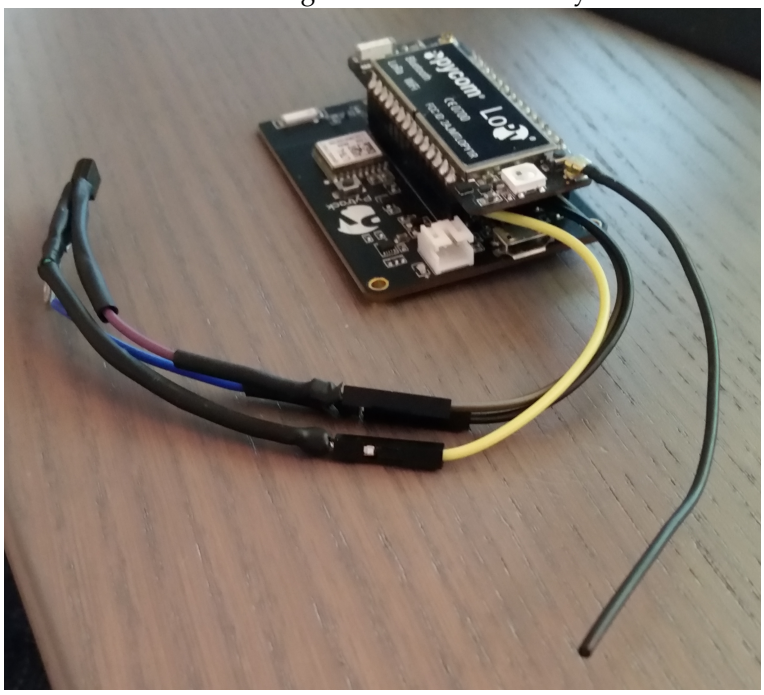
Figure 5: Temperatue sensor connection data sheet



The design of the box is based on being as small and lightweight as possible. Because the box is supposed to hang around an animal's neck, the weight is in focus. It must be fairly easy to replace the battery package as well, without the risk of damaging anything. It should be waterproof as well. We use Tinkercad for designing the box. The final design is displayed in Appendix figure 1. The box is split into two by a plate that can be slided in and out of it. The battery package is on one side and the Pytrack board and temperature sensor is on the other side. The battery and Pytrack are strapped to the plate with hook-and-loop fasteners. There is a hole in the plate for the battery cable. The lid has two edges for fastening the plate in it. The box was 3D printed based on the design generated in Tinkercad. In reality, we had to use the SolidWorks editor to print because we had some issues with adjusting the printer settings. All in all, the design was migrated from Tinkercad to SolidWorks.

The lid is attached to the rest of the box by three screws in each hole. In addition, we fill the crack in the lid with expanding foam to make the box waterproof. Furthermore, we attach the entire box around a dog's neck to test it. We choose to use a german shepherd to test the box because of its size. For a realistic test scenario, it needs to be shaken in all directions and it needs to be pressed against the ground. The picture shows the dog wearing the tracker: BILDE AV LOKE

Figure 6: Final result of Pytrack



4.2 Analysis of battery consumption

On the edge, the battery consumption is in great focus. We send 34 bytes over LoRaWAN network, which is a relatively small amount of bytes, considering the fact that the LoRaWAN bandwidth is at from 52 -200 bytes depending on the connection strength. Taking into account that the Pytracks are in two states, namely “work” and “sleep”, we observe two different power consumption values. We use a lithium battery with 600 mAh at 3.7V for testing. Measurements conducted when the Pytrack is in “work” state, the power consumption equals 64.4 mAh. During the “sleep” state, the consumption equals 18.2 μAh. This means that having longer inactivity periods in deep-sleep will make the battery last longer.

From the measurement above the 600mAh battery is estimated to last 6.5 to 7 hours depending on the environment temperature. We conducted a simple test where we compared how long a battery lasted with and without use of deep-sleep. When running without deep-sleep, transmitting data every minute, the battery lasted 7 hours. When implementing a deep sleep of one minute between each "work" state, the battery lasted 33 hours. The life time was then extended almost 5 times. Although 33 hours is clearly not enough as, animal such as sheep stay out of bounds for several months each summer.

We believe that it is possible to decrease the amount of data being sent. If there is not GPS coverage, "None None" is sent. We can replace that pattern with for example "N N". Although the transmission would decrease by only 6 bytes, it is still worth pointing out this alternative.

We also want to estimate battery life of an observational unit. In this case, we use two Sony 18650 battery coupled together in parallel, delivering 3000 mAh each. Together they deliver 6000 mAh. However, they are a few years old, so the output should be somewhere below 6000 mAh.

4.3 Front-end and server

We are now able to listen on all devices at once with use of the cognito mode for subscribing on topics at Mqtt broker at AWS. We have up and running a map presenting all live things that are able to send their position and if no observational data is received, their previously known location is shown instead. Live and previous positions are separated with different markers to be more intuitive. It is possible to see when the data was registered, so one is able to evaluate freshness and relevance of a position and the observation data. There is a toggle option to hide and show things positions, since having all up at once can get complex with many individuals. There is also an option to view the path of previous positions of each thing. This feature can be viewed in Appendix figure 3. The data fetched for path is fetched with elastic search and cloud API provided by Telenor.

As of know we haven't reached to implement all the features we wanted to have at our server and front-end. Among these is the a system for checking accelerometer values upon suspicion of a deceased animal and setting the device update-rate from our server, which would be the sleep time. We did look into sending data and gave it an attempt, but were unable to succeed within the time of the delivery. The idea is that we can publish a desired state to a things topic at the Mqtt broker, which the MIC cloud is subscribing too. The MIC cloud should then transfer the data down to thing device upon receiving data from it, as it can only receive data when it is awake and just after it has sent its data. We managed to send a desired state to the MIC cloud, but failed to receive it at the thing device. Further debugging and testing is required to finish this feature, but we suspect the failure has something to do with the formating of data and handling of socket connections at the device.

We have deployed the server and website with Heroku web-app service, this is so it would be simpler demonstrate our product any where you have access to the web.

5 Conclusion

In this project we ventured on a task to track animals in the nature using LoRaWAN network and IoT devices. We have gained valuable insights into low-powered edge computing with limited power supply. We saw how edge computing is realized using IoT devices in a meaningful context. Looking at the problem in question, we did solve it to a reasonable extent. Although some feature are missing, we believe we proved that the most important parts of the project were completed. At the very core, we draw a relation between the initial goal to transfer observational data over LoRaWAN and the fact that we did it.

5.1 Ending state

As one of the features we wanted to implement, but did not achieve is the already mentioned two-way communication. This opportunity for communication were to be used for setting the update-rate on the device, and there by also influence the power-consumption. As battery life time is, as mentioned, an important aspect of our product, this feature about setting update-rate is one of the more vital features we didn't achieve to implement.

The website and server have implemented the features we wanted to some extent, but suffers from some lacklusters and anomalies when handling edge cases. We have not reached to implement warnings upon events such as animal on the road or low battery. Although it is not ready to be sold as a product, it works well as a draft on how a website and server could turn out for our system.

5.2 Future work

For a real-world application, the box's mass should be decreased. It's somewhat heavy, even for a 50 kg fully-grown German shepherd. As of now, we are able to track animals of similar size, but not smaller-sized animals. There should be conducted more testing on battery life time with adjusting factors such as sleep time. This is to find out with more certainty how often the device may transmit its observational data for a battery to last several months or maybe a year. The website would require more work on how to present data intuitively and perhaps be supplemented with an app for mobile devices.

An interesting aspect is any future work in the project. One of the COAT's collaboratives is interested in how positions, activity and temperature are collected. Those three data points can be correlated to each other to see the relation between changes. For example, in one season the temperature is at a degrees Celsius in a particular area. Furthermore, there are x individuals in this area. The next year, there can be a lower temperature in the same area. At the same time, it is counted less than x individuals of the same species. By

looking at correlation, one can predict migration patterns, given temperature fluctuations. In fact, Thomas has been in dialog with one of the COAT collaborators about a potential Master's Thesis that builds upon some aspects of this project.

5.3 Relevant work

Rovdata uses similar tracking devices to track wolf, lynx, brown bear, wolverine and golden eagle. However, they seem to not measure temperature. Telespor is a local company based in Tromsø that provides similar services such as we do. However, they use 2G network and don't measure temperature either.

As our device is used for tracking, it is reasonable to think it can suit other tracking purposes as well and not only animals. An example is to use it on fishing equipment if you have problems with thefts and want to see were your items have gone. As we have shown, the little battery of 600mah lasted for 33 hours when updating each minute, which should suit such a purpose well.

References

[1] *DS18B20 data sheet*
`https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf`

[2] *LoRaWAN overview*
`https://www.thethingsnetwork.org/docs/lorawan/, 2018`

[3] *MQTT overview*
`https://en.wikipedia.org/wiki/MQTT, 2018`

[4] *Telenor Managed IoT Cloud*
`https://startiot.telenor.com/lessons/mqtt-and-managed-iot-cloud/, 2018`

Appendices

Figure .1: The box with a Pytrack and battery package strapped onto it

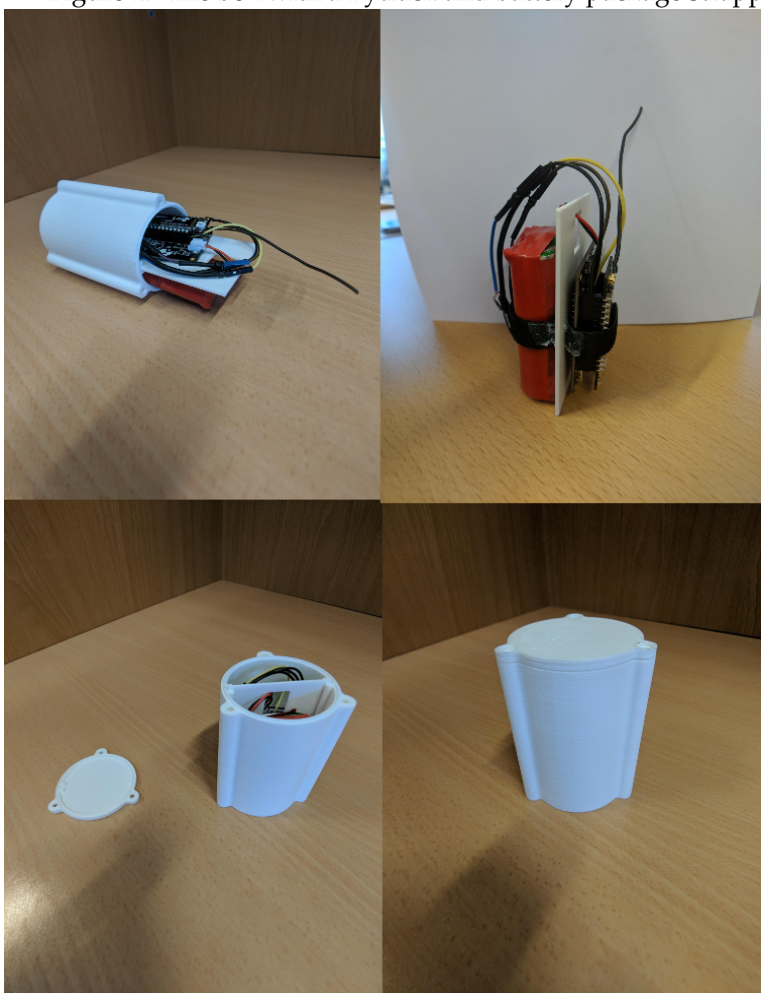


Figure .2: View on how the widgets in MIC displayed observational data.

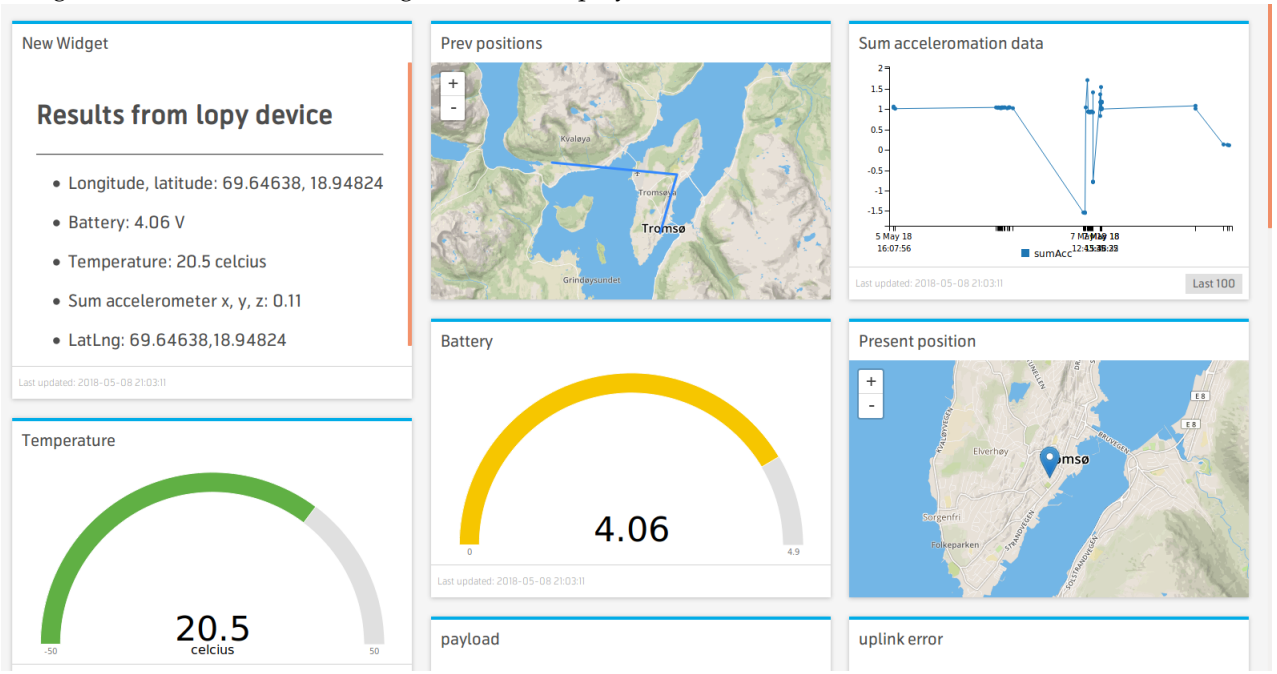


Figure .3: View of website showing history path and latest position of one device.