Linnæus University
Sweden

Report

Motion detection with IoT

Camera surveillance with motion detection using LoRaWAN.

Authors: Lars Petter Ulvatne, lu222bg
Findlay Forsblom, ff222ey
Supervisor: Francesco Flammini
Semester: VT 2020
Course code: 2DT301

Abstract

Devices connected to the Internet are a part of today's society, along with notifications from the applications that are important to us. Entities outside our standard WiFi-range poses a problem for this task and sets the need for another connection setup. Also, camera surveillance and monitoring are implemented at an increasing rate. Those two problems constitute the core of this project.

A camera has the limitation of only detecting motion in the line of sight. This project initiates a solution for a situation of monitoring distant targets with a single camera when motion is detected. LoRa (Long Range) can be used to send data outside WiFi-range and was in this project used to notify the camera server and user that motion was detected.

The final output of this project is a prototype application which, when motion detected, will trigger the camera server and notify the user. The notification includes sending a picture and poses the possibility of setting up a live video stream.

Table of contents

1. Introduction

1.1 Project aim

The main goal of this project was to connect distant devices to the internet and build the full chain, from detecting sensor values to make the user interact with devices based on knowledge of those values. From this core idea a practical implementation of a camera monitoring system emerged, triggered by motion detection at a LoRaWAN device, originated outside WiFi-range.

The user should, from notification, act upon a picture, triggered by the data sent from the LoRa-node to decide if a live video stream should start or not. This report will cover the implementation of long-range communication from a LoRa-device to trigger a camera monitoring system. This system should then evaluated if it could be implemented with multiple motion detectors and a movable camera, to cover a full area.

1.2 Report

The technical information about the project hardware and software will be addressed in the *Background* section, along with a technical motivation why this project was needed. As for implementation methods and project partitions, the *Analytical process* part has to be visited, which also includes the research aims and the results of those aims.

These sections will be followed by a *discussion* about the findings in the project along with a *concluding summary* and *extension proposal* for the interested reader.

1.3 Project parameters and boundaries

1.3.1 Parameters

To gain knowledge how IoT-devices are connected outside local network areas, LoRaWAN was chosen as communication platform. A part of the aim was to build a full chain application, with user interaction through a web application. The interaction should be based upon the data received from LoRa-device and make use of a camera to monitor the environment.

1.3.2 Boundaries

Since only one LoRa-device was available for this project, this had to be used as node and additional nodes were not possible to add. An existing gateway in the area was needed for connection, as an own implementation of a gateway was not possible due to the lack of devices.

A primary idea was to add the camera monitoring at the position of the LoRa-node. Due to the low data rate [1], image/video transmissions were found not feasible.

Limitations of project complexity, available resources and time constraints enforced the use of only one LoRa-node along with a simple camera module. A deeper implementation would be to use multiple motion sensors, which would make the monitoring possible for 360 degrees using a rotating camera with zooming possibilities.

2. Background

Camera surveillance or monitoring for movement on large properties could pose many problems. Multiple cameras would most likely be needed and has the need of a greater power supply source. By using low powered LoRa-devices as motion sensor sources, they could be positioned on the outer limit of an area with batteries as power supply. The idea of this project was to investigate how these devices can interact with a single camera to monitor a specific area, see *figure 1*. The final prototype will only contain one LoRa-node and a static camera, while the extended project will be evaluated by the knowledge of the prototype.

As the "Internet of Things" (IoT) grows rapidly it is important to be able to check your resources through a web application. Along with the progress of IoT, an important issue of security has to be addressed. For an application to be secure there are different security matters to take into act. Authentication and attack management are the core parts implemented in this project along with safe data transfer over HTTPS.

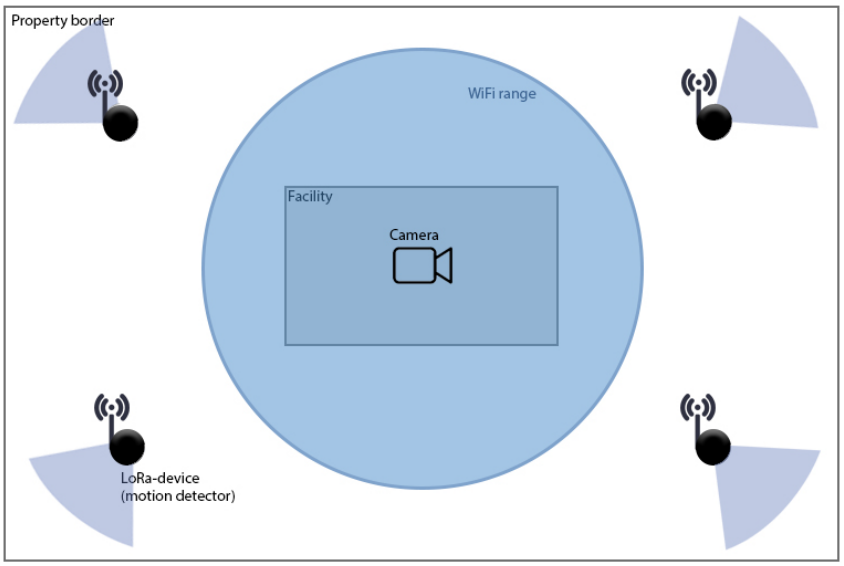

Figure 1. Explanatory picture for the extended project.

2.1 Hardware Components

The components listed below are the hardware used in this project.

- Raspberry Pi 3 model B
- Raspberry Pi Camera Module v2
- Pycom LoPy4
- Pycom Expansion Board 3.1
- Pycom LoRa/Sigfox Antenna
- HC-SR04 Ultrasonic sensor
- MCP9700A Temperature sensor

2.2 LoRaWAN

Due to the usage of efficient modulations, much alike FSK (Frequency Shift Keying), LoRa gains low power consumptions along with a long-range data transfer capability. A single LoRa gateway can cover hundreds of square kilometers and large areas of cities. This along with the low cost, makes LoRaWAN communication excellent for long range data transfer with IoT-devices.

Europe uses the frequency range 867-869 MHz (433 MHz also available, but rarely used), with data rates between 0.25-5.5 kbps on normal channels. The low data rate restricts the amount of data to be sent, which makes LoRa good for sending sensor data [2].

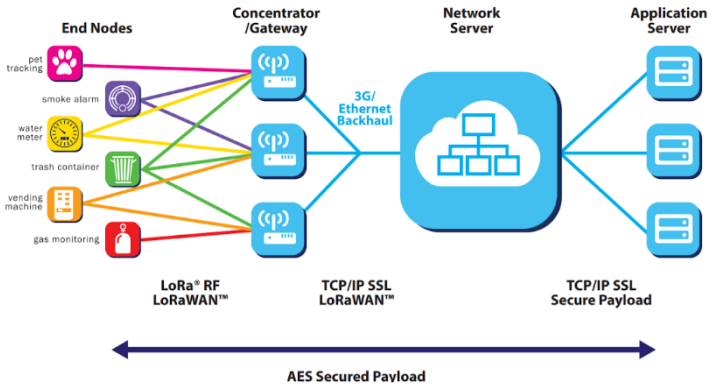

Figure 2. Overview of how LoRaWAN is implemented [2].

2.2.1 TTN - The Things Network

The Things Network (hereinafter TTN) is a global open LoRaWAN network. They provide tools to build IoT application at a low cost, with built in security with end-to-end encryption using LoRa [3]. For users with existing accounts, gateways and applications with possibility to add LoRa nodes can be setup through the console.

TTN uses two different connection methods, OTAA and ABP:

2.2.1.1 OTAA – Over-The-Air-Activation:

The preferred and most secure to connection method to TTN is OTAA. The node and gateway perform a join procedure and negotiations about security keys are made. This method uses a dynamic device address upon connection [3].

2.2.1.2 ABP – Activation by Personalization:

When using ABP the device address and security keys has to be injected into the code. This can be viewed as personalizing each device and can sometimes be more effective, since the join procedure is not made. The cost of this comes with some security downsides [3].

2.2.2 LoRa Gateway

Gateways are the middleware between the LoRa-node and TTN. A node can connect to a gateway using LoRa, while the gateway will forward the packages using a high bandwidth network (WiFi, ethernet, etc.) to TTN.

The gateways can either run on minimal firmware (only packet forwarding) or with an OS installed to give administrators management opportunities [4].

2.2.3 LoRa-node and connected sensors

The node module sends data over LoRa to a connected gateway, which then forwards it to TTN, where the application server can access the data. The following components were used at the node endpoint:

Pycom LoPy4:

The LoPy4 is a development board, enabled with MicroPython. It can use four different network strategies (WiFi, Bluetooth, Sigfox and LoRa). The Semtech LoRa transceiver enables it for connectivity ranges up to 40 km as a node and 22 km as a gateway. It has eight 12-bit ADC analog channels and up to 24 GPIO pins. The very low voltage, compared to other microcontrollers, makes it a good IoT-device when using battery power [5].

Figure 3. Pycom LoPy 4 module.

Pycom Expansion Board 3.1:

The expansion board is used to connect the LoPy4 to a computer using the micro-USB. It also contains outlets for jumper wires to connect sensors to the LoPy. Other peripherals are battery power outlet, MicroSD card, etc. [6].

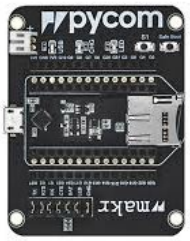

Figure 4. Pycom Expansion Board 3.1

Pycom LoRa/Sigfox Antenna:

External antenna for 868/915 MHz frequency band. Easily mounted onto the LoPy controller to enable LoRa connectivity [7].

Figure 5. Pycom Lora Antenna.

HC-SR04 Ultrasonic Sensor:

For this project the HC-SR04 ultrasonic sensor served as a motion sensor. As the distance to target decreased below a threshold value, motion has been detected.

The sensor measures distance from 2-400 cm at an angle of 15°, with an accuracy of +/- 3 mm. It has an ultrasonic transmitter and receiver, which takes use of ultrasonic sound waves (> 20kHz) [8].

Figure 6. HC-SR04 Ultrasonic sensor.

MCP9700A Temperature Sensor

To find the surrounding temperature with this sensor, the output voltage is measured (mV) and calculations made to retain the environmental temperature.

Figure 7. MCP9700A temperature sensor.

The following PIN configurations were used by the sensors:

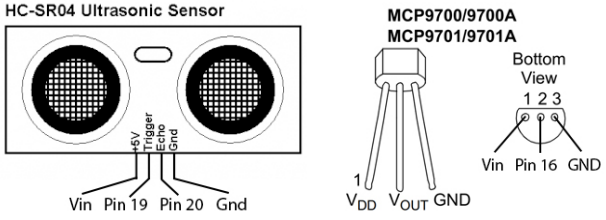

Figure 8. Pin configurations for both sensors used at LoRa-device.

2.3 Raspberry Pi - Video Streaming Server

2.3.1 Raspberry Pi 3 Model B+

Raspberry Pi is a set of single-board computers. The Raspberry Pi 3 Model B+ has a quad core processor with capability of 1.4 GHz along with 1 GB SDRAM. Other specifications for this model are 40 GPIO pins, 4 USB ports, internet connection through WiFi/Ethernet, MicroSD socket, etc [9].

Figure 9. Raspberry Pi 3 Model B+.

2.3.2 Raspberry Pi Camera Module v2

The Raspberry Pi camera module v2 fits to the CSI camera port on the Raspberry Pi, making it very easy to connect. A Sony 8-megapixel sensor is used support high-definition videos in 1080p30, 720p60 and VGA90 modes along with still capture. It is said to be very popular in home-security systems and wild-life camera traps [10].

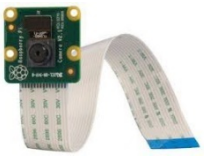

Figure 10. Raspberry Pi Camera Module V2.

2.3.3 Softwares

The list below contains the most important libraries used on the Raspberry Pi, written in Python3:

- ***picamera**:* Accessing the Pi Camera Module v2.
- ***http.server**:* Sets up HTTP server.
- ***socketserver**:* Enables multithreaded server by using sockets.
- ***io.BytesIO**:* Write bytes into buffer.
- ***ssl**:* Set up HTTPS through SSL/TLS certificate.
- ***threading.Condition**:* Used to notify all threads when buffer is ready.
- ***PIL.Image**:* Saves a buffered image to a file on disk.

2.4 Application Server

The application server was built on an Ubuntu server, running on Nodejs and sits behind a NGINX reverse proxy server.

2.4.1 Node Application Modules (Backend Modules)

The web application for user interaction was run in Node. Modules (or packages) are libraries used to solve tasks at hand and are installed using the Node Package Manager (npm) along with the information gathering. Down below are a list of the most important modules along with their purpose [11]:

- ***Express**:* Express is a web application framework for Node, used for building web applications and APIs. It has been called the de facto standard server framework for Node.
- ***Express-hbs**:* Express handlebars is a template engine with multiple layouts, blocks and cached partials coupled with Express.
- ***Express-session**:* Module used for creating and managing sessions.

- ***Cookie-parser**:* Required by express-session to be able to create a session. Makes it possible to add information on a cookie and sign a cookie.
- ***Mongoose**:* Mongoose is a MongoDB object modelling tool. Used to create Schemas for storing data to MongoDB.
- ***Socket.io**:* Library that enables real time, bidirectional and event-based communication between server and browser. Used to alert the client in real time when motion had been detected.
- ***Bcrypt**:* Bcrypt is a security module that hashes passwords and adds salt to the hashed password for increased security.
- ***Helmet**:* Helmet is a security module that enforces content security policy (CSP), XSS filter and other security features. In the application helmet was used to set a content security policy which states what resources are trusted.
- ***Dotenv**:* Dotenv is a module that loads environment variables from a .env file to a process.env variable. Used to hinder sensitive information to be uploaded to GitHub.
- ***Csurf**:* To help prevent against CSRF attack by adding a token to each form.

2.4.2 Frontend frameworks / libraries

Down below are the list of frameworks / libraries used on the client as front-end libraries, to enhance style:

- ***Bootstrap:*** Bootstrap is a free, open-source CSS framework directed to make responsive, mobile first front-end web development. In the application Bootstrap was used for designing the most part of the front-end stylings.
- ***jQuery:*** jQuery is a JavaScript library that simplifies HTML, DOM and CSS modifications.

2.4.3 Other Software

- ***NGINX**:* Nginx was used as a reverse proxy. A reverse proxy is a type of proxy server that sits behind the firewall in a private network. It directs requests to the appropriate backend server endpoint, in this project being the node application [12].
- ***PM2**:* PM2 was the process manager used at the ubuntu server. A process manager helps to keep an application or process running forever and restart the application or process on failure and reloads without downtime. PM2 was used to monitor both the node application and the Redis database to simplify administration.
- ***Redis**:* Redis is an open source in-memory data structure store, used as database cache and message broker. In the application Redis was used to store all sessions.
- ***MongoDB**:* MongoDB is a general purpose, document-based distributed database. In the application MongoDB was used for storing users and previous events.
- ***SSL/TLS certificate**:* By running Certbot [13] scripts a Let's Encrypt SSL/TLS certificate was installed on the server to enable HTTPS.

3. Analytical process

3.1 Research

3.1.1 Research questions

- How can implementing motion detection, using a sensor connected to LoRaWAN outside WiFi range, serve as a trigger point for camera surveillance at a specific area?
- How can implementing multiple sensors minimize the number of camera sources for property surveillance?

3.1.2 Research aims

- Build a sensor detecting system for motion, with LoRa as communication platform with connection to a user interactive application which can trigger camera surveillance.
- Evaluate the method for extension proposals, when multiple sensors are implemented.

3.2 Method

3.2.1 Project partitions

The project work was divided into following parts between the group members:

Findlay Forsblom:

- Application server.
  - Setting up the ubuntu server on openstack and learning the openstack infrastructure
  - updating and downloading the necessary packages in the ubuntu server such as node and npm
  - Setting up the reverse proxy on the ubuntu server
  - Configuring the reverse proxy and getting a tls/ssl certificate for Https
  - Did most of the backend and frontend coding and database setup and connection in the node application with some modifications by Lars Petter
- Raspberry Pi:
  - Setting up OS and environmental settings.
  - Research and implementation of camera server along with stream.
  - Camera setup (Frame rate, connection ports etc.)

Lars Petter Ulvatne:

- LoRa-device with sensor setup.
- Raspberry Pi:

  - Modifying server script and adding functions to secure video stream and image saving strategy.
  - Port forwarding at router.
  - Installing and setting up HTTPS connection.
- Setting up connection to TTN on application server for down/uplink messages to LoRa-node.
- Writing TTN script to parse data.

3.2.1 Learning OpenStack

To set up the Ubuntu server, a few steps had to be made. One of them included learning OpenStack and its infrastructure.

At first, the network topology had to be created. A router was created along with the local network for the server. To crate the local network, a DNS provider also had to be instantiated. The DNS server owned by Linnaeus University was used, which sits at the IP's 194.47.199.4 and 194.47.110.97. The next step was to create a server and connect it to the local network and assign a floating IP, which makes it visible to the internet. SSH-keys were created to access the server remotely. The topology created in OpenStack can be viewed in *figure 11*.

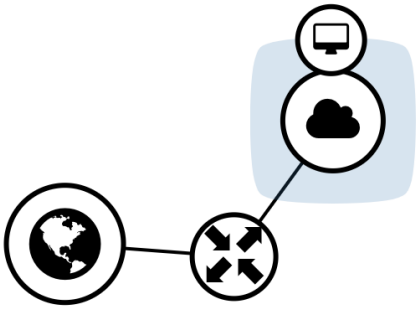

Figure 11. The server network topology created in OpenStack.

3.2.2 Reverse proxy

As mentioned in section *2.4.3*, the NGINX reverse proxy is a virtual server running on the Ubuntu server. It forwards all requests to the Node application locally. Since security was a vital part of this project, the NGINX had to force clients to use HTTPS. This required installing an SSL/TLS certificate, which was fetched by the CA named Let's Encrypt. The code

[*Appendix 1*], shows how the server incorporates HTTPS for incoming requests. *Figure 12* shows that the certificate was successfully installed.

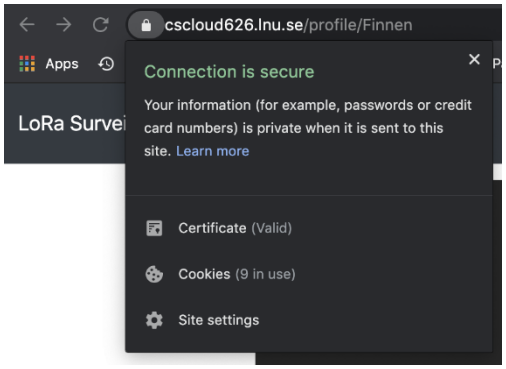

Figure 12. Successful installation of valid certificate.

3.2.3 The Node application

As mentioned in section *2.4.1* the server was implemented with Node and Express as backend framework. Handlebars, as all other modules, was installed using npm and serves as the view engine to serve pages for rendering.

As authentication and sessions were needed, Express-sessions were used as module for this task. Since the session handler uses cookies on the server side, it makes it harder to manipulate the contents of a cookie.

There are two way of managing sessions, save session variables in memory or in a database. To store in memory does not scale very well and is mostly used in development. In production, database storage is preferred and also used in this project with Redis. Redis was installed on the Ubuntu server and runs in the background along with the Node application.

3.2.3.1 Simple authentication

The authentication method was implemented by using an email address as username and a password, which was stored at the MongoDB database. Only authenticated users were allowed to view a page including content from the video server. As mongoose was used as object modelling tool, user schemas, see *figure 13*, were created and included a hashed password along with the unique username. The password was hashed using bcrypt, which adds a salt to the password before hashing. *Figure 14* shows how this was implemented in code.

```
/**
 * Scheme for users that is used by the Database
 */
const mongoose = require('mongoose')
const bcrypt = require('bcryptjs')
const uniqueValidator = require('mongoose-unique-validator')

const userSchema = mongoose.Schema({
  username: { type: String, required: true, unique: true, minlength: 3 },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true, minlength: 8 }
})
```

Figure 13. User Schema code.

```
userSchema.pre('save', async function (next) {
  const user = this

  if (user.isModified('password') || user.isNew) {
    const hash = await bcrypt.hash(user.password, 12)
    user.password = hash
  }
  next()
})
```

Figure 14. Hashing of user password.

When visiting the application without authentication, the user is shown the login page. Upon login failure or success, a flash message appears informing which action was taken. Upon failure, a red flash message is shown. Due to security reasons the wrong input parameter (username or password) will not be seen. The user is redirected to the profile page upon successful authentication, along with a green success flash message.

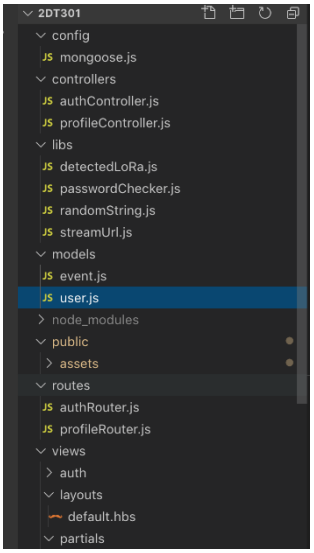

Figure 15. Application folder structure.

3.2.3.2 Code and Folder Structure

A distinct and clear folder structure are vital for this type of applications. As the project can be expanded, a good folder structure will ease the upgrade. The folder structure, *figure 15*, was built upon the application flow shown in *figure 16*.

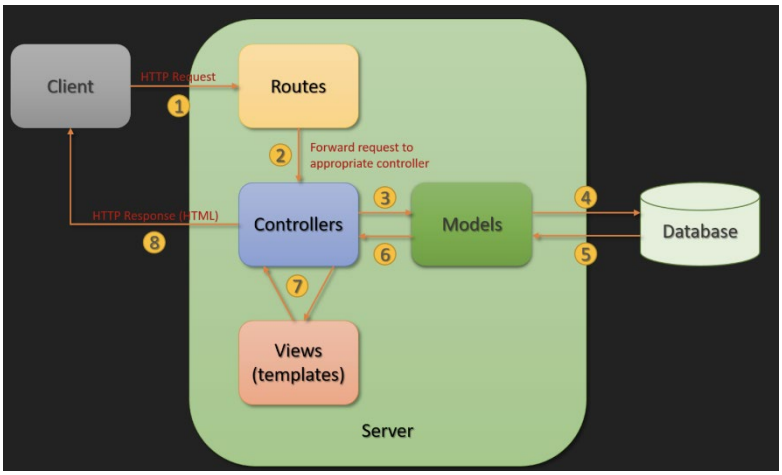

Figure 16. Application flow [14].

3.2.3.3 Alerting the client

As the LoRa-device sensor detects motion, an authenticated user should be alerted. A socket connection was implemented between the server and client to enable real time communication upon receiving an event from The Things Network. *Figure 17* shows how the client was alerted.

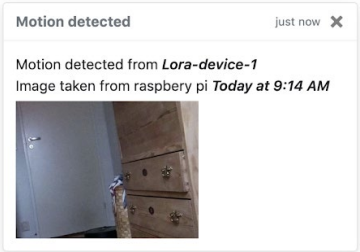

Figure 17. Alert message shown in browser.

To make the application more realistic, the client would be doing something else than monitoring the application at all time. This project takes use of the Notification API which sends a notification to the computer, see *figure 18*.

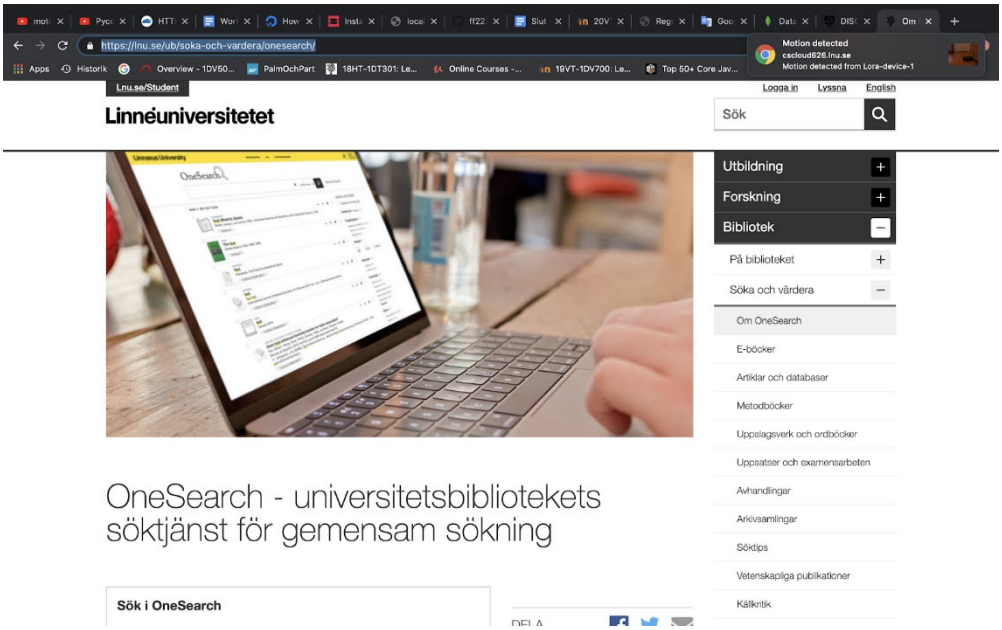

Figure 18. Notification when motion detected on LoRa-device (Upper right corner).

3.2.3.4 Saving events to the database

Upon motion detection, the event was saved at the MongoDB database. An event schema was created, which was stored as in *figure 19*.


```
_id: ObjectId("5e6752f0c013fe114b8e5ae8")
title: "motion detected"
deviceID: "Lora-device-1"
imgUrl: "https://linnaeus.asuscomm.com:8081/img?id=60704cf1fb69a8e2c85a6b68326c..."
time: 2020-03-10T08:42:24.094+00:00
__v: 0
```


Figure 19. Event data stored at MongoDB.

All past events were shown to the user in the “Past events” section, see *figure 20*.

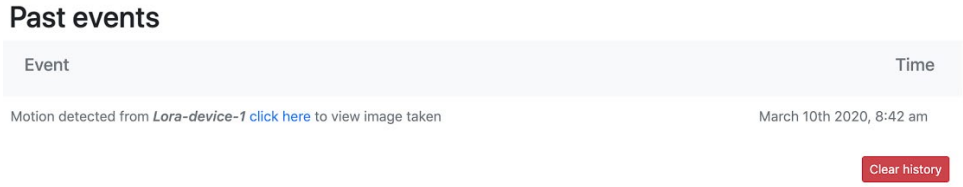

Figure 20. Past events shown in browser, fetched from MongoDB.

3.2.3.5 Production server

Since all coding was made on a local machine, an implementation of uploading the project to the server had to be made. Deployment through git was implemented, over other methods like FTP, SSH Secure Copy (SCP). Since a GitHub repository was used as version control, that motivated the use of git. A repository folder was created on the server, along with a bash script.

The bash-script *[Appendix 2]*, was implemented in the post-receive folder, enabling installation of new modules and application restart by the package manager PM2.

3.2.4 LoRaWAN

At first the firmware on the LoPy module had to be updated. A software for updating the firmware found at forums and was installed, since the Pycom documentation was not updated and contained broken links. Setting up the development environment was done by installing Pymakr plugin for Atom [15].

An application was set up at The Things Network website and the device was registered using the device EUI, which was retrieved by running a code found at the Pycom website [16].

To connect to TTN a boilerplate code was extracted from the Pycom website [17] and tested for both OTAA and ABP. None of the methods could connect to any gateway at first, but as the data rate was lowered, which also extends the range, uplink messages could be sent. Downlink messages was only possible with OTAA, which was the main reason this method was used, since acknowledgements from the application server was needed to ensure detection was noticed by user.

When connectivity was established with TTN, a script was written at their website, to parse and push data as JSON, to ease the interpretability at the application server, see *figure 21.* The script reads the byte array and decodes it into a human readable string.

As the LoRa-device code evolved with sensors attached to the module, the LoPy experienced core panics and rebooted itself. This seemed to be a known issue and was fixed with a new firmware release.

```
function Decoder(bytes, port) {
  // Decode an uplink message from a buffer
  // (array) of bytes to an object of fields.
  var decoded = {};

function uintToString(uintArray) {
    var encodedString = String.fromCharCode.apply(null, uintArray),
        decodedString = decodeURIComponent(escape(encodedString));
    return decodedString;
}

  decoded.value = uintToString(bytes)
  decoded.device = "Lora-device-1"
  decoded.message = "State change detected."
  return decoded;
}
```

Figure 21. Decoding script at TTN application website..

3.2.5 Motion detection sensors

Initially a PIR SR501 motion detector was used for motion detection. The detector worked well and was easy to install, giving digital output value of motion detection. However, both participants of this project will use an ultrasonic sensor in an upcoming thesis project, which was why the ultrasonic sensor was chosen. An LV-MaxSonar-EZ0 sensor was tested at first, but gave large fluctuations in distance measurements. Therefore, the HC-SR04 was tested instead, which gave more precise results.

To measure distance, the trigger pin has to be held high for at least 10 µs, which triggered the transmitter to send 8 ultrasonic bursts. When the last burst was transmitted, the echo-pin was held high until 8 bursts was received by the receiver. The echo pin would then go low, see *figure 22*. From the pulse width of the echo pin the distance to target can be calculated by *equation 1*.

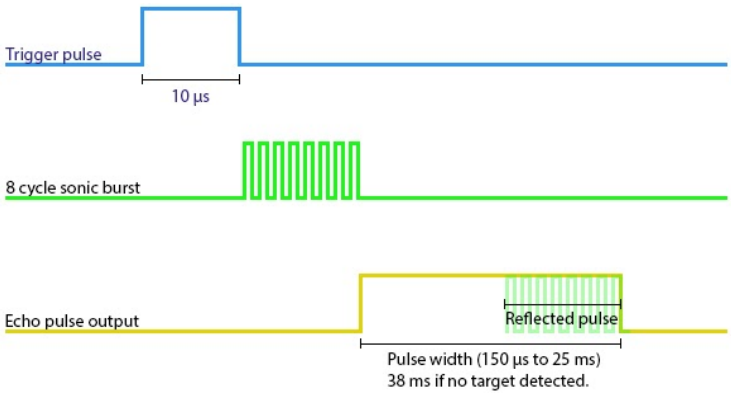

Figure 22. Timing diagram for HC-SR04.

$$d = \frac{t \times v}{2 \times 10^4}\ (cm)$$

Where,

$$v = speed\ of\ sound = 331.4 + 0.6 \times T_c\ (m/s)\ [18],$$
$$t = time\ measured\ (\mu s),$$
$$T_c = Environmental\ temperature,$$
$$d = distance\ between\ sensor\ and\ object$$

Equation 1. Calculation of distance to target, based on time and speed of sound.

Since no library for HC-SR04 was found for LoPy, the code in *figure 23* was written. The trigger pulse was set high for 10 µs and echo pin timed from being high until going low to get the pulse width. With the measured data the distance was calculated from *equation 1*.

To get a correct distance value from the ultrasonic sensor, the temperature was measured with MCP9700A temperature sensor. The output voltage was measured (mV) and calculations were made, see *equation 2*, to retain the environmental temperature. To make those calculations a freezing point voltage and temperature coefficient was extracted from the sensor datasheet [19].

```
run_sensor(trigger, echo, analog_pin):
# Make sure output is low
trigger.value(0)
utime.sleep_us(10)

# Trigger 8 cycle sonic burst by setting trigger to high for 10us
trigger.value(1)
utime.sleep_us(10)
trigger.value(0)

# Wait for pulse to start on input pin.
while echo() == 0:
    pass
start = utime.ticks_us()

# Run until input pin changes to low.
while echo() == 1:
    pass
finish = utime.ticks_us()

utime.sleep_ms(10)

T_A = temp.readTemp(analog_pin)
speed_sound = (331.4 + 0.6 * T_A) * 0.0001        # cm/us

# Calculate and print out distance measured.
return ((utime.ticks_diff(finish, start)) * speed_sound)/2
```

Figure 23. Code implementation to measure distance with HC-SR04.

$$V_{out} = T_A \times T_C + V_{0℃} \Leftrightarrow T_A = \frac{(V_{out} - V_{0℃})}{T_C} \ (℃)$$

Where,

$$V_{out} = Measured\ output\ voltage\ (mV),$$
$$V_{0℃} = Voltage\ at\ freezing\ point (= 500\ mV),$$
$$T_C = Temperature\ coefficient\ \left(= 10\ {}^{℃}/_{mV}\right),$$
$$T_A = Output\ (environmental)\ temperature\ (℃)$$

Equation 2. Environmental temperature equation, based on measured voltage and known values.

3.2.6 Video stream server

For the video stream server, a Raspberry Pi module was chosen, along with the camera module. A complete setup of the OS was made, installing NOOBS and Raspbian. To manage the Raspberry, it had to be connected to a monitor and connect an external keyboard and mouse. After the operating systems were installed, firmware was updated through the commands *sudo apt-get update* and *sudo apt-get upgrade*, which took about two hours to complete.

After upgrading, SSH was enabled and the software XRDP was installed to setup a remote desktop with GUI. At a later stage VNC server was set up on the Raspberry for remote access using VNC viewer at the other end, which worked better on all platforms.

To test the camera the command *raspistill -o imagename.jpg* was run in the terminal, which resulted in an image in the prompted folder. When the camera was confirmed working, Motion was installed on the Raspberry, which is a large library for image/video processing [20]. A video stream could be started and configured by changing some parameters in a configuration file.

When doing further reading on the camera setup, a boilerplate script for a multithreaded video streaming server was found at the Picamera website [21]. This allowed accessing the camera without the Motion library, which also resulted in much better frame rates after some configurations. The resulting script, after modifications to fit the project needs, contains the original setup for camera access and HTTP-server setup.

The server has the possibility of viewing images and video stream, which were used in two separate ways with HTTP GET requests:

3.2.6.1 Image saving:

The application server will upon receiving data of detected motion request to save an image on the Raspberry. This is done by a simple GET request with *id* as query string, containing a randomly created file name connected to the detected motion. The following URL should be used, */img/save?id=IMG_NAME.*

3.2.6.2 Image fetch:

Upon creating a notification to client in the application, the image will be fetched by a GET request to the URL */img?id=IMG_NAME*

3.2.6.3 Video stream access:

The request of a video stream is a two-step process. At first the client requests a stream ID, which contains of a random number, a random string and a socket connection id, concatenated together. The stream ID is requested by GET request to the URL */setstream?id=SOCKET_ID.*

This request will return a JSON object with the stream ID. To gain access to the stream the stream ID has to be handled in a correct way, to get a valid stream URL.

The random number indicates where, in the stream ID, to extract a substring which will be the valid URL query (Example: *2helloworld*, will be *elloword*, since it starts on index 2 of the stream ID). The stream ID can only be used once since it will be removed after it is used and therefore gains some security against unwanted stream usage, as knowledge about how to access the stream is needed. The stream is accessed by GET request to URL */stream?id=VALID_STREAM_ID.*

To gain access to the stream from anywhere a port forwarding rule had to be set at the LAN router (Asus RT-N66U) at the Raspberry Pi location. The port used for incoming requests was 8081 on both sides of the router. DDNS was implemented on the router to enable the possibility for HTTPS through Let's encrypt certificates using the domain name, *linnaeus.asuscomm.com*. The certificate was installed on the Raspberry Pi, using Certbot [13] and the domain for the certificate was set to *linnaeus.asuscomm.com:8081*.

3.3 Result

- Full project code: https://github.com/LasseUlvatne/IoT-Project-2DT301
- Project video presentation: https://youtu.be/6AjE3gcFpCU

As a product of the first aim in section *3.1.2*, a web application was formed. The web application interconnects the IoT-device, used for motion detection, with the camera server to enable surveillance and to notify the end user of registered motion.

The IoT-device, *figure 24*, managed to send sensor data upon motion detection over LoRaWAN. When The Things Network intercepted the data a uplink event was sent to the application server.

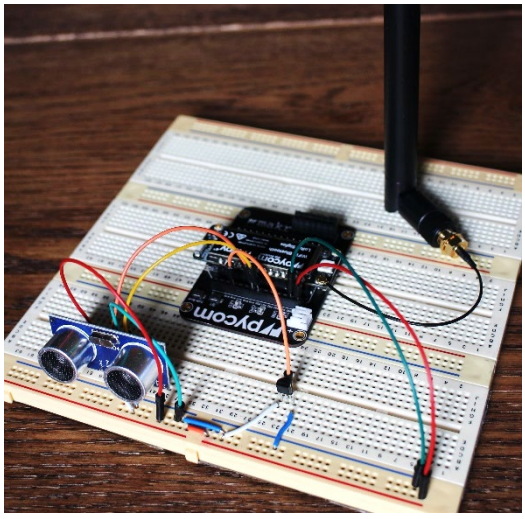

Figure 24. IoT-device, containing LoRa and sensors.

A camera server could serve images or start a video stream upon incoming HTTP requests from the application server, triggered upon motion detection. See the components in *figure 25*.

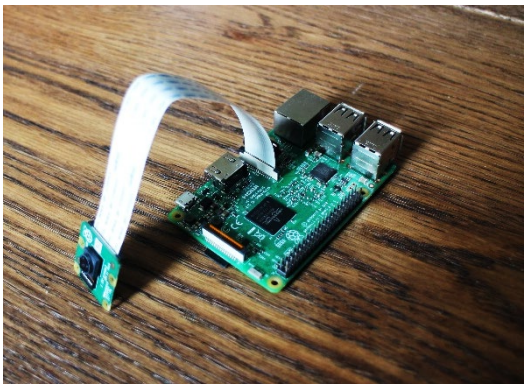

Figure 25. Raspberry Pi used as camera server.

The final surveillance application interface can be viewed in *figure 26*. It included notification messages, past events and possibility to view the live stream. From the final prototype application, the evaluation of the second aim in section *3.1.2* was addressed, and discussed in the next section.

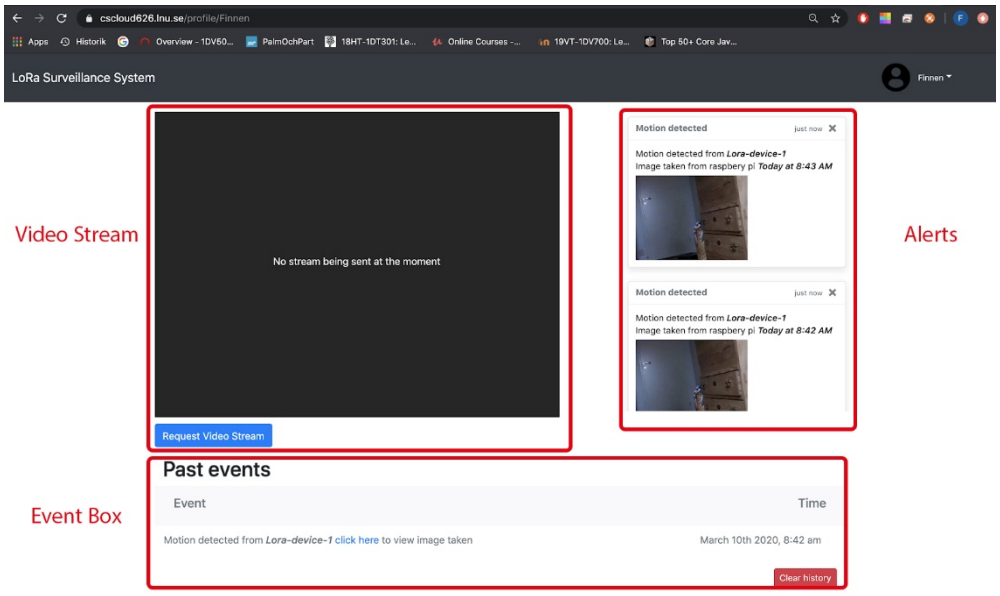

Figure 26. Application screen, with highlighted areas.

4. Discussion and final remarks

The aim of the project, as mentioned in section *3.1.2*, was to create a motion detection system that uses LoRa as the method of communication. In the previous section we got a result that was pretty much what we expected, with some exceptions.

A motion detection system had successful long-range communication, which also notifies the user upon detection, by sending an image to let the user choose to start a stream or not. The implementation method fulfills the first research aim and could serve as an answer for the first research question.

As a motion detector, the ultrasonic sensor works well. However, we think that a normal motion detector would be better due to its longer distance range and wider search angle. The ultrasonic sensor was chosen solely of gaining knowledge for a later thesis project.

Using LoRaWAN as sensor data communication channel worked as we expected. We had some problems with connecting the node to a gateway, since the widest range had to be used, which also gives the lowest data rate, with bandwidth of only 0.25 kbps. As packages are sometimes lost and the data rate is low, the communication time needed to inform the end user would sometimes be too long. If, as the introduction imposes, we had the possibility of implementing our own gateway nearby the camera device and the LoRa-node, a much higher data rate channel could be used. Maybe even the device high speed or FSK channels which would increase the data rate up to 200 times. Also, the step using The Things Network could have been skipped (with own gateway), which also would make data transfer faster.

The Raspberry Pi served as camera module with streaming service and proved itself to be a powerful tool for this quest. The possibility to run a server on the computer along with the ease of installing a camera to its peripherals, suited our needs and helped us to reach the goal of that part of the project. Since there was a lot of documentation from other projects and informational sources, different strategies could be tested to give the best result.

As mentioned in the introduction section, an initial aim for this project was to have the camera at the LoRa-device. This was rapidly turned down due to the low data rate, since it would not be feasible to send frames of the resolution used (640x480). Each pixel consists of 8 bits (1 byte) [22] and a frame would be above 2.4 Mbits, which forced us to have the camera at a location where high-speed transfer was at hand.

Since the goal of implementing an IoT-device along with a camera server running on a different medium was successful, we can now vision a possible implementation of multiple sensors and the usage of a single camera module. This leads us into the second aim of this project in section *3.1.2*.

An implementation with multiple LoRa-devices, with one or more sensors connected to each device, seems very feasible to use to cover larger areas for motion detection. A single camera could be used in this case, but then needs good rotating and zoom possibilities. The main idea is to have known angle positions for each LoRa-device and upon motion detection the camera will focus to that position. Depending on the needs for this implementation, this could pose some obvious problems. If only one camera is used, does it have full 360° vision with full rotation and height control? What if motion is detected at different locations at the same time? These questions do really need the situation at site to be evaluated to full extent, but the solution to both problems could be implementing more cameras.

How does this relate to the second research question asked in section *3.1.1*? We would argue that the implementation of multiple sensors could reduce the number of camera resources needed than if no motion detection devices were used. An evaluation of how many LoRa-devices needed in relation to the cost of reduced camera modules has to be made at site, but hands the situation another possible technique to implement. The low powered LoRa-device makes it good for this task, since the battery levels could hold for multiple years [2].

5. Conclusions

5.1.1 Project conclusion summary

This project has implemented the whole chain of sensor reading, video stream server and application interaction with the end user, which works for the intended scope. Of course, upgrades for the projects can be used in better sensor modules, camera and an own implementation of a LoRa-gateway. However, each device technique used, along with the server implementations seems to be good solutions for the intended use.

5.1.2 Extensions and problems

- Implement multiple motion detection sensors.
- Use a rotating camera to focus on sensor area upon detection.
- LoRaWAN should only be used for small data transfer (e.g. sensor data).

6. References

[1.] Pierre Neumann, Julien Montavont, Thomas Noël, “Indoor deployment of low-powered wide area networks (LPWAN): A LoRaWAN case study”, 2016. [Online] https://www.semanticscholar.org/paper/Indoor-deployment-of-low-power-wide-area-networks-A-Neumann-Montavont/569a6e4e91e7abe03879ab1c2e37c077da520ae9/figure/9

[2.] LoRa Alliance Technical Marketing Workgroup, “LoRaWAN – What is it?”, 2015 (PDF), [Online] https://www.tuv.com/media/corporate/products_1/electronic_components_and_lasers/TUeV_Rheinland_Overview_LoRa_and_LoRaWANtmp.pdf

[3.] The Things Network Documentation, [Online] https://www.thethingsnetwork.org/docs/lorawan/

[4.] The Things Network Documentation, Gateways, [Online] https://www.thethingsnetwork.org/docs/gateways/

[5.] Pycom product documentation, LoPy4, [Online] https://pycom.io/product/lopy4/

[6.] Pycom product documentation, Expansion Board 3.1, [Online] https://pycom.io/product/expansion-board-3-0/

[7.] Pycom product documentation, LoRa/Sigfox antenna, [Online] https://pycom.io/product/lora-868mhz-915mhz-sigfox-antenna-kit/

[8.] Elec Freaks, HC-SR04 Ultrasonic sensor datasheet, PDF, [Online] https://www.electrokit.com/uploads/productfile/41013/HC-SR04.pdf

[9.] Raspberry Pi product documentation, Raspberry Pi 3 Model B+, [Online] https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

[10.] Raspberry Pi product documentation, Raspberry Pi Camera Module V2, [Online] https://www.raspberrypi.org/products/camera-module-v2/

[11.] npmjs website, [Online] https://www.npmjs.com/

[12.] John Häggerud, “Revese proxy”, Linnaeus University, Kalmar, 2018. [Online video] https://www.youtube.com/watch?v=Wod40I7Ob50&feature=youtu.be

[13.] Certbot website, [Online], https://certbot.eff.org/

[14.] Mats Look, Linnaeus University, 1DV523 – Server based webprogramming, 2019, [Online] https://rawgit.com/1dv023/syllabus/master/lectures/03/index.html#/4

[15.] Pycom documentations, “Pymakr plugin for Atom” [Online], https://docs.pycom.io/pymakr/installation/atom/

[16.] Pycom Documentations, “Device registration” [Online], https://docs.pycom.io/gettingstarted/registration/lora/

[17.] Pycom Documentations, “LoRa examples” [Online] https://development.pycom.io/tutorials/lora/

[18.] NDT Resource Center, “Temperature and the speed of sound”, [Online] https://www.nde-ed.org/EducationResources/HighSchool/Sound/tempandspeed.htm

[19.] Mouser Electronics, “MCP9700A Datasheet”, 2005 [Online PDF], https://www.mouser.com/datasheet/2/268/20001942F-461622.pdf

[20.] Motion-project, “Basic setup”, [Online] https://motion-project.github.io/motion_config.html

[21.] Picamera documentation, “Web streaming”, [Online] https://picamera.readthedocs.io/en/latest/recipes2.html#web-streaming

[22.] Norman Koren, Photographer, “Pixels and images”, 2000 [Online], http://www.normankoren.com/pixels_images.html

Appendix

1. NGINX code:

```
server {
    listen 80;
    server_name cscloud626.lnu.se;
    return 301 https://$server_name$request_uri;
}

server {
    server_name cscloud626.lnu.se;
    location / {
        proxy_pass http://localhost:3000;
          proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "Upgrade";
    }
    ssl on;
    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate
/etc/letsencrypt/live/cscloud626.lnu.se/fullchain.pem; # managed by
Certbot
    ssl_certificate_key
/etc/letsencrypt/live/cscloud626.lnu.se/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

server {
    if ($host = cscloud626.lnu.se) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80;
    listen [::]:80;
    server_name cscloud626.lnu.se;
    return 404; # managed by Certbot
}
```

2. Production Server Code:

```
git --work-tree=/var/www/cscloud626.lnu.se --git-
dir=/var/repo/site.git checkou$

cd /var/www/cscloud626.lnu.se
```

```
npm install --production
SERVICE="node /var/www/c"

if pgrep -x "$SERVICE" >/dev/null
then
    echo "$SERVICE restarted"
    sudo pm2 restart app.js
else
    echo "$SERVICE up and running"
    sudo pm2 start app.js
fi
```

3. Full project code: https://github.com/LasseUlvatne/IoT-Project-2DT301
4. Project video presentation: https://youtu.be/6AjE3gcFpCU