UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF ELECTRONICS, TELECOMMUNICATIONS AND INFORMATION TECHNOLOGY

INFORMATION TRANSMISSION BETWEEN A TERRESTRIAL DRONE AND AN UAV

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the Degree of Engineer in the domain *Electronics and Telecommunications*, study program *Technologies and Communications Systems*

Thesis advisor:
Prof. Corneliu BURILEANU, Ph. D.

Student:
Ioana BĂDIȚOIU

Bucharest
2018

University "Politehnica" of Bucharest
Faculty of Electronics, Telecommunications and Information Technology
Department **Tc**

Anexa 1

DIPLOMA THESIS
of student **BĂDIŢOIU I.C. Ioana , 441G**

1. Thesis title: Information Transmission between a Terrestrial Drone and an UAV

2. The student's original contribution will consist of (not including the documentation part):
The purpose of this thesis is the design and practical implementation of a referential model of correlation and coordination of the actions that will be pursued between an UAV and a terrestrial drone. Taking these into account, Wi-Fi, ZigBee, LoRa and NBIoT protocols will be studied in order to identify the advantages and weaknesses of each of them within the projected application.
I will design an application which can be run from any system. The purpose will be to send the UAV flight coordinates. A terrestrial drone will be equipped with a GPS system.
The terrestrial drone and the UAV will periodically communicate their positions (predefined reporting interval and differentiated depending on the type of drone).
The coordinates of the drones will be stored on a MySQL server hosted on a Raspberry PI development board - the connection between the two will be wireless using the protocols specified above. Depending on the contextual position of the UAV, the terrestrial drone will take a decision to change the direction of movement.
By connecting to RaspberryPI, the positioning of the drones can be highlighted on the map in the browser.

3. Pre-existent materials and resources used for the project's development:
UAV, terrestrial drone, Raspberry PI, XBee modules, Java SE, Android SDK, Java, Android

4. The project is based on knowledge mainly from the following 3-4 courses:
Microprocessors Architecture, Microcontrollers, Communications Networks

5. The Intellectual Property upon the project belongs to: U.P.B.

6. Thesis registration date: 2017-11-29 13:58:00

Thesis advisor(s),
Prof. dr. ing. Corneliu BURILEANU
signature:.....

Student,
signature:.....

Departament director,
Conf. dr. ing. Eduard POPOVICI
signature:.....

Dean,
Prof. dr. ing. Cristian NEGRESCU
signature:.....

Validation code: **d96cdd7c70**

Statement of Academic Honesty

I hereby declare that the thesis *"Information Transmission between a Terrestrial Drone and an UAV"*, submitted to the Faculty of Electronics, Telecommunications and Information Technology in partial fulfillment of the requirements for the degree of Engineer of Science in the domain *Electronics and Telecommunications*, study program *Technologies and Communications Systems*, is written by myself and was never before submitted to any other faculty or higher learning institution in Romania or any other country.

I declare that all information sources sources I used, including the ones I found on the Internet, are properly cited in the thesis as bibliographical references. Text fragments cited "as is" or translated from other languages are written between quotes and are referenced to the source. Reformulation using different words of a certain text is also properly referenced. I understand plagiarism constitutes an offence punishable by law.

I declare that all the results I present as coming from simulations and measurements I performed, together with the procedures used to obtain them, are real and indeed come from the respective simulations and measurements. I understand that data faking is an offence punishable according to the University regulations.

Bucharest, *June 2018*

Ioana BĂDIȚOIU

(Student's signature)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

A

ABP = Activation by Personalization
AES = Advanced Encryption Standard
AP = Access Point
AppSKey = Application session key

B

BSS = Basic Service Set

E

EGPRS = Enhanced General Packet Radio Service
ESS = Extended Service Set

F

FFD = Full-Function Device

G

GPRS = General Packet Radio Service
GSM = Global System for Mobile Communications

I

IBSS = Independent Basic Service Set
IoT = Internet of Things
IP = Internet Protocol
ISO/OSI = International Organization for Standardization/Open Systems Interconnection

J

JSON = JavaScript Object Notation

L

LoRaWAN = Long-Range Wide-Area Network
LPWAN = Low-Power Wide Area Networking
LTE = Long-Term Evolution

M

MAC = Medium Access Control
MIC = Message Integrity Code
MQTT = Message Queuing Telemetry Transport

N

NBIoT = Narrow Band Internet of Things
NwkSKey = Network Session Key

O

OTAA = Over The Air Activation

P

PAN = Personal Area Network

R

RAM = Random Access Memory

S

SPI = Serial Peripheral Interface

T
TCP/IP = Transmission Control Protocol/Internet Protocol

W

Wi-Fi = Wireless Fidelity

INTRODUCTION

THESIS MOTIVATION

Nowadays, people are keener to use technology in order to substitute daily tasks, activities that become repetitive or missions in dangerous areas. And these are just a few examples. Applications with drones, which are the main objective of this thesis, are widely known, these being used in many domains such as Agriculture, Networking, Geo Mapping, Filming, Military etc.

Moreover, the necessity of using wireless protocols increased exponentially in the last decade, as current technologies allow the use of a large number of monitoring and controlling devices on a relatively small surface. Besides Wi-Fi or ZigBee, which are the most known, many other protocols are used as an alternative depending on factors such as data requirements, power demands or range or battery life.

MAIN OBJECTIVE

In this context, the thesis aim is to explore and study wireless protocols that can be used in a communication between two moving devices – one being on the ground and the other in the air. More specifically, a protocol that ensures real time transmission between a terrestrial drone and an Unmanned Aerial Vehicle (UAV).

In order to meet all the requirements to the greatest possible extent, while taking into account other parameters such as: low implementation cost, high battery lifetime in a charging cycle and interference resistance, I have studied the use of a wider range of wireless protocols:

- ✓ Wireless Fidelity (Wi-Fi)
- ✓ ZigBee
- ✓ Long-Range Wide-Area Network (LoRaWAN)
- ✓ Narrowband Internet of Things (NBIoT)

The latter are two technologies that are still under development and their implementation in different fields such as military domain, has not yet materialized.

When talking about drones, both terrestrial and UAVs, new wireless communication solutions are constantly being sought, solutions that need to cope with any challenge, on distances as large as possible. Currently, the concept of switching from one wireless technology to another within the same communication process, depending on the operating parameters – distance, altitude, remaining battery, number of packets lost in a transmission, etc. is also studied.

To accomplish what I have proposed, I will follow the practical performance of the protocols mentioned above. Even though, in theory, they all fit into military fieldwork, a series of practical experiments will be needed to determine which is the most adequate protocol, considering all requirments. Everything needs to be taken into account: the distance from the signal source, where is the controlling point for the drones, the interferences that may occur, the number and type of obstacles that can affect the signal, etc.

The UAV that will be used for practical experiments is Lehamann Aviation LA300 – more details about this will be presented in the following chapters. The main advantage of this drone is that it offers the possibility of mounting or removing the weight that is transported during the flight (such as the camera provided or other additional components that are not necessary to be maintained during take-off and flight) while providing an easy installation of wireless transmission modules, in addition to those already in place for information transfer and synchronization with an application.

The terrestrial drone, which will host the server on which information will be kept, is a PhantomX AX Hexapod Mark II robot. This is an open source robot, which means that it completely fits the purpose of this paper.

SPECIFIC OBJECTIVES

- ✓ The paper is focused on investigating in comparison four different solutions: two are LAN centered (Wireless-Fidelity and ZigBee), one is proprietary type (LoRaWAN) – being the most evolved at the moment and considered generic from the perspective of other implementations and one is a solution in development, from the area of transition to 5G (NBIoT).
- ✓ By analyzing the advantages and disadvantages of different protocols from the Internet of Things perspective, the practical implementation will be done with the one that best fits the proposed use case. In this context, LoRaWAN protocol will be studied.
- ✓ In parallel with the IoT protocol implementation, it will be presented a brief description of the hardware that will be used: Lehmann Aviation LA300 UAV and PhantomX AX Hexapod Mark II robot.
- ✓ Two solutions will be implemented: one using an Internet of Things platform and the other, point to point communication.

The thesis is organized in 6 chapters, as follows:

Chapter 1 presents the motivation, the objectives and the outline of this thesis. In *Chapter 2* are exposed four different protocols, with advantages and disadvantages for my use case. *Chapter 3* details the one that best fits my desired implementation. *Chapter 4* presents hardware technologies which will be used to accomplish what I have proposed. *Chapter 5* mainly illustrates the contributions of the author of the thesis. It focuses on the actual implantations of the applications of this paper. Finally, *Chapter 6* summarizes the main conclusions of the thesis and underlines the author's contributions.

CHAPTER 1

EVOLUTION AND ANALYSIS OF WIRELESS PROTOCOLS FROM THE IOT PERSPECTIVE

1.1 INTRODUCTION

There is one major difference between “the Internet” and “the Internet of Things”, this being that the latter requires less of everything: less processing power, less memory, less bandwidth, etc. It is estimated that by 2020, 50 billion devices will be connected. This huge number of devices leads to constraints that limit the applicability of traditional technologies.

The general tendency of the present is to use wireless protocols for the physical level – for computer networks, the most commonly used protocol is Wi-Fi, while mobile communications networks use GSM protocols: 3G, 4G, GPRS, EGRPRS (EDGE), LTE, etc.

In terms of applications, higher levels of protocols stack (TCP/IP or ISO/OSI), there will be no difference when switching from classical, wired, to wireless (at most delays will occur, but if the application is done correctly, it should not make a major difference). When talking about application level, we mean any direct interaction with the user (web browser, games, switches, bulbs, etc.). These applications are "protected" by interfaces between layers of protocols stacks, allowing each software or hardware module to modify its operating mode, while maintaining communication with the other levels in exactly the same way.

Nowadays, there is a relatively large number of wireless protocols that can be used for various applications. In this chapter, I will present some of these protocols, the most used and useful of the existing ones, comparing them with the main protocol of the paper – Long-Range Wide-Area Network (LoRaWAN).

The protocols that will be analyzed are the following: Wireless-Fidelity, ZigBee, LoRaWAN and NBIoT. They will then be compared in terms of standards, evaluating the operation metrics, transmission time, coding efficiency, complexity and average power consumption. The advantages of each protocol will be highlighted, but we will focus on their utility in sensor networks, such as those used in this paper.

1.2 Wireless Fidelity (Wi-Fi)

When talking about Wi-Fi protocol, we have to refer to the following standards: 802.11 a/b/g/n/ac/ad. They permit the user to connect to a network when in proximity of an Access Point (AP) or through a router.

The architecture defines two infrastructure mode topology building blocks: Basic Service Set (BSS) and Extended Service Set (ESS). A BSS consists of an AP which interconnects more clients. If one of the clients that is connected wireless goes outside the Basic Service Area (the coverage area), communication is interrupted.

ESS, on the other hand, is formed when there is not enough radio frequency coverage and two or more BSSs must be joined together through a distribution system.

There is also an ad hoc mode where wireless stations can communicate directly with each other. The building block that stays at the base of this implementation is called Independent Basic Service Set (IBSS).

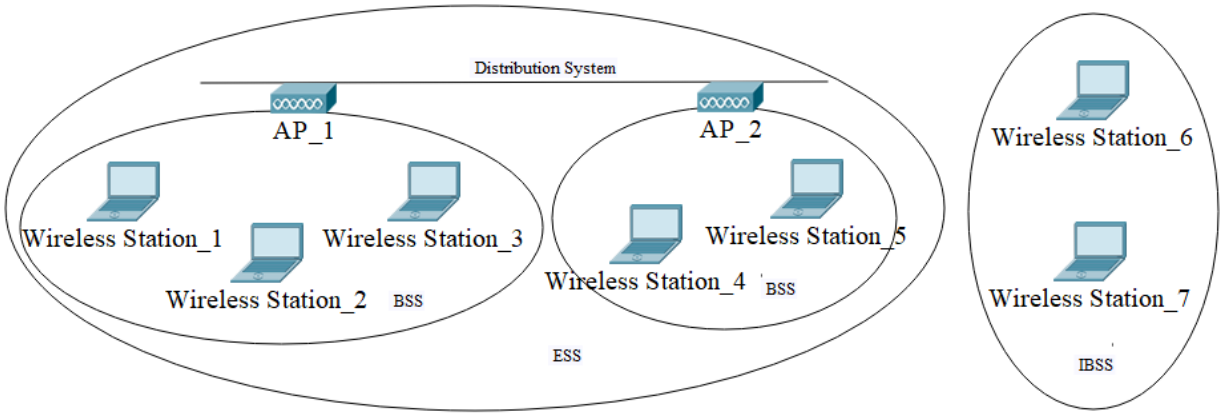

Figure 2.1 – Wi-Fi systems principle

What represents the main advantages of this protocol are the data rate which goes up to 1.3 Gbps and a high resistance to interference caused by transmission environment. However, speed is not a key factor and other characteristics will be more important.

The distance, which is one of the key factors because in this paper I want to implement long-range networks, is a drawback when talking about Wi-Fi. The typical maximum range of a standard equipment that uses this protocol is maximum 100 meters. This is enough for typical homes but insufficient if we consider larger structures.

Moreover, another major drawback is the need for intermediate equipment whose price may increase exponentialy when discussing the connection of multiple nodes (over 10-20). Although they can connect multiple nodes to cheap equipment, their internal components (CPU, RAM, ROM) do not handle traffic volume.

1.3 ZIGBEE

ZigBee, as described in IEEE 802.15.4 standard, is a Wireless Personal Area Network (WPAN) low data rate transmission protocol. It has been designed for a simple connection between devices, keeping power consumption at a minimum.

The ZigBee network is self-organizing, requiring a minimum intervention of the user or administrator at the time of the initial setup. Subsequent interventions are required only in situations with major problems where a large number of nodes are defective or if the running configurations are deleted and reset. Networks organized by ZigBee can be both multi-hop – star and mesh.

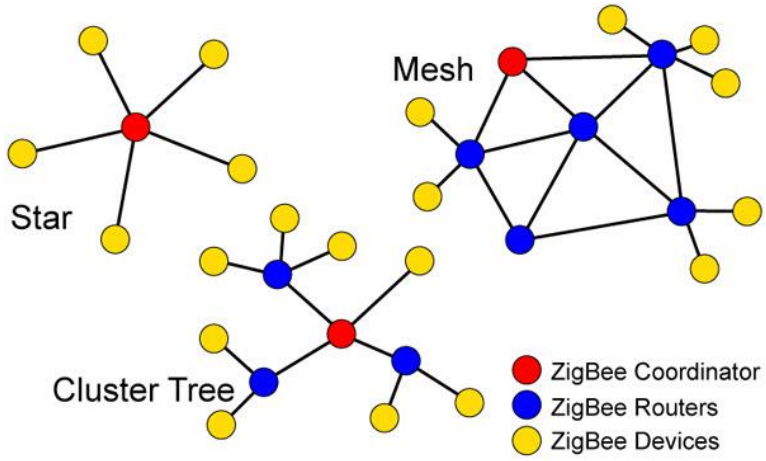

Figure 2.2 – Possible ZigBee Topologies

Within a ZigBee network, devices have two ways of working: full-function device (FFD) or reduced-function device (RFD). FFDs can perform three roles: Personal Area Network (PAN) Coordinator, Router or End-device and it can communicate actively with other FFDs or RFDs whereas a RFD performs only a limited number of tasks and can communicate only with other FFD devices.

Nodes that play the role of RFD have less important purposes within the network, most of the time being passive devices (switch, passive infrared sensor). They do not need to transmit large amounts of data and they can communicate with only one FFD at the time.

Once a fully functional node has been activated for the first time, it is able to form its own network and become PAN coordinator, forming a star network. Several star networks can work independent of each other, separated by an unique network identifier. Once a PAN identifier has been chosen, the coordinator can allow other nodes to connect to the network.

Due to the advantages of the protocol, the possibility of data encryption, of connecting a large number of nodes within a single network (> 65,000), ZigBee is well suited for a large range of IoT applications, but also to more complex applications. Some examples are: automation of home and work processes (starting the coffee machine, washing machine, refrigerator, etc.), medical monitoring (EEG, EKG), safety monitoring and seismological monitoring. Within these applications, ZigBee compatible nodes can be battery-powered, the power consumption associated with the protocol being very low.

1.4 LONG-RANGE WIDE-AREA NETWORK (LORAWAN)

In order to fully understand what Long-Range Wide-Area Network (LoRaWAN) is and what is the purpose for developing such a protocol, firstly I have to define and explain the concept of Low-Power Wide-Area Network (LPWAN).

In a world where connecting many things over long distances became a priority in order to make smart cities, agriculture, asset monitoring and tracking, metering and so on, LPWAN became a necessity. In order to achieve multi-kilometers communication range, LPWAN combines robust modulation and low data rate.

These were meant to fill the gap between cellular networks (GSM, LTE, UMTS) and short-range high-bandwidth networks (Bluetooth, Wi-Fi, and ZigBee).

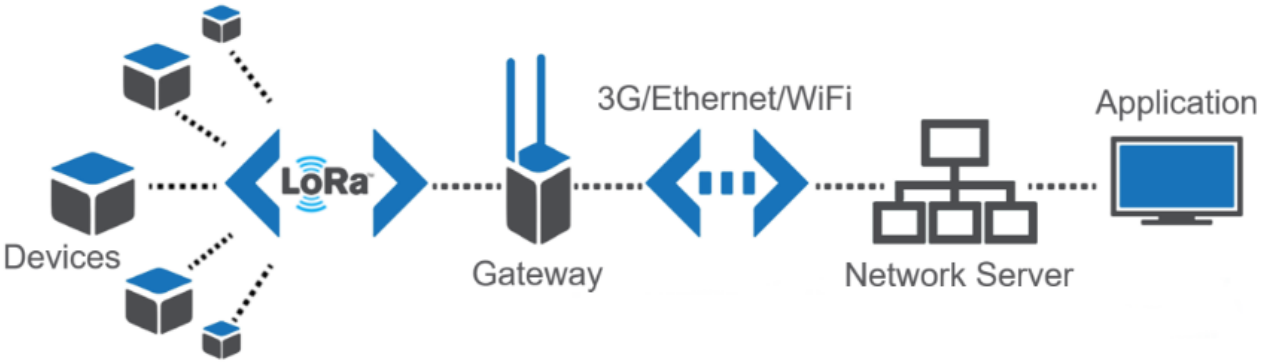

Figure 2.3 – The architecture of a typical LoRaWAN network

There are several solutions such as Ultra Narrow Band including SigFox, NBIoT, Ingenu or Weightless W, N and P.

Taking into account recent analysis, we can affirm that SigFox, together with LoRaWAN are the most used at the moment. However, the restrictions which are applied on SigFox (such as frequencies, maximum packet payload and the number of packets per device per day) make LoRaWAN a more desired solution, being considered more flexible and open.

LoRaWAN protocol was specified by the LoRa Alliance in 2015 and from that moment, has grown rapidly, being adopted by numerous telecommunications providers or electronics companies.

The topology used by LoRaWAN is a star-of-stars. Communication is bidirectional and works in the following manner: gateways gather together data from end-devices which are sent over a single wireless hop. Next, gateways are connected through a non-LoRaWAN network to the network server.

Three types of devices are defined within the standard: Class A, B and C, each of them with different capabilities.

More details concerning this protocol will be found in *Chapter 3*, where I intend to describe in detail the characteristics of LoRaWAN.

1.5 Narrowband Internet of Things (NBIoT)

As stated earlier, Narrowband IoT is a Low Power Wide Area Network radio access technology developed by 3rd Generation Partnership Project (3GPP) that can enable new IoT services and devices.

NBIoT provides the folowing features:

- Extended coverage; it has a signal gain higher with 20 dB than LTE so the emitting power rate increases consistently. This means that also coverage improves;
- Suport of massive number of low throughput devices;
- Low power consumption;
- Optimized network architecture;

- High allowed latency;

Figure 2.4 – NBIoT device

While coexisting with 2G, 3G and 4G and being considered a solution of transition to 5G mobile network, it is supported by all major mobile equipment, chipset and module manufacturers. It is a versatille solution that can be deployed on existent GSM/UMTS/LTE cellular networks.

It offers three deployement scenarios, which are the following:

1. Standalone Solution

The most typical Standalone deployment is to introduce NB-IoT in the GSM band (typically 900MHz) [4].

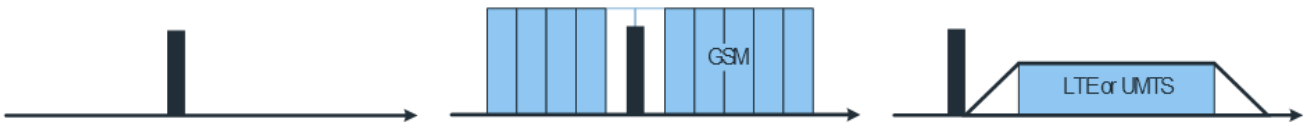

Figure 2.5 – Standalone Solution

Adding more NB-IoT carriers is possible but impacts GSM capacity and frequency planning.

2. In-band Solution

With In-band deployment, a NB-IoT carrier is introduced within the LTE carrier, the power being taken from the LTE carrier.

In case in the In-band solution is used the IoT, capacity of the baseband can be used for broadband services when there is no NBIoT traffic. This means dynamic sharing of broadband resources between LTE and NBIoT [4].

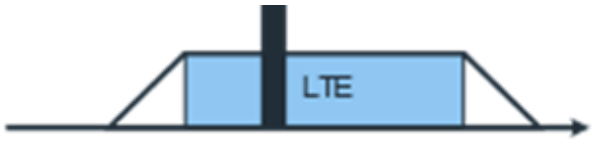

Figure 2.6 – In-band Solution

This solution has the advantage that it is easy to scale by adding more NB-IoT carriers.

3. Guard Band Solution

We are talking about Guard Band deployment if we introduce NB-IoT in the guard band of LTE.

However, this means limited possibilities to scale (one or max 2 NB-IoT carriers per guard band) [4].

Figure 2.7 – Guard Band Solution

1.6 CONCLUSIONS

Each of the protocols presents characteristics and advantages that recommend themselves for usage in different applications but, at the same time, disadvantages for what I have proposed to implement along this thesis.

In Table 2.1 I will present general characteristics of presented protocols, together with LoRaWAN.

Protocol	Wi Fi	ZigBee	LoRa	NB-IoT
IEEE spec.	802.11 a/b/g/n/ac/ad	802.15.4	802.11 ah	No spec.
Bandwidth	2.4 GHz; 5 GHz	868/915 MHz; 2.4 GHz	868/915 MHz; of 125 kHz	According to national regulations
Throughput	Up to 1.3 Gbps	250 kbps	27 kbps	~ 200 kbps
Coverage	100 m	10 - 100 m	5 – 15 km	Up to 20 km
Topology	BSS, ESS	star, peer-to-peer, mesh	star of stars	star
Transmission Power	15-20 dBm	-25 (0 dBm)	−4 dBm to 20 dBm	23 dBm
Channel Bandwidth	22 MHz	0.3/0.6 MHz; 2 MHz	125 kHz/500 kHz (Europe)	180 KHz
Maximum number of nodes	>20, 25	>65000	thousands of nodes	thousands of nodes
Data Cryptography	RC4 (WEP)	AES	AES128	LTE data transmission encryption: AES
Authentication	WPA2	CBC-MAC (CCM)	ABP/OTAA	LTE based
Data Security	CRC 32-bit	CRC 16-bit	CRC 16-bit	LTE based

Tabel 2.1 – General characteristics

Considering all the characteristics and limitations listed above, in the next chapters I will focus on LoraWAN and its practical implementation.

I chose this protocol because it is a relatively new standard which has grown rapidly and it has been adopted by many companies worldwide.

In fact, there are many reasons that made me believe that LoRaWAN is the best choice, among Wi-Fi, ZigBee or other proprietary LPWAN technologies that are also hitting a large market, for the implementation that will be described later in this paper.

IoT vision requires long-range communication while interconnecting more sensor nodes. This means that energy consumption is an important issue that must be addressed. Wi-Fi clearly remains behind the trend with high energy consumption and, as it can be seen from the table, a small number of devices that can be interconnected. Moreover, to allow a point-to-multipoint connection, devices must run a software that consumes more internal hardware resources, and their consumption increases as the number of connected devices increases. Even though this would make ZigBee a good alternative, we also have to take into account the distance covered.

Because we are talking about long-range communication and ZigBee only covers up to 100 meters, this would be considered a major disadvantage, taking it out of the race. A LoRaWAN gateway can cover up to 15 kilometers, almost the same distance covered by NBIoT, which unfortunately does not have any practical implementation in Romania yet. Considering all of these, together with the ability to serve thousands of end-devices with a low-power consumption, LoRaWAN is considered the best choice for my implementations, described more detailed in *Chapter 2*.

CHAPTER 2

LONG-RANGE WIDE-AREA NETWORK (LORAWAN)

2.1 INTRODUCTION

There is a difference between LoRa and LoRaWAN, which I want to state from the beginning: the first one defines the physical layer, while the latter is the protocol which is based on LoRa. Even though there is no restriction on using this protocol, LoRa Alliance decided that it is better to specifically design LoRaWAN for this purpose because many protocols already existent would lack security at MAC level or would trigger a high amount of communication to a single gateway. Taking these into account, LoRaWAN was developed to allow mobility without handovers.

In Figure 3.1 it is presented LoRa Protocol stack with the two distinct layers: the physical layer using the Chirp Spread Spectrum (CSS) radio modulation technique and the MAC layer protocol (LoRaWAN).

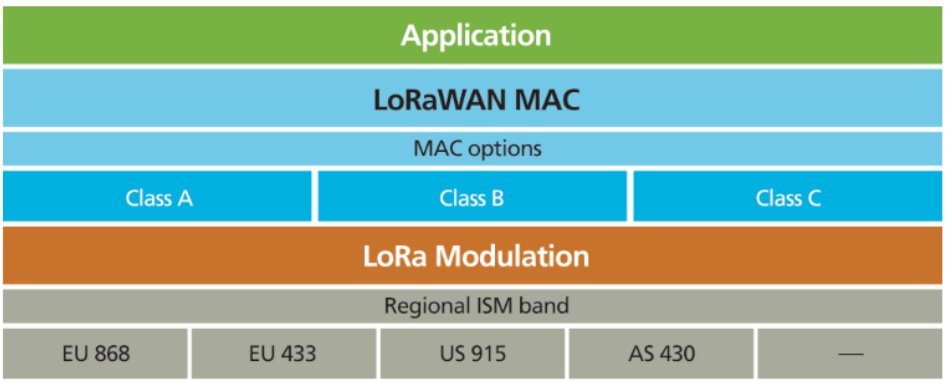

Figure 3.1 – LoRa physical layer and LoRaWAN [5]

In this Chapter, I will provide an independent analysis of LoRaWAN Protocol, LoRa, the physical layer, types of end devices, LoRaWAN networks and devices' connection and ways of securing communication within LoRaWAN networks.

2.2 LoRaWAN Protocol

In a LoRaWAN topology, we can distinguish between three types of components: Nodes, Gateways and Network Servers. Figure 3.2 displays a "star-of-stars" network topology with the elements mentioned before. A star topology is one the most common models and it consists of a central node to which other nodes are connected, being simpler to maintain than mesh networks. This means, for a LoRa network, that multiple Nodes are connected to one Gateway and, multiple Gateways to a single Network Server.

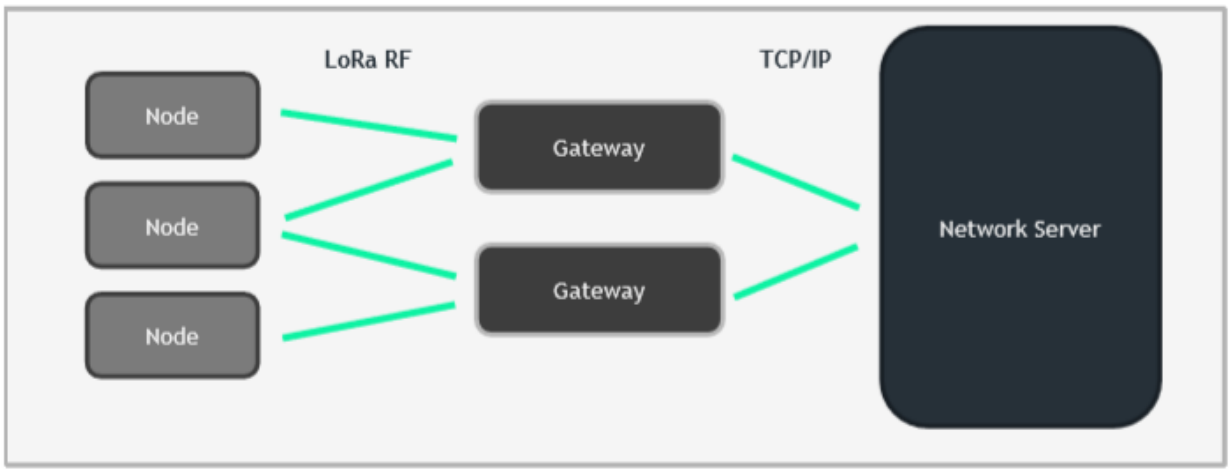

Figure 3.2 – "Star-of-stars" LoRa Network Topology [11]

Nodes are also known as end-devices and they are used to measure or control external systems, being formed by a microcontroller which manages a LoRa transceiver. They communicate wirelessly with gateways and are low powered.

They are divided into three classes as it will be explained later in this chapter and, depending on these classes, they will work in a different configuration. Even though they can listen all the times, the most common and low-power consuming option is to work in a "call then listen" configuration. This means that after the Node sends data to a Network Server, it will have short windows to listen for data coming back from Network Server.

Gateways transfer data from Nodes to the central Network Server. They are fewer in number because a single gateway can support thousands of devices. Because the connection between the

Gateway and the Network Server is done by IP connections, packets need to be converted. Gateways act as bridges, converting RF packets to IP packets or vice-versa.

Network Server represents the edge of the presented system and gathers together data sent from Nodes. It can be represented by an Internet facing web service to which Gateways can connect through, for example, cellular networks [11].

2.3 LoRa Physical Layer

As stated before, LoRa represents the physical layer and it is a Semtech proprietary technology.

LoRa was designed such that it allows low-power, low-throughput and long-range communications. It uses 433 MHz, 868 MHz and 915 MHz Industrial, Scientific and Medical (ISM) unlicensed frequency bands, depending on the region in which it is deployed, being able to transmit, depending on the environment, over several kilometers. For Europe, the used band is 868 MHz.

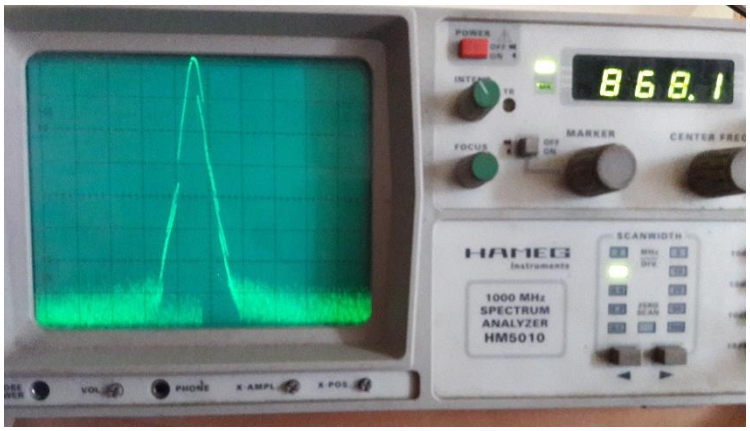

Figure 3.3 – LoRa frequency specter on 868.1 MHz

2.3.1 Overview of the Physical Layer

LoRa uses Chirp Spread Spectrum (CSS) modulation. This technique allows the signal to be modulated by chirp pulses (sinusoidal pulses which vary in frequency). Because of this variation, chirp-modulated signals improve resilience and robustness against multi-path interference and Doppler effect which is equivalent to frequency offset [6].

LoRa has many advantages such as:

- Thousands of devices can be connected per gateway, enabling high capacity networks;
- Long communication range - it goes up to 2-5 kilometers in urban areas and up to 15 kilometers in suburban areas; coverage is way greater in range than that of existing cellular networks or other IoT protocols;
- Operates with low power – battery lifetime is around 10 years;

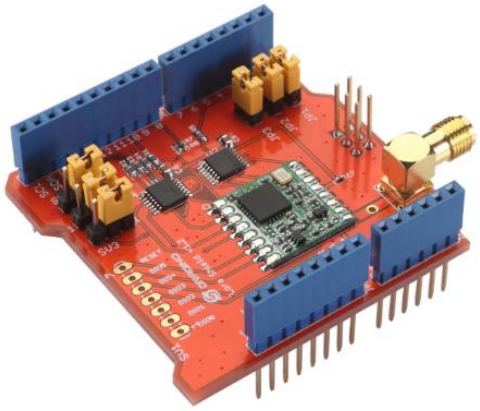

Figure 3.4 – Dragino LoRa Shield [16]

2.3.2 Parameters of the physical Layer

LoRa modulation have several parameters that can be customized: Bandwidth, Spreading Factor (SF) and Code Rate. All these parameters have an influence on the resistance to interference noise of the modulation, its effective bitrate and its ease of decoding.

Bandwidth – this is the most important parameter. A LoRa symbol is composed of 2^{SF} chirps, which cover the entire frequency band. It starts with a series of upward chirps. When the maximum frequency of the band is reached, the frequency wraps around, and the increase in frequency starts again from the minimum frequency.

The chirp rate depends only on bandwidth, being equal to this.

LoRa also includes a forward error correction code.

These parameters also influence decoder sensitivity. Generally speaking, an increase of bandwidth lowers the receiver sensitivity, whereas an increase of the spreading factor increases the receiver sensitivity. Decreasing the code rate helps reduce the Packet Error Rate (PER) in the presence of short bursts of interference.

2.4 END DEVICES

Within LoRaWAN Protocol, based on MAC layer, we can define three classes of operation: Class A, B and C. They have different modes of functioning and capabilities, but they all refer to bi-directional communication.

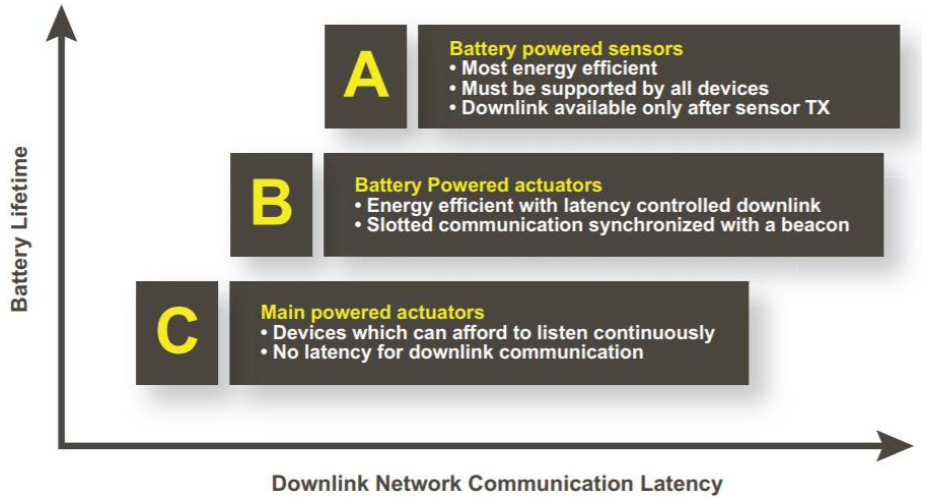

Figure 3.5 – Classes of devices within LoRaWAN [5]

Class A – It is the basic mode of operation being supported by all devices and also the class with the lowest power consumption.

They use pure ALOHA access for the uplink. This means that they will send randomly, at any time, uplink messages [6]. After an uplink message, the device will open two downlink receive windows. The recommended values for these widows are 1s and 2s, respectively. We have three situations as described in the figure below:

- 1st situation: server does not respond in any of the receive windows; in this case, the next opportunity will be after the next uplink.
- 2nd and 3rd situations: server can respond in one of the receive windows. However it should not use both of them; if the downlink traffic is received in the first window, the second is disabled.

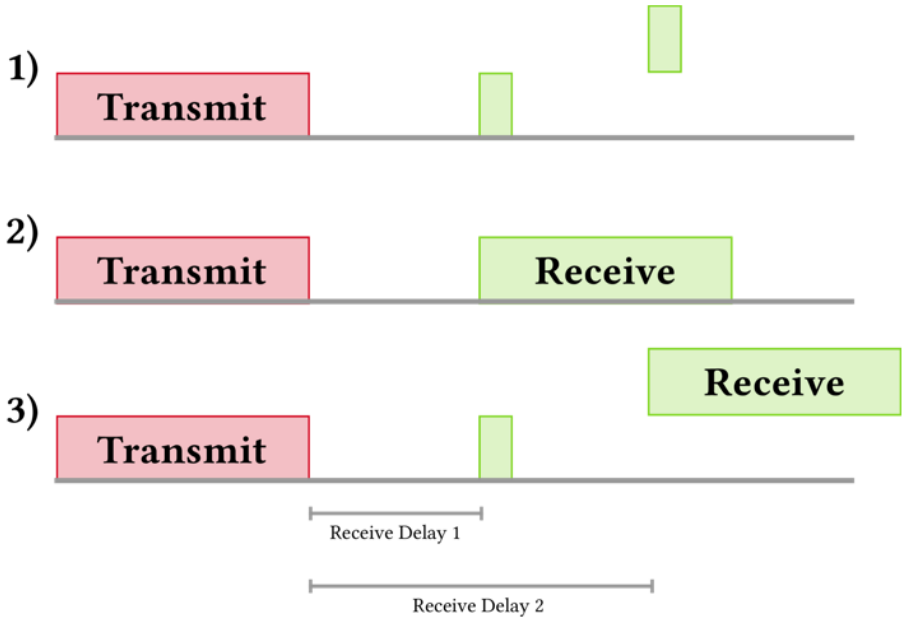

Figure 3.6 – Class A Receive Windows [10]

Moreover, class A is the only class that must be implemented in all end-devices.

Class B – When additional downlink traffic is needed, class B devices can be used. They come as an extension of class A devices because they transmit, periodically, beacon frames [6].

Usually, beacon frames are used, in IEEE 802.11, to transmit the presence of a wireless LAN and they contain all information about the network. In this case, the frames are sent by the gateway and allow, without the need of a prior successful uplink transmission, the schedule of receive windows for downlink traffic. Only class B and class C devices can receive them.

However, power consumption is higher than in the case of class A devices.

Class C – These types of devices are defined by the fact that they can receive frames continuously because they are always listening to the channel [6]. This means that, as shown in the figure below, the receive window is open, unless they transmit.

Because of this, within this class we have low-latency but they consume more energy.

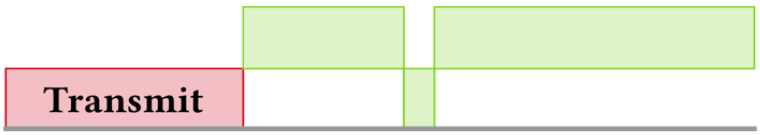

Figure 3.7 – Class C Receive Windows [10]

The devices can switch from one class to another because all the three classes can coexist in the same network. However, the gateways are not informed about the class that a device is part of because there is no specific message defined by LoRaWAN and this is up to the application.

2.5 SECURITY

From the beginning of IoT, threats of cyber-attacks have become a high concern. This is why, the issue of security is a topic more and more debated lately. Due to the fact that IoT connects more people and devices, attackers could take over data, cause malfunction or gain access to intellectual property, causing harm to larger groups in a relatively short time. So, in order to prevent system disruption, networks require high level of security. LoRaWAN offers, like many other protocols, signing and encryption for parts of LoRaWAN packets and ways of securing data when talking about connecting devices to LoRaWAN networks.

Security strategies to protect connections and data transfers should have two important characteristics: they should not be complex in order to be supported in IoT endpoints with a minimum additional demand on device power and, also, they should be inexpensive.

LoRaWAN provides security mechanisms that protect communications by mutual authentication. This is a way of ensuring that the device that connects to the network is registered and of the authenticity of the network that the device is joining. This implies that, both the device and the network have knowledge of AppKey which allow encryption and decryption of messages.

When talking about LoRaWAN security, we can also define two session keys, each of them with a length of 128 bits: network session key – NwkSKey and application session key – AppSKey, an extension of the security developed for IEEE 802.15.4 radio communication which is used [11].

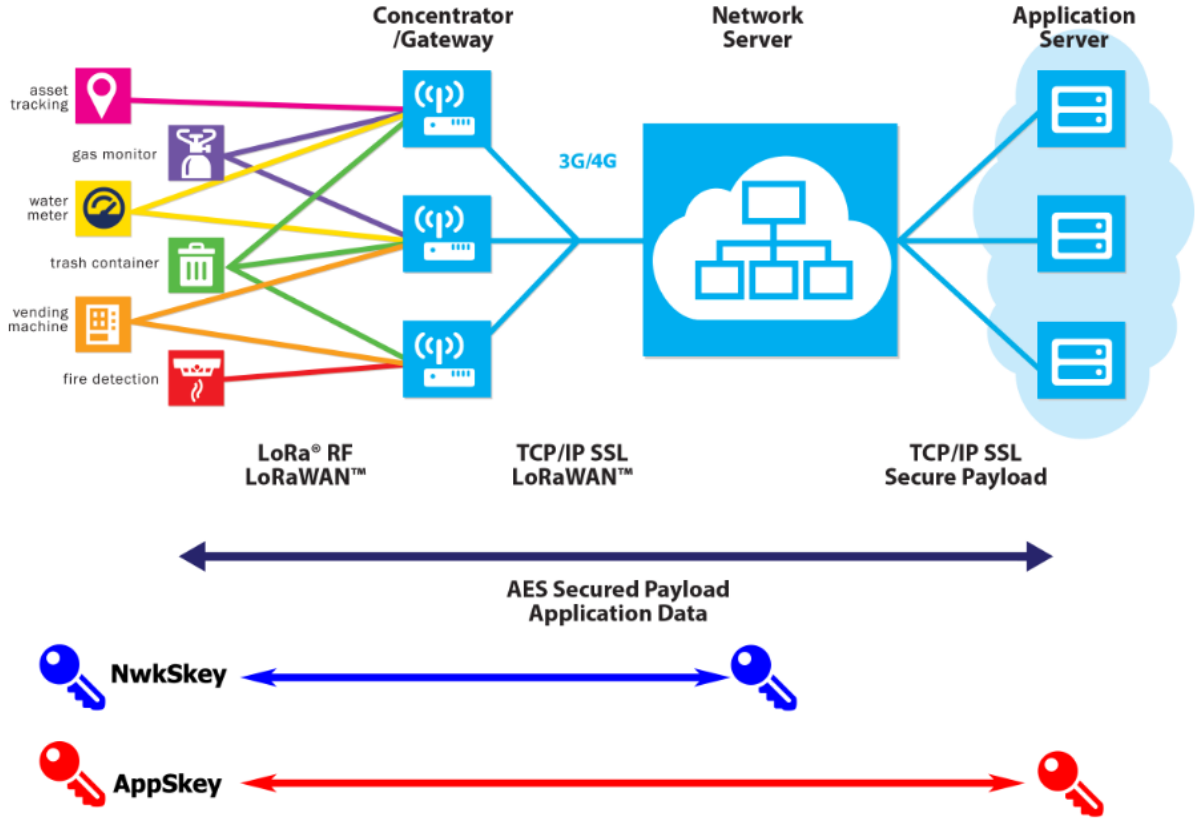

Figure 3.8 – Encryption keys in LoRaWAN networks [9]

As stated earlier, we can use them in the following way:

- Network session key (NwkSKey) – used between Node and Network, guaranteeing the message integrity;
- Application session key (AppSKey) – used between Node and Application Server for payload encryption and decryption.

Before communicating on a LoRaWAN network, devices must be activated. There are two ways of doing this: Over The Air Activation (OTAA) or Activation By Personalization (ABP). However, in both cases, before connecting, not only the device, but also the network must demonstrate they have the security keys.

Packets that are exchanged in LoRaWAN networks contain a MAC header, frame header or counter, the payload and a message integrity code (MIC) which is generated using NwkSKey.

As it can be seen in Figure 3.8, application payload is encrypted using Advanced Encryption Standard in counter mode (AES-CTR). The frame header is included as part of the LoRaWAN packet. This prevents attackers from gaining access by replaying messages. The counter needs to be managed correctly so no sequences are repeated or the counter is not reset by forcing the node to reconnect to the network [11].

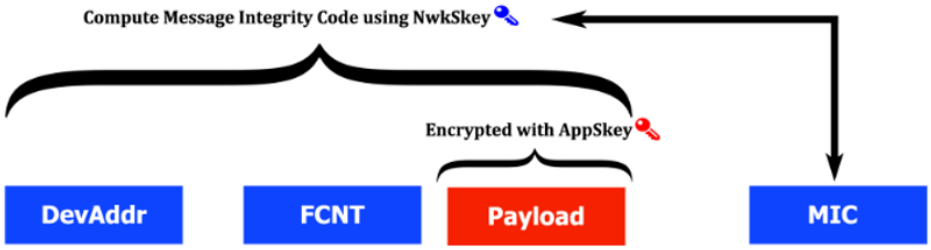

Figure 3.9 – Encryption of LoRaWAN packets to prevent interception and attacks [9]

2.6 Connection to LoRaWAN networks

LoRaWAN is usually used for communication between devices and gateways. All other communication is done by IP networks.

Devices can connect to a network in the following ways:

2.6.1 Over-the-Air-Activation (OTAA)

This is the preferred and frequently used way to connect devices with a network because it is also the most secure. In this case, devices perform a join-procedure with the network. During this, DevAddr is assigned dynamically and security keys are negotiated.

The only disadvantage is that it adds a layer of complexity to the process. In order to connect OTAA, devices need a DevEUI (it is a 64-bit end-device identifier, assigned by chip manufacturer, globally unique), an AppEUI (it is a 64-bit unique application key and it identifies the application to which the device will connect) and an AppKey (128-bit key that is shared between the end-device and the network) [7].

During OTAA, after an authorized device connects to the network and the encryption keys are exchanged with the network core, network server sends the following information to the end-device:

- Device address (DevAddr) – this is the logical address used for communication;
- Application Session Key (AppSKey) – it is an encryption key between the device and the operator via application;

- Network session key (NwkSKey) – it is an encryption key between the device and the operator.

2.6.2 Activation by Personalization (ABP)

There are cases when OTAA can be skipped. This is done when devices are manually registered and keys are directly obtained. However, this procedure weakens security because keys are practically preconfigured and so they can be easily stolen.

2.6.3 Default Activation for Generic Devices

This is a special case in which devices use default keys which are supported by all network operators. This is what we call a generic device. They mainly use ABP. Packets sent from these devices are usually not encrypted so they lack security.

They use globally-known NwkSKey and AppSKey and, for data encryption, AppSKey. Moreover, many attributes such as DevAddr, length and time at which the packet was sent, signal strength and other gateway information are accessible to public [7].

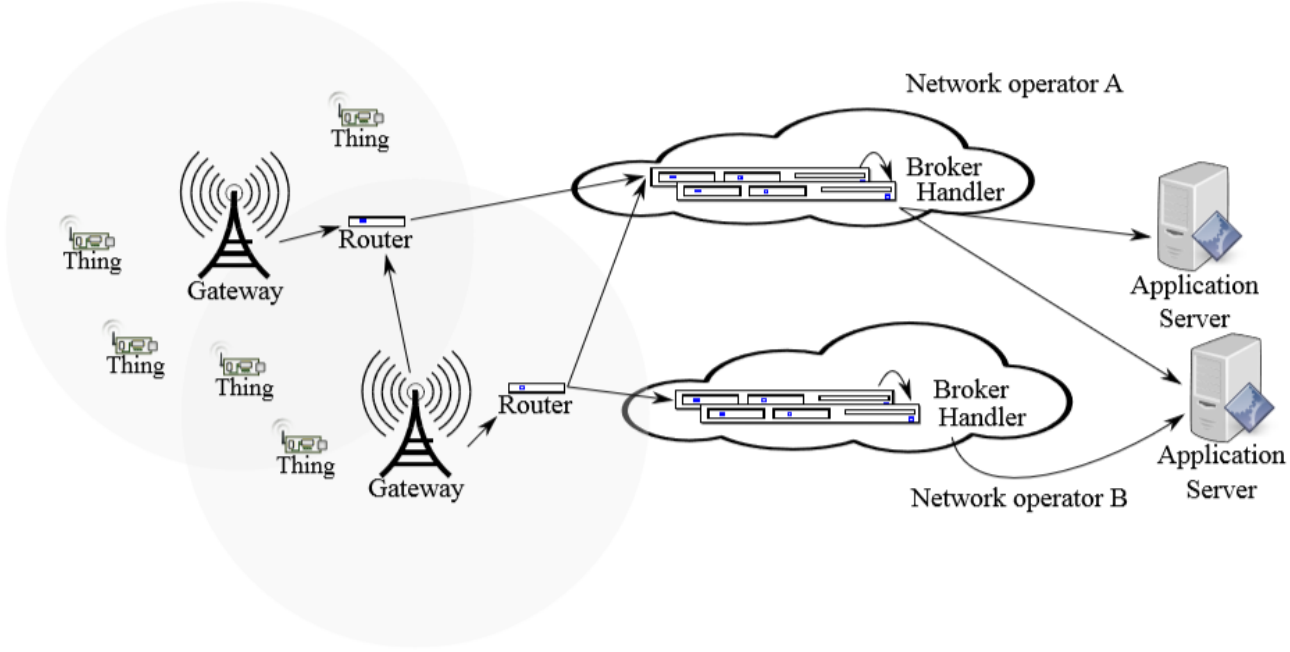

Figure 3.10 – LoRaWAN class A network [7]

CHAPTER 3

DRONES – HARDWARE TECHNOLOGIES

3.1 LEHMANN AVIATION LA300

LA300 is an UAV fully automatic, produced by Lehmann Aviation, with application in agriculture, high precision mapping and constructions or mining. It is a small dimension drone, which can take photos at low altitudes of 50-100 meters using an additional camera such as GoPro (Hero 3 or latest models), Canon or MicaSense.

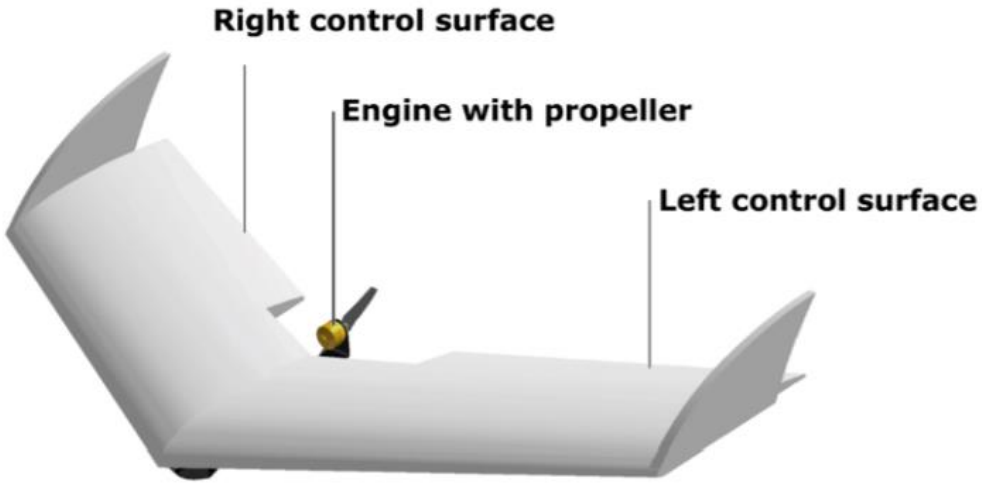

Figure 4.1 – Aircraft (wing assembled with the main Electronic part) [12]

The drone was designed by its manufacturers in order to be used for agricultural lands real-time surveillance, based on the analysis of the images taken with the camera provided. However, the purpose can be easily extended and images can also be taken from different locations that need to be supervised to analyze possible dangers or threats.

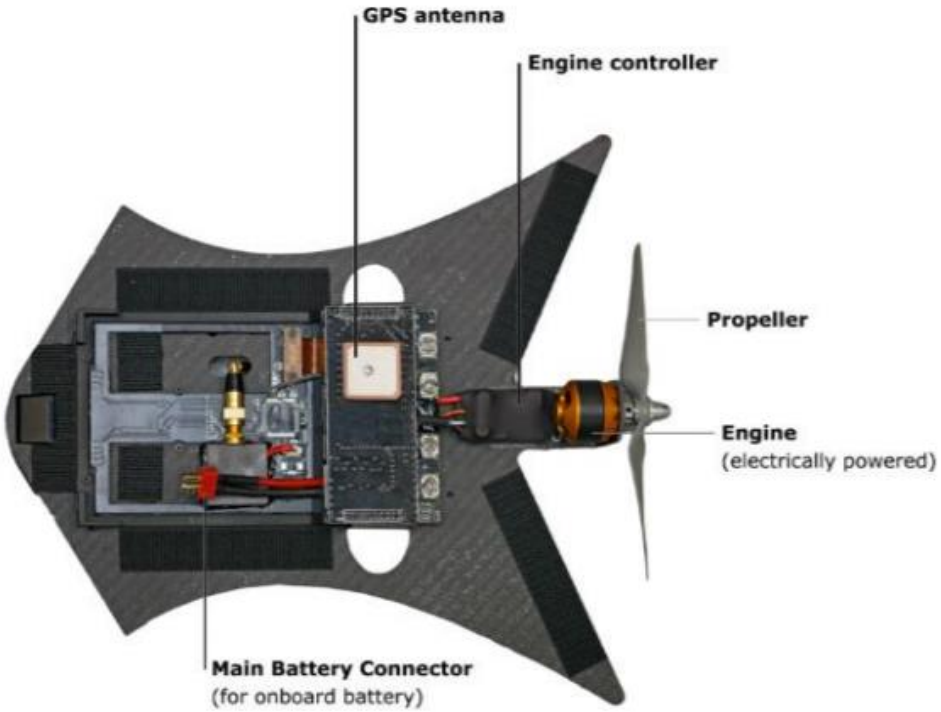

Figure 4.2 – Aircraft Main Electronic part details [12]

As presented in Figure 4.2, the electronic components of LA300 drone are the following:

- GPS antenna
- Engine controller
- LiPo battery
- Electrically powered engine
- Wi-Fi SNIC SN8200 interface

The built-in GPS antenna gives the drone a radionavigation system to find the geolocation of the points they need through the satellites.

Battery is Lithium Polymer (LiPo) with a total capacity of 2600 mAh and a nominal voltage of 11.1V. This is enough to assure a 30 minutes flight or to cover a distance around 15km.

The SN8200 wireless interface is a Murata controller that is specifically designed to support wireless communications. Through this controller, the ground control station can communicate with the UAV and send it the coordinates of the trajectory. The configuration is done in the initial phase of the setup and then, after the UAV lands, we can read the actual GPS coordinates of the flight.

The engine controller reads the settings received on the wireless interface and operates the electric engine so that the drones will fly over the desired trajectory.

It also has the advantage of being easily maneuvered, being launched without a special frame or training. The route can be scheduled with the application with which the drones are delivered, but we also have the possibility to develop an application.

Drone settings are transmitted to the UART interface of the SN8200 wireless module in the format of a JavaScript Object Notation (JSON), the received and interpreted settings being then transmitted by the wireless interface on the output ports to the controller operating the UAV's engine.

All of the features presented above are advantages that recommends Lehmann Aviation L300 UAV for use in military applications: high battery capacity (in relation to other UAVs), fast charging time, the possibility of using a larger number of wireless protocols (by replacing the SNIC SN8200 controller or adding independent modules), global satellite coordination, security transmissions, etc.

Also, the fact that the drone has the ability to acquire images gives us the possibility to synchronize with the terrestrial one, or to provide a complete overview of the overlay surfaces, all of these being practical implementations of the following phases of the project.

3.2 PHANTOMX HEXAPOD ROBOT

3.2.1 Hexapod Robots

Firstly, I have to define what the term Hexapod Robot refers to. This kind of robot is mechanical device which motion is based on its six legs. The main advantage is that the robot has more flexibility and stability than other types with two, three or four legs. Furthermore, its behavior is more complex. Because not all the legs needed for movement or stability, the others can be used to lift objects, to manipulate payloads or to target the robot to certain areas.

In figure 4.1, it is present *PhantomX AX Hexapod Mark II* robot.

Figure 4.3 – PhantomX AX Hexapod Mark II [13]

This types of robots make a good target for the use case defined within this paper. Their primary target was to be used for testing biological theories related to the movement of insects, engine control and neurobiology. However, such robots can be successfully used in discovery or research missions in places hard to reach people (e.g. in areas devastated by earthquakes or other natural disasters or in military missions).

There are different types of Hexapod robots. The design may vary from the point of view of the arrangement of their legs. Most of them, being inspired by the anatomy of insects, have their feet symmetrically distributed. Moreover, their feet have two to six points of freedom.

The movements of a hexapod robot are controlled by the types of steps it can make:

- Crawl – Single-leg movement
- Alternating tripod – three legs on the ground at a time
- Quadruped

In addition to these standard hexapod control steps, motion is also influenced by the environment and depends on the type of surface that the robot walks on.

Other important factor for hexapod robots is the man-robot interaction. Human control over the hexapod varies between different levels of autonomy.

Man can have absolute control over the movements of the robot by programming it; however, the robot can be programmed to take more complex decisions based on the commands it receives, in order to meet the requirements. There are also autonomous robots that can work for a long time without interacting with humans, reacting on the basis of well-defined models.

3.2.2 PhantomX AX Hexapod Mark II Hardware Structure

PhantomX hexapod is fairly complex robot, developed by Vanadium Labs. It is open source, which means that it completely fits the purpose of this paper.

The heart of the PhantomX robot is the Arbotix Robocontroller, which works on a kinematics and reverse motion system, commanding the Dynamixel AX-12 network for leg positioning. Arbotix accepts navigation commands via the Control Protocol, a simple serial protocol that allows for a proportional control of the robot movement. The robot controller can communicate wireless with a manual command system via this protocol and with a pair of XBee Wireless Modules. The same protocol can also be used for communication between a PC and the hexapod, using the XBee USB

interface. PhantomX hexapod control can be done using any programming language that is capable of transmitting data through a serial port.

Dynamixel AX-12A Robot Actuator

The AX 12-A actuator is one of the most advanced and it has become a standard actuator for the next generation of robotics. Among its features, the most important are the ability to track its speed, temperature, position, voltage and load. Moreover, the control algorithm used to maintain the correct position of the actuator can be adjusted individually for each motor, allowing the control of the speed and strength of the motor's response.

The Dynamixel AX-12A Robot Actuator is presented in Figure 4.4.

Figure 4.4 – Dynamixel AX-12A Robot Actuator [13]

Hardware specifications of the actuator are [13]:

- ✓ Weight: 53.5g
- ✓ Size: 32 mm x 50 mm x 40 mm
- ✓ Operating temperature: -5° C ~ +70° C
- ✓ Supply voltage: 9 ~ 12V (recommended voltage is 11.1V)
- ✓ Resolution: 0.29°
- ✓ Protocol type: half-duplex, serial asyncron
- ✓ Physical connection: TTL
- ✓ Communication speed: 7343bps ~ 1Mpbs

CHAPTER 4

CASE STUDIES FOR AERIAL AND TERRESTRIAL DRONES AND OPTIMIZED COMMUNICATION – CONSIDERING THE PARTICULARITIES OF BOTH ASSEMBLES

4.1 INTRODUCTION

This study aims to be a viable use case which analyze and demonstrates the advantages and limitations of using LoRa and LoRaWAN protocol within an application which proposes to transmit real time GPS coordinates between an UAV and a server placed on the ground.

In order to develope what I have proposed, the following components will be used:

- ✓ Arduino Uno
- ✓ Seeduino Cloud – Arduino Yun compatible openWRT controller
- ✓ Raspberry PI 3
- ✓ LoRa Dragino shield
- ✓ LoRa GPS HAT

- ✓ Lehmann Aviation LA300
- ✓ Terrestrial Drone

4.2 DEVELOPMENT BORDS

4.2.1 Arduino Uno

Arduino Uno is one of the most popular development boards used in the development of electronic projects because it is simple to configure, has a fairly large number of pins and it is compatible with a large number of shields that allow the addition of various functionalities.

Technical specification [14]:

- ✓ Microcontroller: ATmega328P
- ✓ Flash memory: 32 KB
- ✓ SRAM: 2 KB
- ✓ EEPROM: 1 KB
- ✓ Clock speed: 16 MHz
- ✓ 14 digital pins: they can be used both as input or output. In addition, some of them have specialized functions:
  - Pins 0 (RX), 1 (TX). These pins are used to receive (RX) and transmit (TX) TTL serial data.
  - Pins 2, 3: External Interrupts. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.
  - Pins 3, 5, 6, 9, 10, 11: 8-bit PWM
  - SPI: Pins 10(SS), 11(MOSI), 12(MISO), 13(SCK)
  - Pin 13: LED; there is a built-in LED connected to the digital Pin 13. When the pin is set on HIGH value, the LED is on, when the pin is LOW, the LED is off.
- ✓ 6 analog pins: they are used as analog inputs

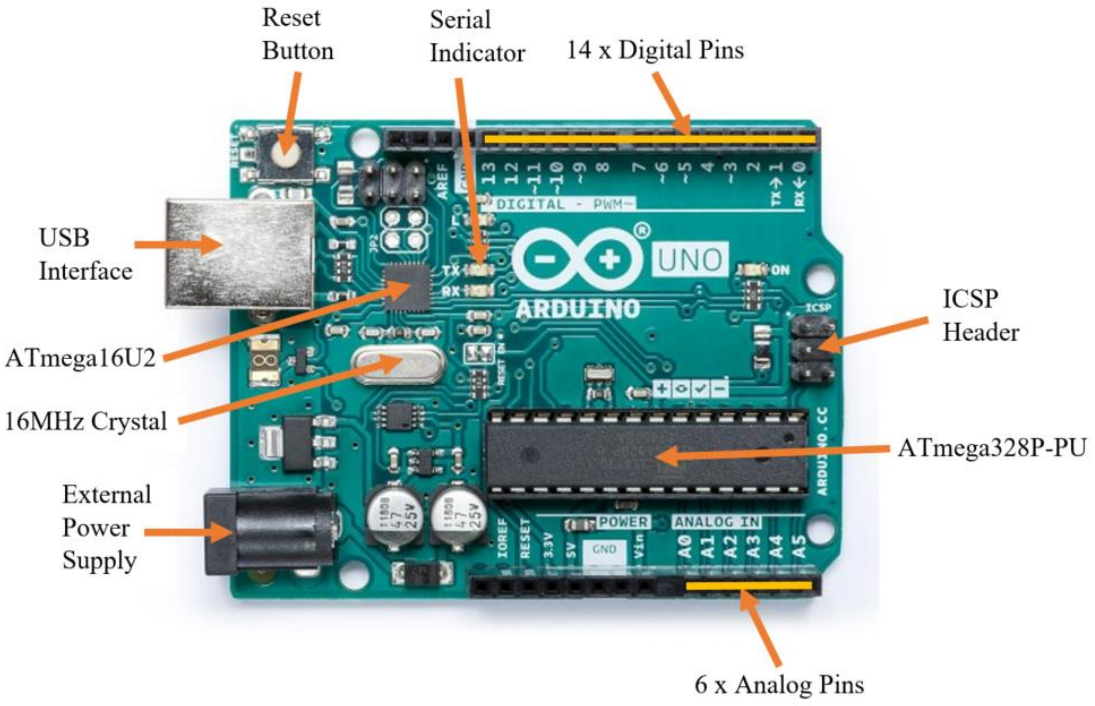

Figure 5.1 – Arduino Uno architecture [14]

From the list of advantages of using Arduino in my project I'm underlying the fact that its development environment comes with multiple predefined functions included in the basic library. All that has to be done in order to use those functions is including the appropriate files in the project. Conveniently, this allowed me to test several shields and external components in order to select the most appropriate ones for my project and my requirements, without having to write additional code for each and every one of them.

Also, this development board, unless other of its kind, offers the possibly to load and save a sequence of instructions in the flash memory, without having to re-upload it after every reboot or lack of power.

There are only a few disadvantages that can be named when talking about the Arduino Uno development board – the relatively low frequency of its microprocessor and its limited flash and EEPROM memory are the most important ones. Another technical detail that doesn't comply very well with the concept of Internet of Things is the fact that the board doesn't come with an Ethernet adapter included, meaning that a separate shield must be acquired and installed, with the appropriate libraries having to be included in any developed application. However, neither one of those can be classified as a critical issue as they can be resolved without too much trouble and without interfering with the application that is being developed.

4.2.2 Seeeduino Cloud - Arduino Yun compatible openWRT controller

Seeeduino Cloud is a microcontroller board based on both Atmega24u4 and Atheros AR9331.

It is very similar with Arduino Yun, the only difference being the operating system which is OpenWRT, a Linux operating system which target embedded systems.

It has a normal Arduino interface, but also a built-in Ethernet interface, Wi-Fi and an USB-A port which makes it very suitable for those prototype design that needs network connection and mass storage.

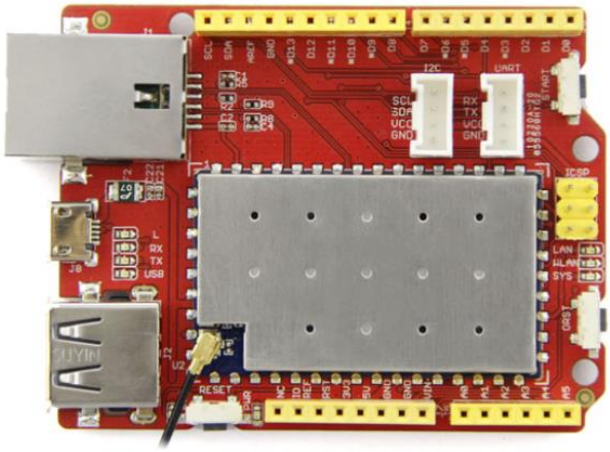

Figure 5.2 – Seeeduino Cloud - Arduino Yun compatible openWRT controller [15]

Technical specifications [15]:

- ✓ AVR Arduino microcontroller:
- ✓ Microcontroller: ATmega32u4
- ✓ Flash memory: 32 kB (of which 4 kB used by bootloader)
- ✓ SRAM: 2.5 kB

- ✓ EEPROM: 1 kB
- ✓ Clock speed: 16 MHz
- ✓ 20 digital pins: they can be used both as input or output. In addition, some of them have specialized functions:
  - 7 pins can be used as PWN
  - 12 pins can also be used as analog 6 analog pins: they are used as analog inputs

Microprocessor:

- ✓ Processor: Atheros AR9331
- ✓ Ethernet: 802.3 10/100Mbit/s
- ✓ Wi-Fi: 802.11b/g/n 2.4 GHz
- ✓ RAM: 64 MB
- ✓ Flash memory: 16 MB
- ✓ SRAM: 2.5 KB
- ✓ EEPROM: 1 KB
- ✓ Clock speed: 400 MHz
- ✓ OS: Open SourceWrt

4.3 LoRa Dragino Shield

LoRa GPS Shield presented in Figure 5.3 and used within my application is an expansion board of LoRa/GPS which can be used with Arduino.

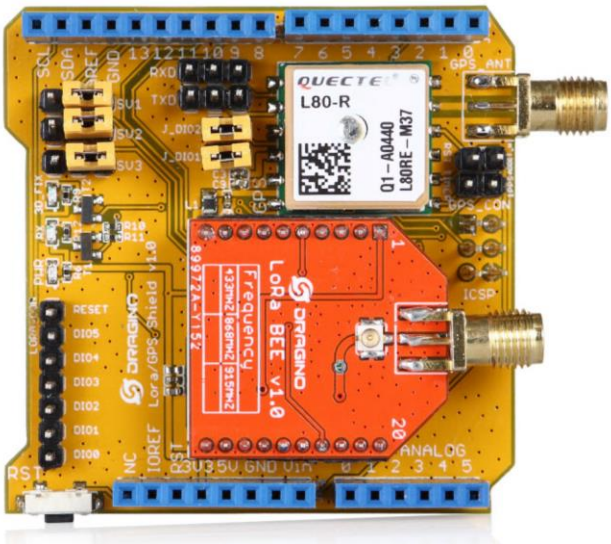

Figure 5.3 – LoRa GPS Shield [16]

The LoRa part of the LoRa GPS Shield is based on the SX1276/SX1278 transceiver. The transceivers of the shield feature the LoRa long range modem that provides ultra-long range spread spectrum communication and high interference immunity whilst minimising current consumption.

In the GPS part, the add on L80 GPS is designed for applications that use a GPS connected via the serial ports to the Arduino, such as timing applications or general applications that require GPS information. This GPS module can calculate and predict orbits automatically using the ephemeris

data (up to 3 days) stored in internal flash memory, so the shield can fix position quickly even at indoor signal levels with low power consumption.

Some of its features are listed below [16]:

- ✓ Frequency Band: 868 MHz/433 MHz/915 MHz
- ✓ Low power consumption
- ✓ Compatible with Arduino Leonardo, Uno, Mega,Due etc.
- ✓ FSK, GFSK, MSK, GMSK, LoRa and OOK modulation
- ✓ Preamble detection
- ✓ Baud rate configurable
- ✓ Built-in temperature sensor and low battery indicator
- ✓ GPS automatic switching between internal patch antenna and external active antenna
- ✓ Support SDK command
- ✓ Built-in LNA for better sensitivity
- ✓ EASY, advanced AGPS technology without external memory
- ✓ AlwaysLocate, an intelligent controller of periodic mode
- ✓ GPS FLP mode, about 50% power consumption of normal mode
- ✓ GPS support short circuit protection and antenna detection

4.4 LoRa GPS HAT for Raspberry Pi

LoRa GPS HAT is based on SX1276/SX1278 transceiver, being similar to LoRa Dragino shield used for end devices. The difference is that it is specifically designed to work with Raspberry Pi.

The L80 GPS is designed, this time, to connect via serial ports to Raspberry Pi.

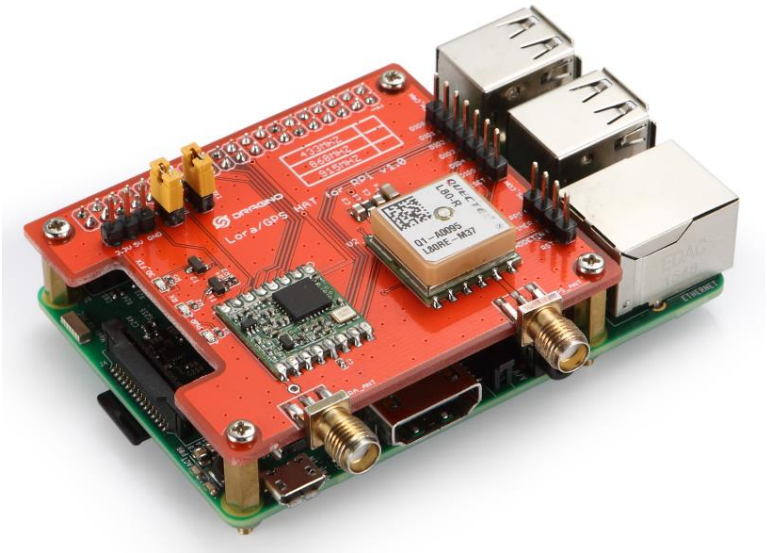

Figure 5.4 – LoRa GPS HAT on top of a Raspberry Pi 3 [20]

Some of its features are listed below [20]:

- ✓ Frequency Band: 868 MHz/433 MHz/915 MHz
- ✓ Low power consumption
- ✓ Compatible with Raspberry Pi 2 Model B/Raspberry Pi 3

- ✓ FSK, GFSK, MSK, GMSK, LoRa and OOK modulation
- ✓ Preamble detection
- ✓ Baud rate configurable
- ✓ Built-in temperature sensor and low battery indicator

4.5 RASPBERRY PI 3

As a family, the Raspberry PI mini-computers come with an integrated set of CPU, GPU and RAM memory, alongside a large number of interfaces and connectors which allow for many external components and sensors to be connected. Some of those components are essentials for any kind of application that is developed – as Raspberry PI 3 is basically a computer it needs (at least for initialization) a mouse, a keyboard and an external monitor. After the basic configuration is uploaded on the computer, a SSH connection can be configured in order to have remote access.

Raspberry PI 3 is the third generation of computers from this family and can be very efficiently used for a large number of application, surpassing by far the previous two models available on the market. While keeping the same form-factor as the previous released boards, this generation has a CPU that is ten times more faster the Raspberry PI 2, a bigger RAM memory and an integrated connection for both Wireless LAN and Bluetooth.

As general specifications, I must mention [17]:

- ✓ 1.2 GHz Quad-Core ARM-Cortex-A53 CPU
- ✓ Dual Core VideoCore IV GPU
- ✓ 1GB LPDDR2 RAM memory
- ✓ The operating system is loaded from an SD card – the OS can be represented by either a Linux distribution or Windows 10 IoT
- ✓ Board size: 85 x 56 x 17 mm
- ✓ Power requirements: micro USB providing at least 1A (max 2.5A) and 5V

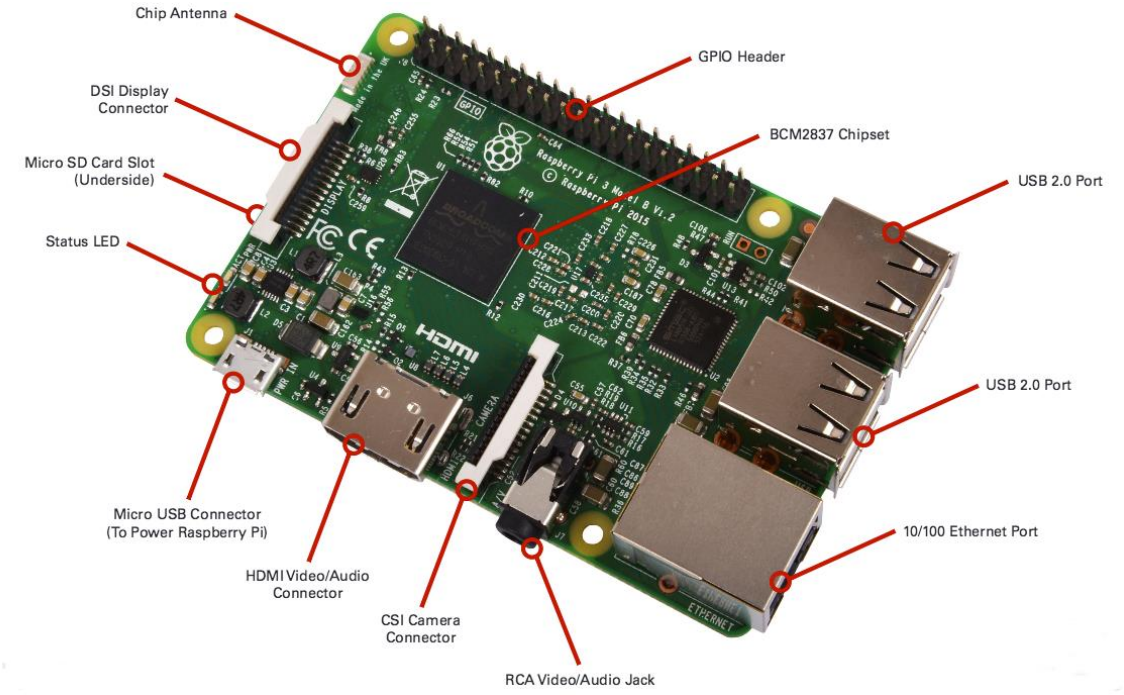

Figure 5.5 – Raspberry PI 3 architecture [17]

4.6 LEHMANN LA300 UAV IMPROVEMENTS

The Lehmann LA300 UAV used within this application is described in *Chapter 4* in its initial configuration.

During multiple tests inside and outside the laboratory, I encountered problems with drone stabilization during the initial setup and multiple attempts of take-off.

After an analysis of the possible causes of this behavior, I realized that the issue was the right elevon, which was not functioning correctly.

The elevon is a control surface of the aircraft. Elevons are mainly used on tailless aircraft such as flying wings (as in the case of our UAV) and they combine the functions of the aileron, which is used for roll control, and the elevator, used for pitch control.

Both servomechanisms were tested separately, to eliminate the possibility of a mechanical failure, and proved working fine. The only possible cause that remained was the autopilot.

Because the autopilot of the UAV was difficult to be eliminated and tested piece by piece, the best way seemed to replace everything, including the controller and the software.

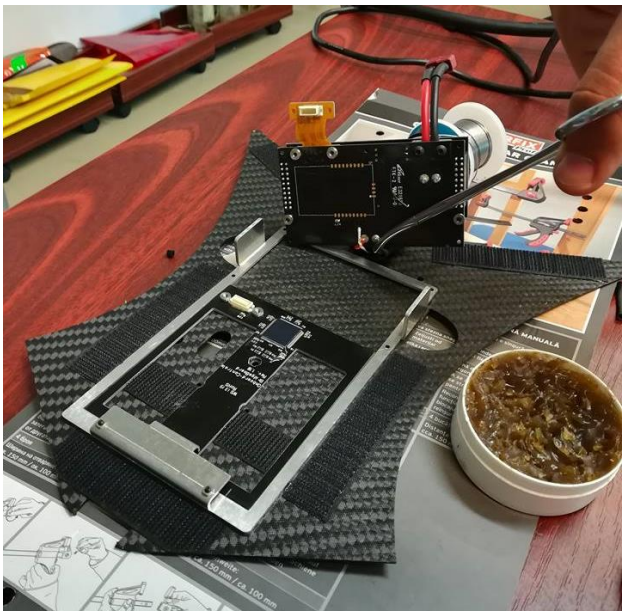

Figure 5.6 – Removing the electrical part of the UAV

After checking the space available for the components that will replace original ones and calculating the weight, in order not to exceed maximum weight of 950 grams, fully equipped, I analyzed the options and came with what I thought was the best implantation, replacing all control and command components of the original UAV.

The parts that were kept were: the electrical motor, the servos and the battery. Besides this, the UAV has been equipped with a Pixracer v1.0, a X8R radio receiver, an Ublox GPS, a pair of 433 MHz radio modems for telemetry, a MC5983 3-Axis IIC/SPI Digital Compass magnetometer, a buzzer and a safety switch.

Figure 5.7 – Different solutions that were analyzed for replacing the UAV components

The software that is used is called Ardupilot. This is an open source autopilot software which can control many vehicle systems, from airplanes, multirotors, to boats and helicopters.

Mission Planner is a ground station application for Ardupilot. To plan missions, this tool is a necessity. This allows multiple actions such as:

- ✓ Loading the software into the autopilot
- ✓ Setup, configuration and tuning of the UAV for optimum performance
- ✓ Planning, saving and loading autonomous missions into autopilot using waypoints on Google or other maps
- ✓ Downloading logs created by autopilot
- ✓ Interfacing with a PC flight simulator to create an UAV simulator

Figure 5.8 – Mission Planner – Flight simulator

After fixing the flight controller, inside the UAV, I started connecting the rest of the components, placing them such that the center of gravity of the vehicle remains unchanged. This is important for a flying wing as it should not be heavier neither in front, nor in the back.

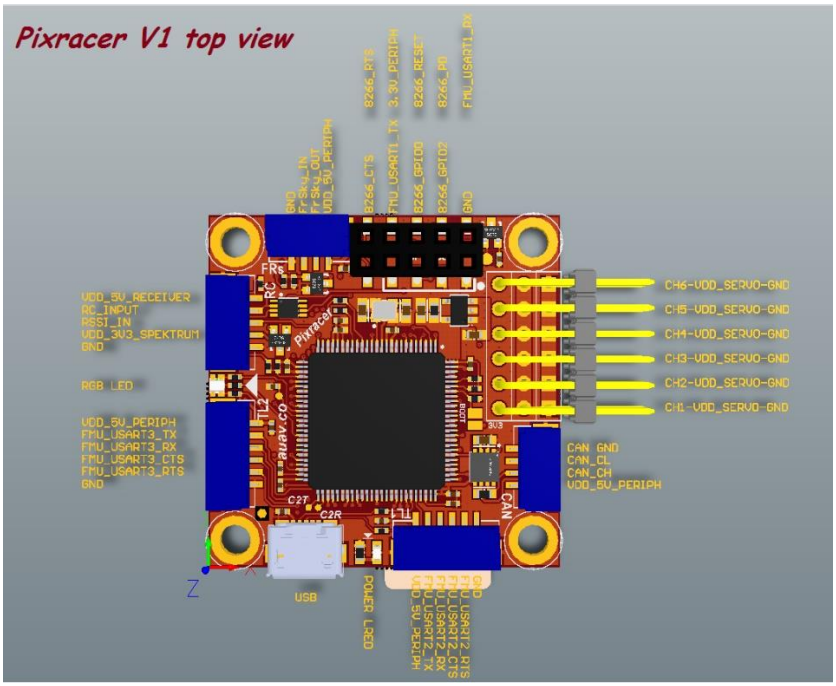

Figure 5.9 – Pixracer v1.0 top view [19]

Connections with other components were done as in the above schematic.

The telemetry radio allows the UAV, using MAVLink protocol, to communicate with the ground station from the air. Several actions can be done: monitor the status of the UAV while in operation, record telemetry logs, or view and analyze the previously recorded logs.

The telemetry chosen for this application works on 433 MHz, being the perfect choice for this, because there is no interference with LoRa, which works on 868 MHz.

Telemetry is formed by two interchangeable air and ground modules, one being placed on the UAV and the other, connected to a laptop.

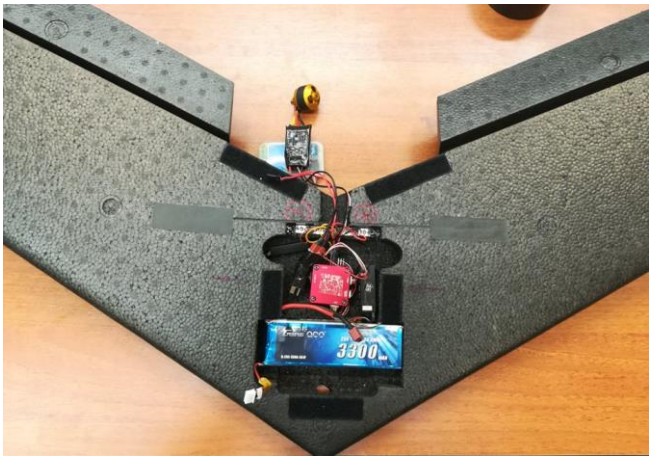

Figure 5.10 – Assambled components

To have the possibility to control the UAV also with a Taranis X9D Transmitter, I connected to the controller a X8R receiver.

A component that was not present in the initial configuration of the UAV, but it is useful for signaling different events is the buzzer. It is used to play sounds such as Arming/Disarming buzz or Lost Copter Alarm.

A safety switch is used to enable or disable the outputs to motors and servos.

Figure 5.11 – Fully equipped UAV with attached video camera and Taranis X9D Transmitter

After the setup was ready, to plan a flight, several steps must be fulfilled [18]:

- ✓ Mission Planner must be connected to the autopilot to have the possibility to control the UAV and receive telemetry
- ✓ Hardware configuration must be done:
  - Choose frame type
  - Calibrate the compass; to do this, the vehicle must be hold in the air and rotated in such way that each side points down towards the earth
  - Calibrate the Radio Control Transmitter; this is done by moving each switch or stick through its full range
  - Calibrate the Accelerometer; this can be accomplished by placing the vehicle in each of the indicated positions: level, on right side, left side, nose down, nose up and on its back
  - Configuration of the Flight Modes; there is a mapping between switch position and flight mode which is set in Mission Planner Flight Mode screen
  - Calibrate the Electronic Speed Controller; Electronic speed controllers are responsible for spinning the engines at the speed requested by the autopilot
- ✓ Mission must be planned. This is done using waypoints. For my experimental flights I used a Take off point, a Return To Launch point and some Waypoints in between.

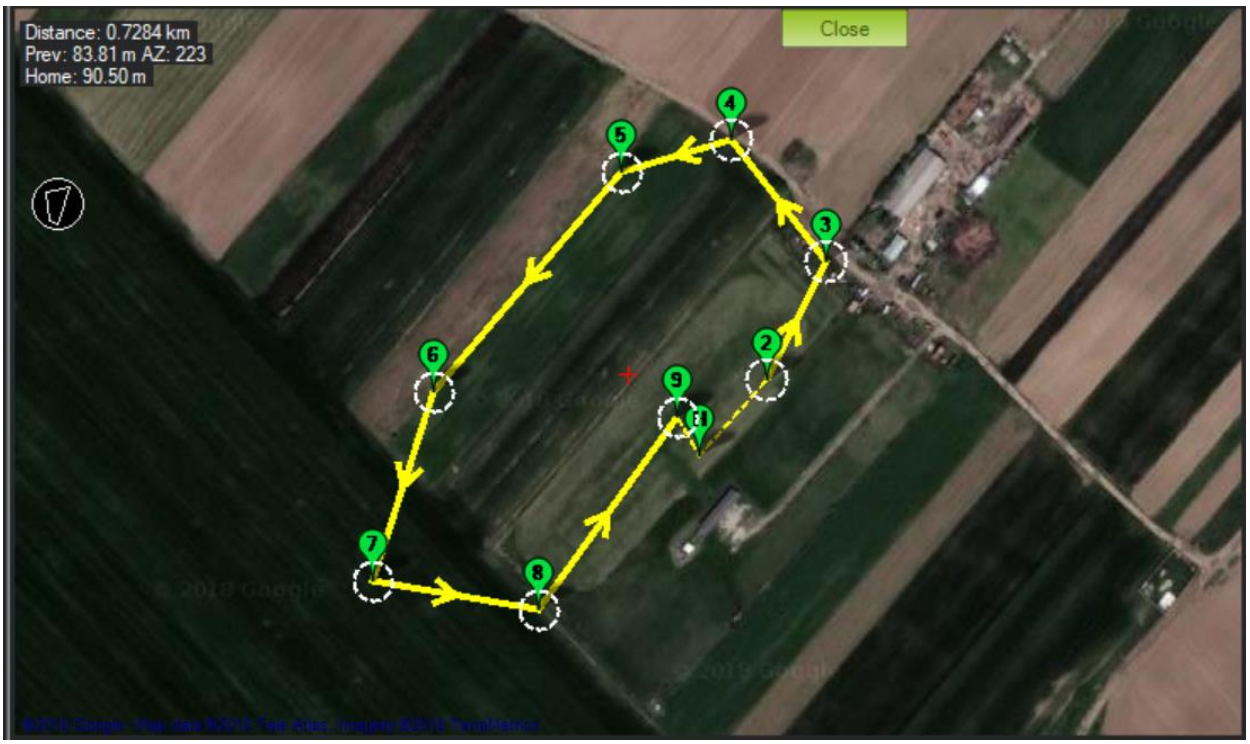

Figure 5.12 – Planned Mission

The presented route contains the take off point marked on the map with H (Home), 8 waypoints that the UAV has to follow during the flight marked on the map in order of 1 to 8, and the landing point marked with 9. If we set the UAV to land at the same place it has taken off, the two points, take off and land may be overlapped. To exemplify, the points were chosen differently but rather close enough so that the UAV can be recovered by the operator without the need to move between the take-off and landing.

The waypoints' type and parameters can be visualisez in a table like the one presented below, I Figure 5.13.

Waypoints

WP Radius 10 | Loiter Radius 45 | Default Alt 100 | Relative | Verify Height | Add Below | Alt Warn 0

	Command	Pitch/ Angl						Alt	Delete	Up	Down	Grad %	Angle	Dist	AZ
1	TAKEOFF	15	0	0	0	0	0	2	X			0	0	0	0
2	WAYPOINT	0	0	0	0	44.2651593	26.0342020	100	X			197.8	63.2	112.1	41
3	WAYPOINT	0	0	0	0	44.2656971	26.0345882	100	X			0.0	0.0	67.2	27
4	WAYPOINT	0	0	0	0	44.2662579	26.0339874	100	X			0.0	0.0	78.6	323
5	WAYPOINT	0	0	0	0	44.2661081	26.0332900	100	X			0.0	0.0	58.0	253
6	WAYPOINT	0	0	0	0	44.2651055	26.0320884	100	X			0.0	0.0	146.9	221
7	WAYPOINT	0	0	0	0	44.2642488	26.0317075	100	X			0.0	0.0	100.0	198
8	WAYPOINT	0	0	0	0	44.2641220	26.0327536	100	X			0.0	0.0	84.5	100
9	WAYPOINT	0	0	0	0	44.2649941	26.0336387	100	X			0.0	0.0	119.9	36
10	RETURN_TO_LAUNCH	0	0	0	0	44.2649595	26.0334939	100	X			0.0	0.0	99.7	27

Figure 5.13 – Waypoints

The flight path described by the UAV, taken from logs, can be seen in Figure 5.14. The preliminary tests and flights took place at Aeropower near Adunații-Copăceni, Romania.

Figure 5.14 – The path that describes the UAV route at Aeropower near Adunații-Copăceni, Romania

4.7 LoRaWAN Communication using an IoT Platform

The initial setup was thought using The Things Network in the idea that we take the data directly from there using Message Queuing Telemetry Transport (MQTT) protocol.

The Things Network is a platform for IoT. It uses LoRaWAN, allowing end-devices to connect to the internet without using 3G, 4G or Wi-Fi.

TTN uses MQTT to publish device activations and messages and it also allows the user to publish a message for a specific device in response.

MQTT is a messaging protocol. It fits perfectly all the requirements of IoT because of its design which ensure reliability, assurance of delivery and tries to minimize device resource requirements and network bandwidth.

In order to send data over to TTN, it is necessary to have an end device (also known as LoRa node) and a gatway to which the device connects and forward data to the IoT platform as presented below.

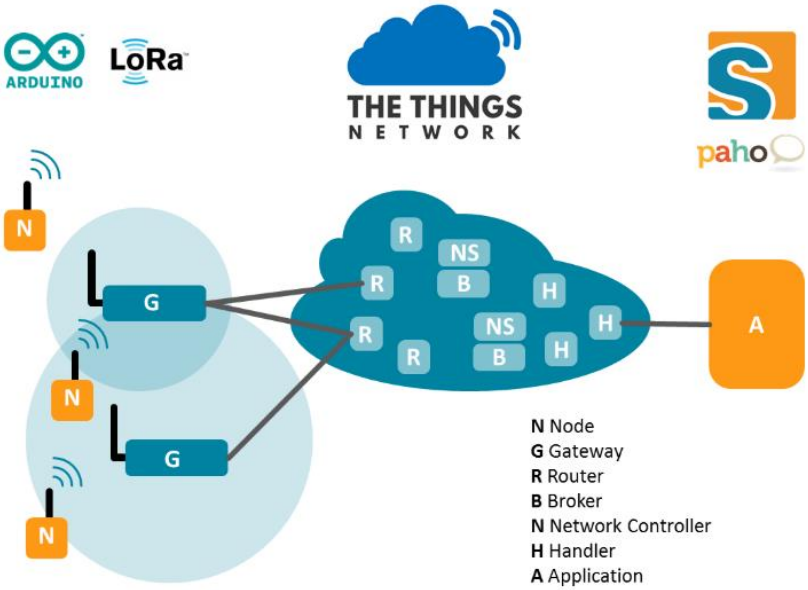

Figure 5.15 – Network example using TTN [10]

For all the above to be implemented, in the absence of a LoRaWAN network in the area where the study took place (Bucharest), I built a LoRaWAN gateway using a Raspberry PI 3 and a LoRa HAT shield for Raspberry PI.

For functional reasons, this chipset only allows the use of a single frequency from the standard. I chose for this 868.1 MHz.

The resulted gateway listens to on this frequency and sends to the TTN server the received information encrypted in a double way: at application level and at network level.

To deploy the gateway, I used a template from GitHub where I customized the IP address of the server, the coordinates where the LoRaWAN gateway is placed and the initialization of the chipset placed on the Dragino shield, LoRa SX1276.

This represented a documentation work to understand the program and adapt it to the manufacturer's specifications, the library having only a previous edition (SX1272) from the same manufacturer, but with slight differences from the chipset used by me.

Next, the gateway device must be enrolled on TTN, using the specifications of the cloud application.

Also, the nodes created for communication and reading GPS coordinates must be enrolled. The process of enrollment in the application area is similar. Below, in Figure 5.16, it is presented one of the nodes after enrollement using ABP with all the associated keys.

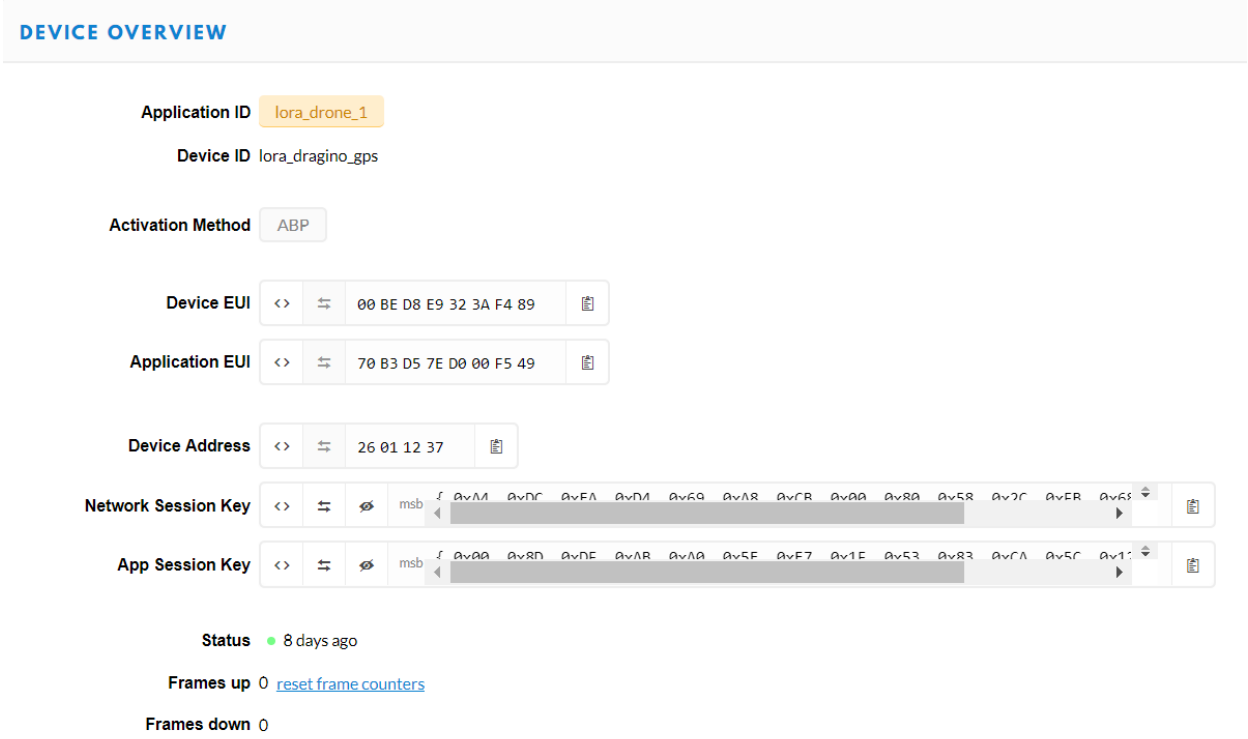

Figure 5.16 – Example of enrolled end device using ABP

The enrollment was done using ABP, also having the possibility to use the alternative mechanism, OTAA (they were both described in *Chapter 3*). However, even though OTAA is a more secure and the preferred way to enroll devices, this chipset supports only the legacy mechanism, ABP.

An important study in the evolution of the solution was the LoRa Dragino pins configuration and their use within the program. The pins must be used/reserved in the application and synchronized with lmic.h library as it follows:

```
const lmic_pinmap lmic_pins = {
    .nss = 10,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 9,
    .dio = {2, 6, 7},
};
```

Firstly, I created a *Hello World* program to verify connection with The Things Network platform via LoRaWAN and I noticed the constrains which apply on this cloud solution. The most important is that the information is transmitted at about two minutes, which, taking into consideration the speed of the UAV, I can say that it does not fit my purpose.

To communicate with TTN, I used the libraries listed below:

- ✓ Lmic.h
- ✓ Hal/hal.h
- ✓ SPI.h

These libraries can be found on GitHub.

Lmic.h is a LoRaWAN C-library developed my IBM. This allows the portable implementations of the specifications of LoRa MAC, supporting EU-868 and US-915, class A and B devices.

Hal/hal.h allows the implementation of the hardware abstraction layer functions which means that it simplies the use of additional hardware and portability to new platforms.

SPI.h allows communication with Serial Peripheral Interface (SPI) devices, using Arduino as the master device.

In the same time, I have been studying reading the GPS via Software.Serial. This was possible using the following libraries:

- ✓ TinyGPS.h
- ✓ Software.Serial.h
- ✓ SPI.h

TinyGPS.h provides National Marine Electronics Association GPS data such as position, altitude, time, date, etc. The library ignores all but a few key GPS fields and avoids floating point to keep resource consumption at a low level.

Software.Serial.h allows serial communication on other digital pins of the Arduino board (not only pins 0 and 1 that are specifically designed for this), replicating the functionality.

When integrating the GPS reading and the coordinates, the reading, stack organization and verification of the status of the two implied chipsets (GPS and LoRa which are functionally independent even though they coexist on the same shield), it results a program whose compilation exceeded the 32 kB workload limitations of Arduino Uno.

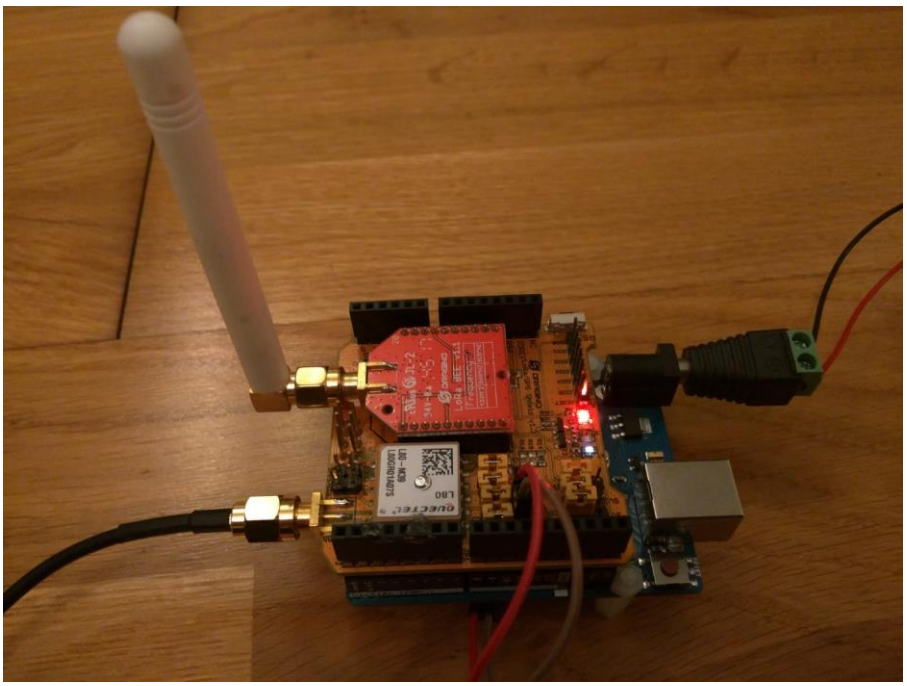

Figure 5.17 – LoRa Node with GPS

Considering all these, I replaced the Uno platform with an Arduino Mega 2560 which has 256 kB program memory.

This action produced results as the communication and data transmission to TTN became possible. However, the platform constraints lead to obtaining results at approximately two minutes which is unacceptable from the perspective of the UAV speed (this can finish a mission in two minutes so, in this case, receiving the coordinates at this interval would be useless).

All the above outline an IoT solution with a limited recurrence with respect to our purpose due to repeatability. Getting information about two minutes can be an use case for humidity sensor networks (smart agriculture), measuring temperature during some processes, sampled energy consumption (smart grid), but not necessary for our study because the UAV travels with a fairly high speed.

In the same time, I studied the coverage offered by the gateway that I deployed and, I concluded that it is about 2.1 kilometers. However, when talking about coverage, several things should be kept in mind: the environment (in this case studies were conducted in an urban context), the placement of the gateway, which was outdoor, the geometric shading, and the height (the site was not at a reasonable height).

Instead, in open field, the gateway had a coverage of 5.7 kilometers, showing that LoRaWAN is a great solution of communication for an UAV.

Obviously, LoRa, as a standard, promises much more, but the measurements done by me were made on normal devices with antennas not necessarily optimized.

During the study, I used a Hameg HM5010 specter analyzer (frequency range: 0.15 – 1050 MHz) to analyze LoRa Europe band loading (868 MHz) in the context of the test conducted in Bucharest. There was not any traffic/interference in this band. Furthermore, I tested other available antennas, in adjacent bands, the results being sensitive to frequency centering (the antennas that were used: Wi-Fi, GSM 900/1800/2100).

From all that were described above, I decided that a client – server solution, independent of the constraints of an IoT platform, would be better for my implementation. This will be described in the next part.

4.8 Point to Point Communication using LoRaWAN Protocol

Am rethought the approach, using the previous experience with the gateway and LoRa node with GPS. I kept the LoRa node already developed, but, this time with changes to report the coordinates over LoRa protocol at two seconds. This is good enough from the perspective of the speed of the UAV.

In Figure 5.18 it can be seen the transmission data flow both using the IoT platform and the alternative point to point solution.

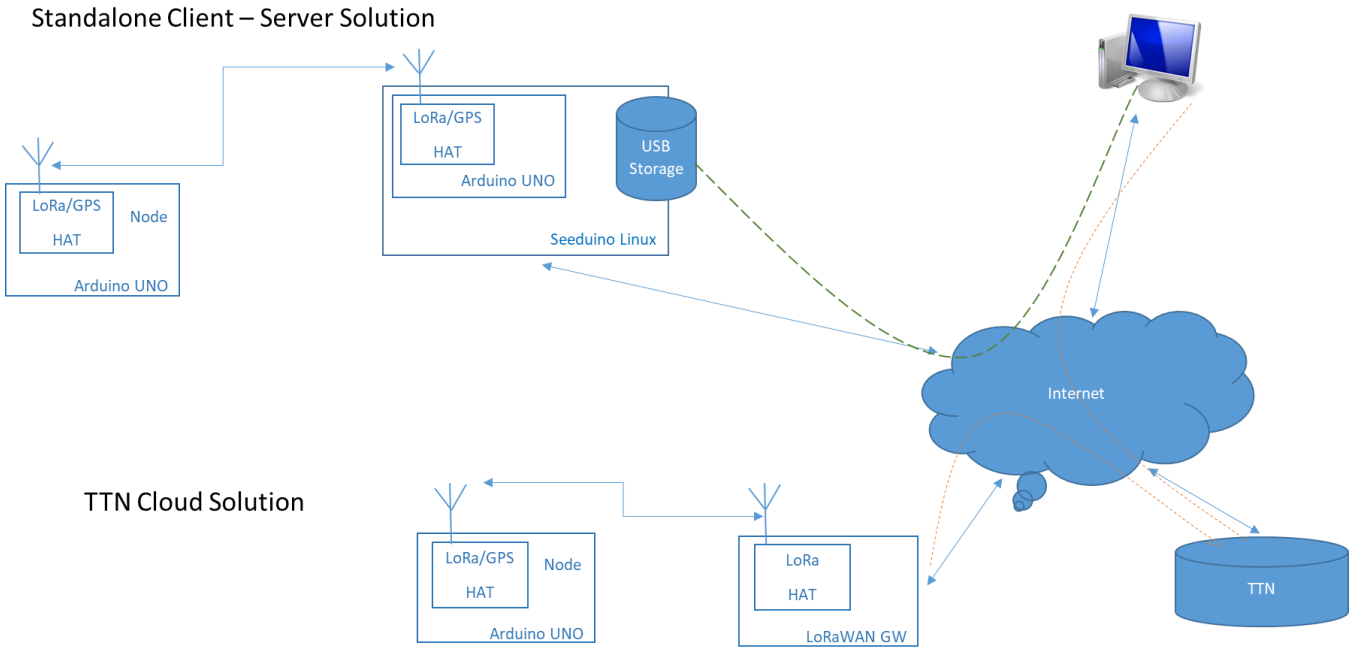

Figure 5.18 – Data transmission flow using TTN and point to point communication

For client side I used an Arduino Uno and a LoRa/GPS Shield and for server side, an Arduino Uno, a LoRa Sheild, a Yun Shield and a USB flash.

The flow of the implementation is the following: GPS data captured by client will be sent out via LoRa to the server. In the meantime, the server-side listens on the LoRa specific frequency; once it receives data from client, a LED will be turned on and GPS data will be logged to a USB flash.

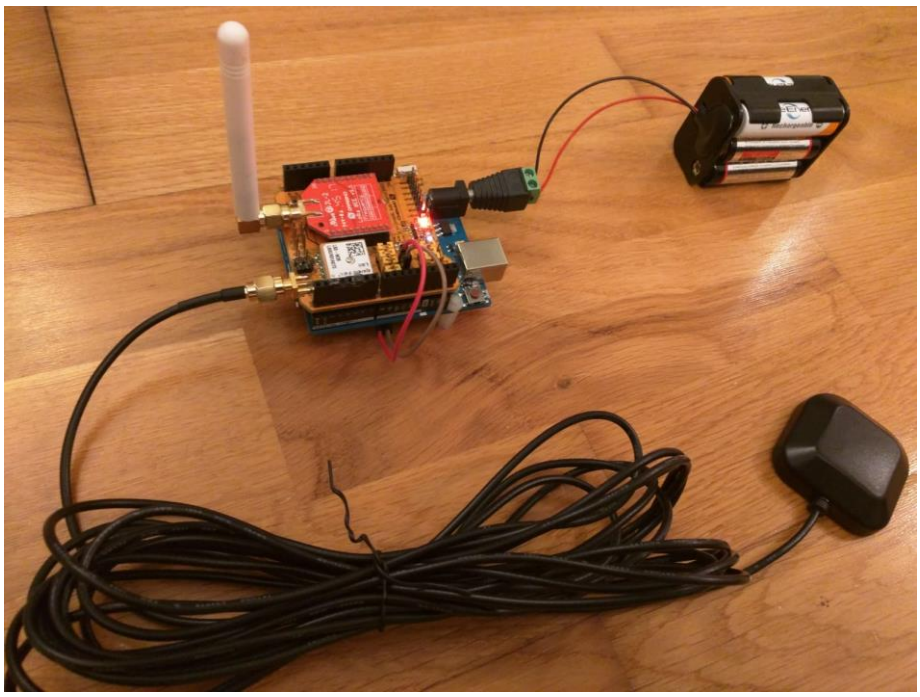

Figure 5.19 – Client side with external GPS antenna

Using a Seeduino Cloud, I created a server for GPS coordinates acquisition through sequential writing in a .csv file.

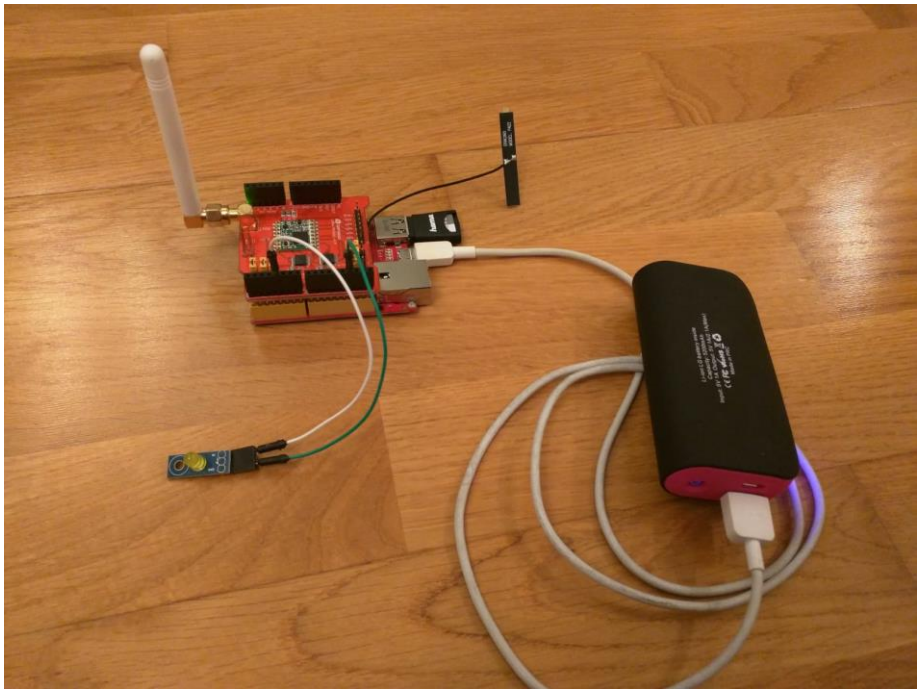

Figure 5.20 – Server side

Seeeduino is a device that incorporates an Arduino Uno device type and a Linux device (more exactly an OpenWRT edition), connected in bridge. The Linux device has both Ethernet and Wi-Fi interfaces, being accessed easy.

It resulted a mobile server which can be placed anywhere (such as UAV testing area) with the possibility of offline data acquisition or through mobile hotspot connection.

Data stored in .csv file can be converted in .kml file using KMLCSV Converter, this way obtaining the UAV coordinates during the flight.

	A	B	C	D
1	25.99082	44.50414	06/19/18-16:50:26	
2	25.99085	44.50412	06/19/18-16:50:28	
3	25.99091	44.50418	06/19/18-16:50:30	
4	25.99097	44.50422	06/19/18-16:50:32	
5	25.99102	44.50425	06/19/18-16:50:34	
6	25.99105	44.50427	06/19/18-16:50:36	
7	25.99106	44.50431	06/19/18-16:50:38	
8	25.99104	44.50435	06/19/18-16:50:40	
9	25.991	44.50439	06/19/18-16:50:42	
10	25.99096	44.50443	06/19/18-16:50:44	
11	25.99092	44.50445	06/19/18-16:50:46	
12	25.99088	44.50448	06/19/18-16:50:48	
13	25.99083	44.50451	06/19/18-16:50:50	
14	25.99079	44.50456	06/19/18-16:50:52	
15	25.99074	44.5046	06/19/18-16:50:54	
16	25.99069	44.50464	06/19/18-16:50:56	
17	25.99064	44.50469	06/19/18-16:50:58	
18	25.99057	44.50475	06/19/18-16:51:00	
19	25.99051	44.5048	06/19/18-16:51:02	

Figure 5.21 – Coordinates stored on a .csv file on the server

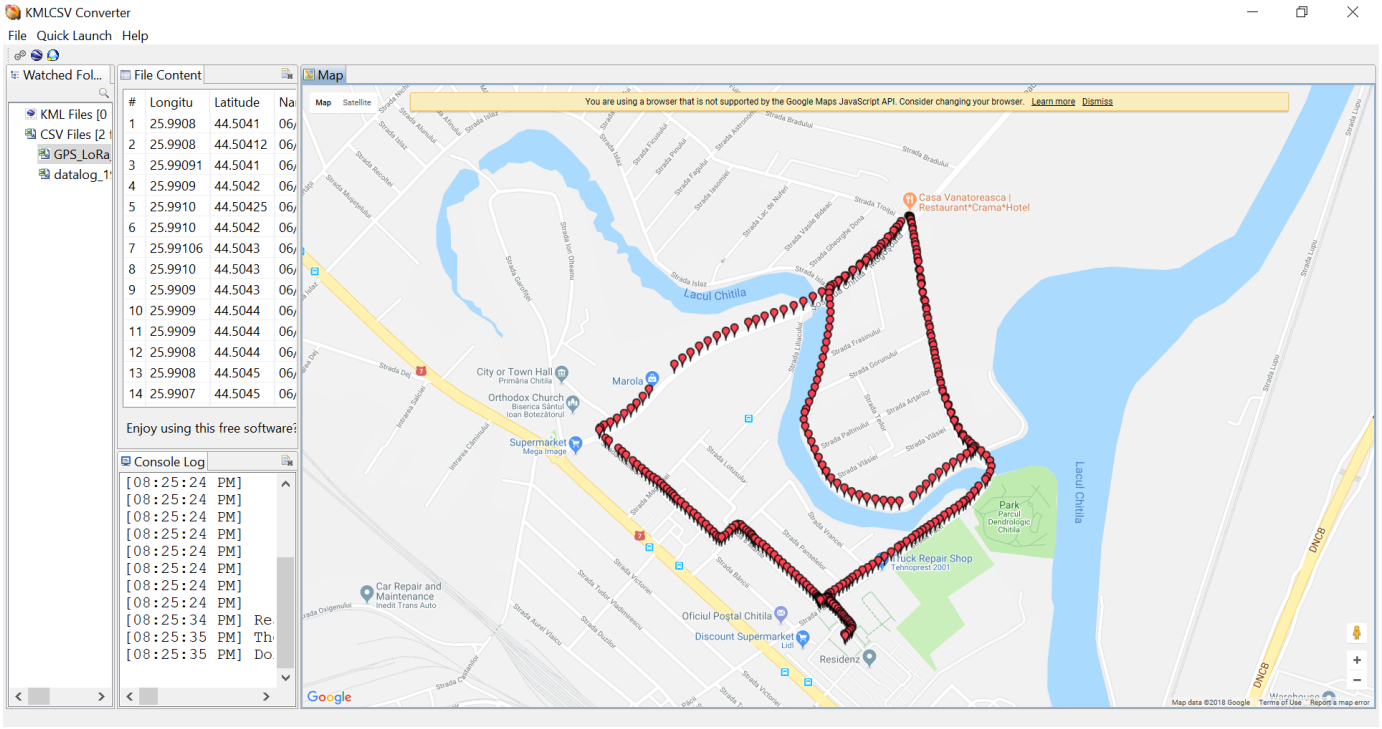

Figure 5.22 – UAV flight path highlighted on a map

Depending on the purpose of communication with the UAV, the terrestrial station may need system portability and/or large hardware resources – processing power, memory. As the UAV is a mobile system, the wireless control system needs to adapt to this requirement.

On the other hand, if we assume that this system must simultaneously control or store the information received from multiple UAVs, the control station needs to have large hardware resources.

CONCLUSION AND FUTURE IMPLEMENTATIONS

GENERAL CONCLUSIONS

As I have already presented from the introduction of the paper, my purpose was to investigate as many as possible wireless protocols that might be compatible for ground-to-air communication with an UAV in motion.

The drone used for test flights and the study of wireless protocols communication was the Lehmann Aviation LA300. In the materials that were part of the research phase of the paper, the drone was used for agriculture related purposes (monitoring the harvest), mentioning the possibility of use within military applications.

The UAV required to have the original electronic part replaced, as elevons were no longer effective during flight maneuvers. For take-off, this is initially propelled manually by a human operator, the motor starting when it reaches a certain acceleration.

For the moment, the UAV, equipped with a LoRa module has to transmit the GPS coordinates of its location in a manner as close to real-time as possible. The data is transmitted using the LoRaWAN protocol that has proven to be the most appropriate of the studied protocols for this case study.

PERSONAL CONTRIBUTIONS

My personal contributions in this paper are the following:

- ✓ After I found and analyzed the causes of the malfunctions of the UAV (the right elevon, was not functioning correctly), I replaced some components with the following: a Pixracer v1.0,

a X8R radio receiver, an Ublox GPS, a pair of 433 MHz radio modems for telemetry, a MC5983 3-Axis IIC/SPI Digital Compass magnetometer, a buzzer and a safety switch, connecting them with the original parts that were kept: the electrical motor, the servos and the battery.

- ✓ I deployed a LoRaWAN gateway because there was no LoRaWAN network in this region.
- ✓ I implemented two ways of information transmission between the UAV and a server: one using TTN platform, the LoRaWAN gateway and a LoRa node and the other using point to point communication
- ✓ I developed the code for GPS coordinates transmission

Figure 6.1 – Lehmann Aviation LA300 UAV equipped with GoPro Hero 4

FUTURE WORK

A domain as large and wide as the one represented by the wireless protocols is in a continuous development. For the time being, I've succeeded in determining the most appropriate protocol that can be used for a communication process between an UAV and a terrestrial drone. However, since new advantages are added periodically to most of the protocols that were studied in the context of this paper, part of the future works will be represented by a monitoring process of the wireless standards and their implementations, to see if another one can be used for my applications, with even better results.

Also, while talking about the applications that can be implemented with the help of the UAVs and the drones, numerous examples can be named in various domains: starting from delivery of packages (partially already implemented by Amazon), to medical urgent delivery of drugs and instruments and finishing with military surveillance and interventions missions. Personally, I can consider the last one as being the most important and relevant, since a successful implementation can protect millions of human lives in cities that are threatened by war while also backing up brave soldiers that are putting their life on the line each time they have a major intervention.

Considering those factors and areas that need urgent support from the telecommunications field, I will focus my future works on a coordinate set of actions between the Lehman LA 300 UAV and the PhantomX AX Hexapod Mark II. Precisely, after gathering GPS coordinates from the aerial drone and synchronizing them with a series of images taken with the GoPro 4 camera, that information can be send to the terrestrial drone, where it will make decisions on its own regarding

where it should move, what tools will it need (e.g. for bomb defusing) while also calculating the risk factor regarding an intervention (both human and non-human, using only drones).

All of this will be possible thanks to a mini-computer that will be mounted on the back of the hexapod and the client-server application that was described in Chapter 4. The same logic will be used here, while also better solutions will be sought for the mini-computer and the way the GPS coordinates are being stored.

This is just one example of the applications that can be developed using the technologies described in the presented paper, but it is perhaps the most urgent and significant one since it will be able to save lives. Thousands of others can be implemented, because the solutions that I presented are "open-door", giving the engineers only one limit: their imagination.

REFERENCES

[1] http://www.zigbee.org – accessed on June 5, 2018

[2] Lee, J., Suu Y., Shen, C., "A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi", The 33rd Annual Conference of the IEEE Industrial Electronics Society (IECON), Taiwan, 2007

[3] Faludi, R., *Building Wireless Sensor Networks*, Published by O'Reilly, Sebastopol, 2011

[4] NB-IoT DEPLOYMENT GUIDE to Basic Feature set Requirements, https://www.gsma.com/iot/wp-content/uploads/2018/04/NB-IoT_Deployment_Guide_v2_5Apr2018.pdf – accessed on June 23, 2018

[5] https://www.leverege.com/research-papers/lora-lorawan-primer – accessed on June 7, 2018

[6] Ferran, A., Xavier, V., Pere T.P., Borja M., Joan, M.S., Thomas, W., "Understanding the Limits of LoRaWAN", IEEE Communications Magazine, Volume 55, Issue 9, 2017

[7] Norbert, B., Fernando K., "LoRaWAN in the Wild: Measurements from The Things Network"

[8] https://lora-alliance.org/about-lorawan – accessed on June 25, 2018

[9] https://www.resiot.io/en/what-is-lorawan/ – accessed on June 20, 2018

[10] https://www.thethingsnetwork.org/docs/lorawan/ – accessed on June 14, 2018

[11] LoRa Security Building a Secure LoRa Solution, https://labs.mwrinfosecurity.com/assets/BlogFiles/mwri-LoRa-security-guide-1.2-2016-03-22.pdf – accessed on June 8, 2018

[12] "L-A Series Drones User Guide", Lehmann Aviation, 2015

[13] http://www.trossenrobotics.com/hex-mk2 – accessed on June 5, 2018

[14] https://store.arduino.cc/usa/arduino-uno-rev3 – accessed on June 10, 2018

[15] https://www.seeedstudio.com/Seeeduino-Cloud-Arduino-Yun-compatible-openWRT-controller-p-2123.html – accessed on June 10, 2018

[16] http://www.dragino.com/products/module/item/102-lora-shield.html – accessed on June 10, 2018

[17] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ – accessed on June 12, 2018

[18] http://ardupilot.org/planner/ – accessed on June 12, 2018

[19] https://docs.px4.io/en/flight_controller/pixracer.html – accessed on June 12, 2018

[20] http://www.dragino.com/products/lora/item/106-lora-gps-hat.html – accessed on June 10, 2018

ANNEX 1

```
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>

static const PROG_MEM u1_t NWKSKEY[16] = { 0x01, 0xC9, 0x75, 0x6B,
0xFC, 0x55, 0xA7, 0xB2, 0x42, 0x85, 0x73, 0xDC, 0x30, 0xC2, 0x8E,
0x5B };

static const u1_t PROG_MEM APPSKEY[16] = { 0x93, 0xB6, 0xD0, 0x3A,
0x3A, 0x8A, 0x9C, 0x72, 0x5D, 0xC7, 0x49, 0x53, 0x83, 0x42, 0x6D,
0x39 };

static const u4_t DEV_ADDR = 0x260112D9 ;

void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

static uint8_t mydata[] = "Hello Ioana";
static osjob_t sendjob;

const unsigned TX_INTERVAL = 20; //transmission interval

const lmic_pinmap lmic_pins = {
    .nss = 10,
```

```
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 9,
    .dio = {2, 6, 7},
};

void onEvent (ev_t ev) {
    Serial.print(os_getTime());
    Serial.print(": ");
    switch(ev) {
        case EV_SCAN_TIMEOUT:
            Serial.println(F("EV_SCAN_TIMEOUT"));
            break;
        case EV_BEACON_FOUND:
            Serial.println(F("EV_BEACON_FOUND"));
            break;
        case EV_BEACON_MISSED:
            Serial.println(F("EV_BEACON_MISSED"));
            break;
        case EV_BEACON_TRACKED:
            Serial.println(F("EV_BEACON_TRACKED"));
            break;
        case EV_JOINING:
            Serial.println(F("EV_JOINING"));
            break;
        case EV_JOINED:
            Serial.println(F("EV_JOINED"));
            break;
        case EV_RFU1:
            Serial.println(F("EV_RFU1"));
            break;
        case EV_JOIN_FAILED:
            Serial.println(F("EV_JOIN_FAILED"));
            break;
        case EV_REJOIN_FAILED:
            Serial.println(F("EV_REJOIN_FAILED"));
            break;
        case EV_TXCOMPLETE:
            Serial.println(F("EV_TXCOMPLETE (includes waiting for
RX windows)"));
            if (LMIC.txrxFlags & TXRX_ACK)
              Serial.println(F("Received ack"));
            if (LMIC.dataLen) {
              Serial.println(F("Received "));
              Serial.println(LMIC.dataLen);
              Serial.println(F(" bytes of payload"));
            }

            os_setTimedCallback(&sendjob,
os_getTime()+sec2osticks(TX_INTERVAL), do_send);
            break;
        case EV_LOST_TSYNC:
            Serial.println(F("EV_LOST_TSYNC"));
            break;
```

```
        case EV_RESET:
            Serial.println(F("EV_RESET"));
            break;
        case EV_RXCOMPLETE:
            Serial.println(F("EV_RXCOMPLETE"));
            break;
        case EV_LINK_DEAD:
            Serial.println(F("EV_LINK_DEAD"));
            break;
        case EV_LINK_ALIVE:
            Serial.println(F("EV_LINK_ALIVE"));
            break;
         default:
            Serial.println(F("Unknown event"));
            break;
    }
}

void do_send(osjob_t* j){
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    } else {
        LMIC_setTxData2(1, mydata, sizeof(mydata)-1, 0);
        Serial.println(F("Packet queued"));
    }
}

void setup() {
    Serial.begin(9600);
    Serial.println(F("Starting"));

    os_init();
    LMIC_reset();

    #ifdef PROG_MEM
    uint8_t appskey[sizeof(APPSKEY)];
    uint8_t nwkskey[sizeof(NWKSKEY)];
    memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
    memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
    LMIC_setSession (0x1, DEV_ADDR, nwkskey, appskey);
    #else
    LMIC_setSession (0x1, DEV_ADDR, NWKSKEY, APPSKEY); // prepare
data
    #endif
    LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI); // set the channel

    LMIC_setLinkCheckMode(0);

    LMIC.dn2Dr = DR_SF9;

    LMIC_setDrTxpow(DR_SF7,14);
```

```
    do_send(&sendjob);
}

void loop() {
    os_runloop_once();
}
```

ANNEX 2

```
#include <SoftwareSerial.h>
#include <TinyGPS.h>

TinyGPS gps_data;
SoftwareSerial ss(3, 4);

static void smart_delay(unsigned long ms);
static void printfloat(float val, float invalid, int len, int
prec);
static void printint(unsigned long val, unsigned long invalid, int
len);
static void printdate(TinyGPS &gps_data);
static void printstr(const char *str, int len);

void setup()
{
Serial.begin(9600);
  ss.begin(9600);
  while (!Serial) {
     ;
  };
  Serial.println("Minitor Dragino LoRa GPS Shield Status");
  Serial.print("Testing TinyGPS library v. ");
  Serial.println(TinyGPS::library_version());
```

```
  Serial.println();
  Serial.println("Sats Latitude  Longitude  Fix  Date       Time
Date Alt");
  Serial.println("     (deg)     (deg)      Age
Age  (m)");
  Serial.println("--------------------------------------------------
----------");
}

void loop()
{
  float flat, flon;
  unsigned long age, date, time, chars = 0;
  unsigned short sentences = 0, failed = 0;

  printint(gps_data.satellites(), TinyGPS::GPS_INVALID_SATELLITES,
5);
  gps_data.f_get_position(&flat, &flon, &age);
  printdate(gps_data);
  printfloat(gps_data.f_altitude(),
TinyGPS::GPS_INVALID_F_ALTITUDE, 7, 2);

  gps_data.stats(&chars, &sentences, &failed);
  printint(chars, 0xFFFFFFFF, 6);
  printint(sentences, 0xFFFFFFFF, 10);
  printint(failed, 0xFFFFFFFF, 9);
  Serial.println();

  smart_delay(1000);
  }

static void smart_delay(unsigned long ms)
{
  unsigned long start = millis();
  do
  {
    while (ss.available())
    {
      gps_data.encode(ss.read());
    }
  } while (millis() - start < ms);
}

static void printfloat(float val, float invalid, int len, int prec)
{
  if (val == invalid)
  {
    while (len-- > 1)
      Serial.print('*');
    Serial.print(' ');
  }
  else
  {
```

```
    Serial.print(val, prec);
    int vi = abs((int)val);
    int flen = prec + (val < 0.0 ? 2 : 1);
    flen += vi >= 1000 ? 4 : vi >= 100 ? 3 : vi >= 10 ? 2 : 1;
    for (int i=flen; i<len; ++i)
      Serial.print(' ');
  }
  smart_delay(0);
}

static void printint(unsigned long val, unsigned long invalid, int
len)
{
  char sz[32];
  if (val == invalid)
    strcpy(sz, "*******");
  else
    sprintf(sz, "%ld", val);
  sz[len] = 0;
  for (int i=strlen(sz); i<len; ++i)
    sz[i] = ' ';
  if (len > 0)
    sz[len-1] = ' ';
  Serial.print(sz);
  smart_delay(0);
}

static void printdate(TinyGPS &gps_data)
{
  int year;
  byte month, day, hour, minute, second, hundredths;
  unsigned long age;
  gps_data.crack_datetime(&year, &month, &day, &hour, &minute,
&second, &hundredths, &age);
  if (age == TinyGPS::GPS_INVALID_AGE)
    Serial.print("********** ******** ");
  else
  {
    char sz[32];
    sprintf(sz, "%02d/%02d/%02d %02d:%02d:%02d ",
        month, day, year, hour, minute, second);
    Serial.print(sz);
  }
  printint(age, TinyGPS::GPS_INVALID_AGE, 5);
  smart_delay(0);
}

static void printstr(const char *str, int len)
{
  int slen = strlen(str);
  for (int i=0; i<len; ++i)
    Serial.print(i<slen ? str[i] : ' ');
  smart_delay(0);
```

```
}
```

ANNEX 3

```
#include <SoftwareSerial.h>
#include <TinyGPS.h>
#include <SPI.h>
#include <RH_RF95.h>

RH_RF95 rf95;
TinyGPS gps_data;
SoftwareSerial ss(3, 4);
String string_a="";
String string_b="";
char databuf[100];
uint8_t string_out[100];
char lon[20]={"\0"};
char lat[20]={"\0"};

void setup()
{
  Serial.begin(9600);
  ss.begin(9600);
    if (!rf95.init())
    Serial.println("init failed");
    ss.print("Simple TinyGPS library v. ");
    ss.println(TinyGPS::library_version());
    Serial.println();
}
```

```
void loop()
{
  ss.println("Sending to rf95_server");
  bool newData = false;
  unsigned long chars;
  unsigned short sentences, failed;

  for (unsigned long start = millis(); millis() - start < 1000;)
  {
    while (Serial.available())
    {
      char c = Serial.read();
      if (gps_data.encode(c))
      newData = true;
    }
  }
    if (newData)
  {
    float f_lat, f_lon;
    unsigned long _age;
    gps_data.f_get_position(&f_lat, &f_lon, &_age);
    ss.print("LAT =");
    ss.print(f_lat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : f_lat,
6);
    ss.print(" LON =");
    ss.print(f_lon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : f_lon,
6);
    ss.print(" SAT =");
    ss.print(gps_data.satellites() ==
TinyGPS::GPS_INVALID_SATELLITES ? 0 : gps_data.satellites());
    ss.print(" PREC =");
    ss.print(gps_data.hdop() == TinyGPS::GPS_INVALID_HDOP ? 0 :
gps_data.hdop());
    f_lat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : f_lat, 6;
    f_lon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : f_lon, 6;
    string_a +=dtostrf(f_lat, 0, 6, lat);
    string_b +=dtostrf(f_lon, 0, 6, lon);
    ss.println(strcat(strcat(lon,","),lat));
    strcpy(lat,lon);
    ss.println(lat);
    strcpy((char *)string_out,lat);
    rf95.send(string_out, sizeof(string_out));
    uint8_t indatabuf[RH_RF95_MAX_MESSAGE_LEN];
    uint8_t len = sizeof(indatabuf);
    if (rf95.waitAvailableTimeout(3000))
     {
       if (rf95.recv(indatabuf, &len))
      {
         ss.print("got reply: ");
         ss.println((char*)indatabuf);
      }
      else
```

```
      {
      ss.println("recv failed");
      }
    }
    else
    {
      ss.println("No reply, is rf95_server running?");
    }
  delay(400);
 }
  gps_data.stats(&chars, &sentences, &failed);
  ss.print(" CHARS=");
  ss.print(chars);
  ss.print(" SENTENCES=");
  ss.print(sentences);
  ss.print(" CSUM ERR=");
  ss.println(failed);
  if (chars == 0)
  ss.println("No characters received");
}
```

ANNEX 4

```
#include <SPI.h>
#include <RH_RF95.h>

#include <FileIO.h>
#include <Console.h>

RH_RF95 rf95;
int led = 4;
int reset_lora = 9;
String data_string = "";

void setup()
{
  pinMode(led, OUTPUT);
  pinMode(reset_lora, OUTPUT);
  Bridge.begin();
  Console.begin();
  FileSystem.begin();

  digitalWrite(reset_lora, LOW);
  delay(1000);
  digitalWrite(reset_lora, HIGH);

  if (!rf95.init())
    Console.println("init failed");
```

```
}

void loop()
{
  data_string="";
  if (rf95.available())
  {
    Console.println("Get new message");
    uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
    uint8_t len = sizeof(buf);
    if (rf95.recv(buf, &len))
    {
      digitalWrite(led, HIGH);
      Console.print("got message: ");
      Console.println((char*)buf);
      Console.print("RSSI: ");
      Console.println(rf95.lastRssi(), DEC);

      data_string += String((char*)buf);
      data_string += ",";
      data_string += getTimeStamp();

      uint8_t data[] = "200 OK";
      rf95.send(data, sizeof(data));
      rf95.waitPacketSent();
      Console.println("Sent a reply");

      File dataFile = FileSystem.open("/mnt/data/datalog.csv",
FILE_APPEND);

      if (dataFile) {
        dataFile.println(data_string);
        dataFile.close();
        Console.println(data_string);
        Console.println("");
      }
        else
      {
        Console.println("error opening datalog.csv");
      }
      digitalWrite(led, LOW);
    }
    else
    {
      Console.println("recv failed");
    }
  }
}
String getTimeStamp() {
  String result;
  Process time;
  time.begin("date");
```

```
  time.addParameter("+%D-%T");
  time.run();
    while(time.available()>0) {
    char c = time.read();
    if(c != '\n')
      result += c;
  }
  return result;
}
```