34322 Communication Network for Control and Data Acquisition

Mini-project

LoRa in racing cars

Anders Vedel Eriksen s194017

Date: May 9, 2022

Abstract

This project concerns the implementation of wireless telemetry for use in student projects such as the Vermilion Racing Team race car at DTU. LoRa and LoRaWAN are introduced and compared with LTE technology, and finally an experiment with a peer-to-peer setup with two LoRa nodes communicating is set up and put to the test. The conclusion on the experiment is that while data can be transferred with high integrity, the transmission time in conjunction with regulations in the EU concerning the duty cycle might make the data rate too slow to use for a live data feel, monitoring sensor data from the car. Further experimentation could be done to better decide if LoRa is a suitable technology for this usecase.

1 Introduction

At DTU, several student teams are working with building cars for different purposes. For example, the Ecocar is competing for driving the furthest possible on a liter of fuel, the Solar car is trying to drive on sunpower only, and Vermilion Racing is building an electric racecar to compete in Formula Student. The Vermilion racecar is the inspiration for this project, where the topic is to investigate if LoRa is a suitable technology for wireless telemetry in situations, reading sensors on a high speed moving vehicle in real time.
A concrete case is that the car in the summer is shipped to Italy, where the car is gonna race on the Riccardo Paletti di Varano 'de Melegrari racetrack, near Parma. On the track, the distance between the car and the paddock where the team are during the race, is approximately 600 meters, and the biggest distance from one point to another on the track is just shy of one kilometer. The terrain on and around the track is mostly unobstructed.

2 Technologies

In Formula 1, each car is equipped with over 300 sensors, generating 1.1 million datapoints between the racetrack and the pits[6]. Through the years, Formula 1 has adopted new technologies and have traditionally been on the forefront of wireless technology. Different proprietary WiFi solutions have been used for telemetry between the cars on the track, and recently, 5G has been adopted to be used for telemetry[2]. As 5G is still relatively new, we may not have coverage at the track, so it is safer to base this discussion on 4G/LTE.

2.1 LTE

LTE has one big advantage over LoRa, in that it has a much higher bitrate at 300 Mbps download and 75 Mbps upload, or even higher with newer LTE technology[8]. With such a high bit rate, it should be possible to send and receive much more detailed data. In a car, it would mean that one could have more sensors on the car, or increase the frequency at which values are updated. 4G/LTE is however working on a network, and the system built for the car will be reliant on stable operation of the network, and that the network won't be overpopulated with clients. This will vary from place to place. Typically, solutions with LTE is sending the data to a server on the internet, which is then accesible on a laptop, from which the data can be read and used. A lot of subsystems is needed for this to work, and the setup could be complicated. Furthermore, as LTE exist in one of the commercial frequency spectrums, and server functionality may be subsciption based also, there could be a continuous economic expense.

2.2 LoRaWAN

LoRa and LoRaWAN are two different things not to be confused; LoRa is the physical layer level, the transmitting technology, while LoRaWAN is a Data Link layer protocol. LoRaWAN is optimized for low power consumption primarily for battery powered IoT devices. Typically, a pool of end devices are laid out in a star-of-stars topology, connected to a central gateway that links the end devices with a network server. Communication between end devices and gateway is radiocommunication, while communication between gateway and network server would be IP connected[3]. LoRa operates in the sub GHz band, with a number of channels spread out over this band. These channels have different bandwidths, and some are uplink while others are downlink. Most of these channels have a 125 KHz bandwidth, while a few have 250- or 500 KHz bandwidth. Within these bands, communication are provided with a special chirp modulation technique proprietary to LoRa[1]. An end device have a few rules to obey by; The end device should change channel for every transmission, making the system more robust to interference. The end device should change its transmit periodicity, preventing syncronization between end devices. Finally, an end device should comply with local regulations. For example, the open band in EU is different to the one in the US, and there are different rules for transmit duty cycles. Within the LoRaWAN specification there are three different classes of end devices, whereas most are class A. Class A devices have bi-directional communication, where an end devices uplink transmission is followed by a downlink acknowledge message. The end device is "awake" in the transmission time and two following windows for receiving confirmation of succesfull transmission. If the message is acknowledged, the end device can go to sleep, wherein the powersaving aspect is. This is an example of an ALOHA type protocol with small time variations in transmission periodicity[3].

2.3 LoRa and Peer-to-Peer

While LoRaWAN is network based, LoRa technology can also be used on a peer-to-peer (P2P) basis, that is that two devices can talk directly. This is done by utilizing the LoRa physical layer and the way radio communication is done here. The beforementioned chirp modulation, see figure 1, is a series of chirps where the frequency is sweeped across the channel bandwidth a number of times with varying direction. This way of modulation makes is easier to decode the signal even if it is has lower amplitude, and is actually under the noise floor[1][7].

LoRa works in a unlicensed band, so a system can be build with no continuous economic expenses. The main limitation of using the 868MHz band is that there are dutycycle regulations, instated by EU. A node can transmit only 1% of the time. That means, that for the time it takes to transmit a message, the sender should be quiet 99 times that time afterwards. This of course puts a limitation to rate at which messages are send, or it limits the length of messages to reduce transmission time.

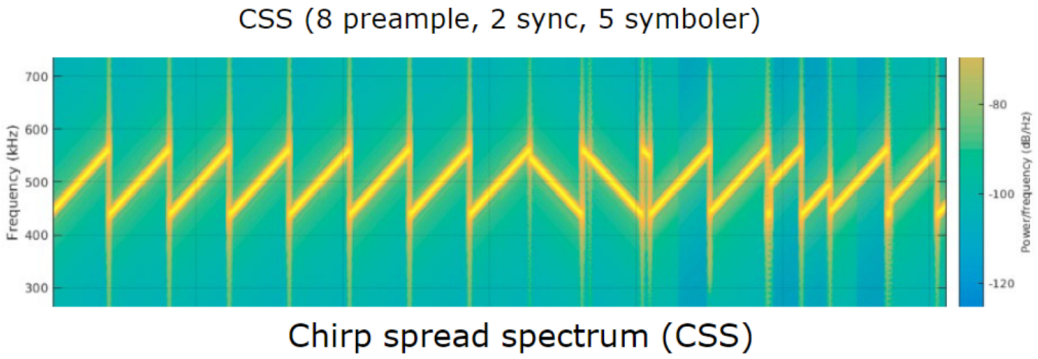

Figure 1: Chirp frequency modulation, proprietary to LoRa

3 Lora on RN2483A

When setting up a LoRa P2P connection, some parameters must be tuned and matched between nodes. The highest possible datarate should be spreading factor 7 with a bandwidth of 250KHz[5]. Is a transmitter transmitting with a 125KHz bandwidth, a receiver should receive in a matching 125KHz bandwidth. For maximum signal range, the power of the transmitter should be maximized. The RN2483A is designed to transceive at a power up to 14 dBm. LoRa has some amount of forward error correction built in which can be configured. In an area with high interference, one might choose a high ratio of error correction, while in an area with low interference, a low ratio might be better suited. A high coding rate ratio will make the transmission more reliable, but at the cost of transfer rate[4].

```
loraSerial.println("radio set freq 869100000");   #centerfrequency
loraSerial.println("radio set sf sf7");           #spread factor
loraSerial.println("radio set bw 125");           #radio bandwidth
loraSerial.println("radio set rxbw 125");         #receiver bandwidth
loraSerial.println("radio set pwr 14")            #power
```

4 Experiment and Results

A experiment was set up, to see LoRa working. The setup was a NodeMCU microcontroller connected to a Microchip®RN2483A LoRa module, working as a transmitter. A similar setup with the same microcontroller and RN2483A LoRa board was set up as receiver, and through a USB cable connected to a PC via UART, communication with serial. The PC is running a python script to

read and interpret the incoming data and plotting it on a graph.

The transmitter has a preloaded array of twenty values, which when plottet on a time axis shows a sinusoidal wave. Also, a value containing time information is also transmitted. All in all, a 4 byte value is to be transmitted, where the least significant byte is the sinewave value, and the remaining are for the time value. To send a message with the RN2483A board, a command containing a specific string and the data, also transmitted as string type, is send from the microcontroller to the LoRa module via a serial connection. For this program, a virtual serial connection is used between microcontroller and LoRa module, as we want to save the physical UART serial communication in the microcontroller for communication between the receiver microcontroller and the PC. The command for transmitting via serial is:

```
loraserial.println("radio rx " + String(message(), HEX));
```

"radio rx" signals that we want to transmit. In my code, the function message() generates the 4 byte value to transmit, which is converted to a string and formatted in HEX. HEX requires the least amount of characters to transmit, why it is used here. As a visual cue, a LED lights up when a transmission is underway.

The receiver is continuously listening for messages. When it has received a message, a led lights up. When a message is picked up on the virtual serial connection between LoRa module and microcontroller, it is forwarded to the PC via the built in UART interface.

A Python script is handling the data coming in via the serial connection. When printing the incoming value in the Python terminal, it could look something like this:

```
data = b'radio rx  0812467F\r\r\n'
```

The data is coming in as a byte array of size 21. The actual content of the message is the eight bytes formatted in HEX, '0812467F', where the last two digits represent the least significant byte from the original data, the sinusoidal value. This shows an inefficiency in transmitting; The original 4 byte integer data is formatted in HEX (8 digits), which is transferred as character bytes, taking up 8 bytes (one for each digit/character), double the amount of bits than originally required. While the data is formatted in HEX, it is in fact a byte, meaning that it will have the decimal value of the corresponding character in the ASCII table, being something completely different to the actual value. This is fixed with a few lines of python code:

```
data = data[10:18]          #data is a byte array of 8 bytes
data_hex = int(data, 16)    #casts an int, formats in base 16 (HEX)
```

The value can now be used, for example plottet to continuously monitor the value. This is what has been done in the following test. On two different days, experiments was carried out. Different spread factors was used, and finally the delay between transmissions was decreased. On day one, with spread factor 7 and 2000ms delay, some transmissions were lost, figure 2.
On the second day, in another building, the same experient was carried out, with no dropped transmissions, figure 3.
To test the datarate, the delay was decreased to 100ms. Furthermore, all unnecesary serial operations was left out to increase microcontroller speed. This resulted in still very few dropped transmissions, see figure 4. The transmission time was in the microcontroller measured to 20ms.

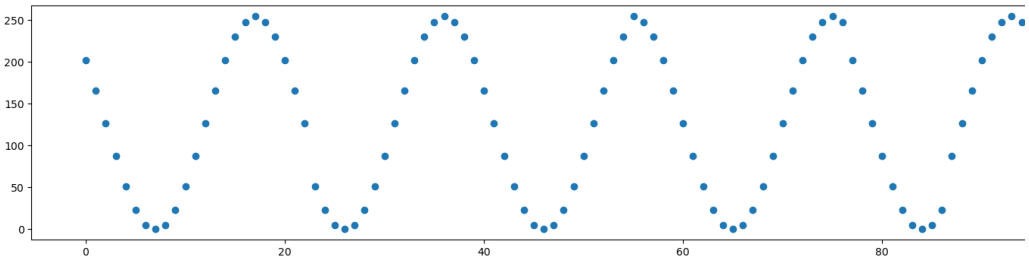

Figure 2: Day 1, sf7, delay 2000 ms

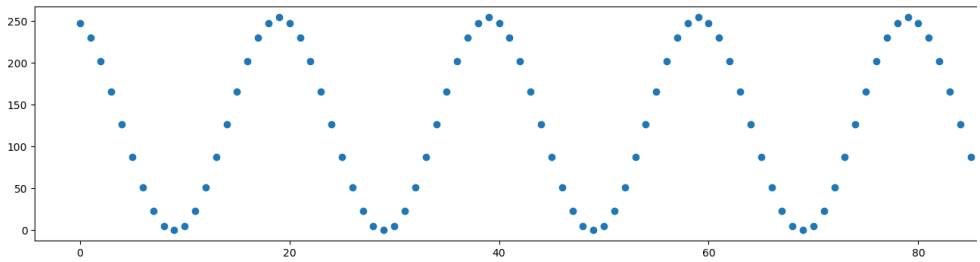

Figure 3: Day 2, sf7, delay 2000ms

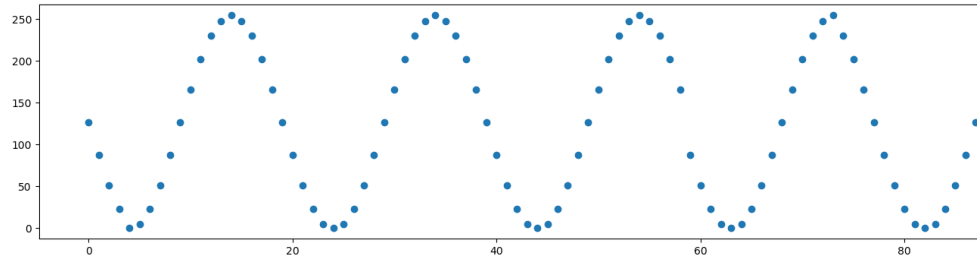

Figure 4: Day 2, sf7, delay 100 ms

On day one, a wider spread factor of 9 was also tested, which worked fine at 2000ms delay, but failed completely at 1000ms delay, figure 5.

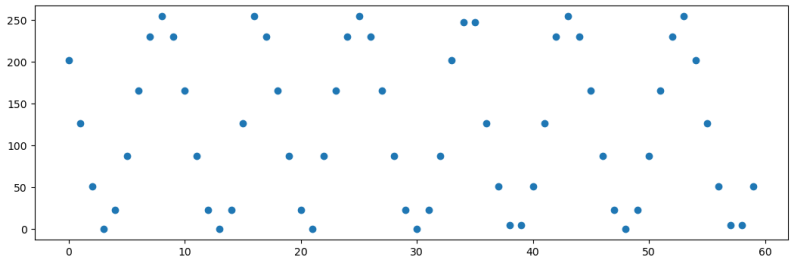

Figure 5: sf9, delay of 1 second

Conclusion

There was an obvious difference between the two locations from day one and day two, even though they were both inside buildings, with the same setups and configurations, and in a few instances it seemed the connection was very unreliable. Visual cues showed that transmission seemed two work, but receiving had problems. Maybe the issue was in the microcontroller or the connection between microcontroller and PC, leading to the microcontroller not able to keep up. This could be investigated further. The transmission time could be optimized to 20ms through removing unnecessary serial communication in the transmitter. With the beforementioned duty cycle requirements, this would mean that a transmission could take place every 2000ms, which is slower than what one could wish for in a race situation. Transmission integrity was however very good, with only very few lost transmissions. To really determine if the technology can be used for race cars, the distance between the nodes should be extended to expected real life distance on a race track of approximately 600m, which the transmitter moving at speed. A spread factor of 7 might prove too low here. Experimenting with a 500KHz channel could improve data rate.

Source code

See files on GitHub.

References

[1] Mahesh Sooriyabandara Usman Raza Parag Kulkarni. *Low Power Wide Area Networks: An Overview*. 2017. URL: https://learn.inside.dtu.dk/d2l/le/content/102624/viewContent/392082/Viewx.

[2] Fastco Works. *Connected at 200 mph: How 5G is transforming auto racing and beyond*. 2019. URL: https://www.fastcompany.com/90379283/connected-at-200-mph-how-5g-is-transforming-auto-racing-and-beyond.

[3] LoRa Alliance Inc. *TS001-1.0.4 LoRaWAN® L2 1.0.4 Specification*. 2020. URL: https://lora-alliance.org/resource_hub/lorawan-104-specification-package/.

[4] markqvist. *LoRa bitrate calculator and understanding LoRa parameters*. 2020. URL: https://unsigned.io/understanding-lora-parameters/.

[5] LoRa Alliance Inc. *RP002-1.0.3 LoRaWAN® Regional Parameters*. 2021. URL: https://lora-alliance.org/wp-content/uploads/2021/05/RP002-1.0.3-FINAL-1.pdf.

[6] Rich miller. *How Cloud Data-Crunching Power Accelerates the F1 Racing Experience*. 2021. URL: https://datacenterfrontier.com/how-cloud-data-crunching-power-accelerates-the-f1-racing-experience/.

[7] Martin Nordal Petersen. *Introduktion til LoRaWAN og SigFox*. 2022. URL: https://learn.inside.dtu.dk/d2l/le/content/102624/viewContent/392099/View.

[8] Lars Staalhagen. *Mobile Networks for Data Communication*. 2022. URL: https://learn.inside.dtu.dk/d2l/le/content/102624/viewContent/392011/View.