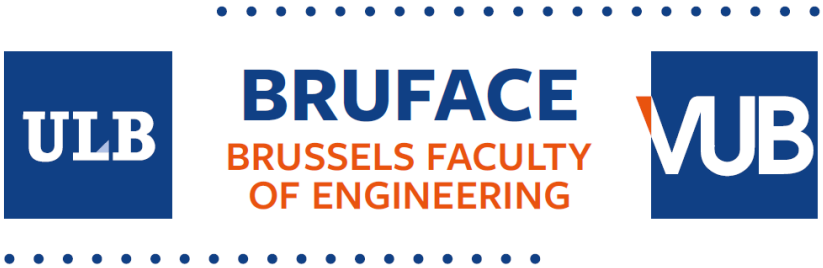

PROJ-H-417
Shell Eco-Marathon 2021-2022

Telemetry system

Written By:
Adrien Deraes
Arno Jamsin
Bilal Tahini
Gonzalo Gómez
Nicolas Landreville
Javier de la Fuente
Lucas Prieels
Martin Segaert
Sebastián Reckziegel

Date: December 8, 2022

Contents

1 Introduction

In the previous years, the car prototype didn't contain any live telemetry system: it was possible to save some data, but only using an SD card in the car. Therefore data could not be analyzed live, and that was a limitation point during testing because it was difficult to adapt the driving strategy according to these data. Moreover, the number of sensors was quite small, and the resulting data were thus limited.

The main goals of the creation of the telemetry system for this year are the following :

- Create a live communication system allowing to receive data coming from the Arduino in the car. This transmission system should be *reliable* (small number of errors in the transmission), work with a *sufficiently large range*, and send a *large enough data rate*
- Implement a software on the receiving computer to treat data live as it arrives and represent it in an interpretable manner to the user
- Connect additional sensors in the car and send their data through the communication system

Even though the primary objective of the telemetry system is to send data to an interpreter outside of the car, it can also be improved by transmitting data to the driver. In this way the driver can adapt its driving strategy depending on live measurements, and have access to other useful information such as the charge of the battery or its speed.

The rest of this chapter will explain how this telemetry system has been created. The first section will show an outline of the telemetry system, and of the components it will need. The second section is going to show how the electronic components needed for the transmission have been compared and selected. Next, the implementation and construction of the telemetry system will be detailed, to help future students that are going to work with these components.

2 Outline of the telemetry system

Figure 2.1 shows the outline of the whole telemetry system. At the start of the year, only the Arduino Uno and one of the current sensors were already installed in the car.

2.1 Current and voltage measurements

All the sensors are located inside the car itself. One current sensor is used to measure the current flowing into the motor while the voltage over the motor is also measured by an analog port of the Arduino Uno. However, since the Arduino has a supply of 5V, it will naturally only be able to measure voltages up to this value. Since voltages to be measured are larger, they need a voltage divider which has been installed in the buck converter. Since the voltage applied over the motor is Pulse-Width Modulated (PWM), it is needed to measure it using a moving average value in the Arduino code. The maximum measure of the voltage over the motor is particularly important because it is the same value as the battery voltage, which can be directly correlated to the quantity of energy left in the battery as seen in section 4.

The combination of the measures of voltage and current in the motor give the instantaneous power given by $p(t) = v(t)i(t)$. This measure of power at the motor side is very useful because it allows to monitor the usage of electrical energy in function of time. That can be linked to

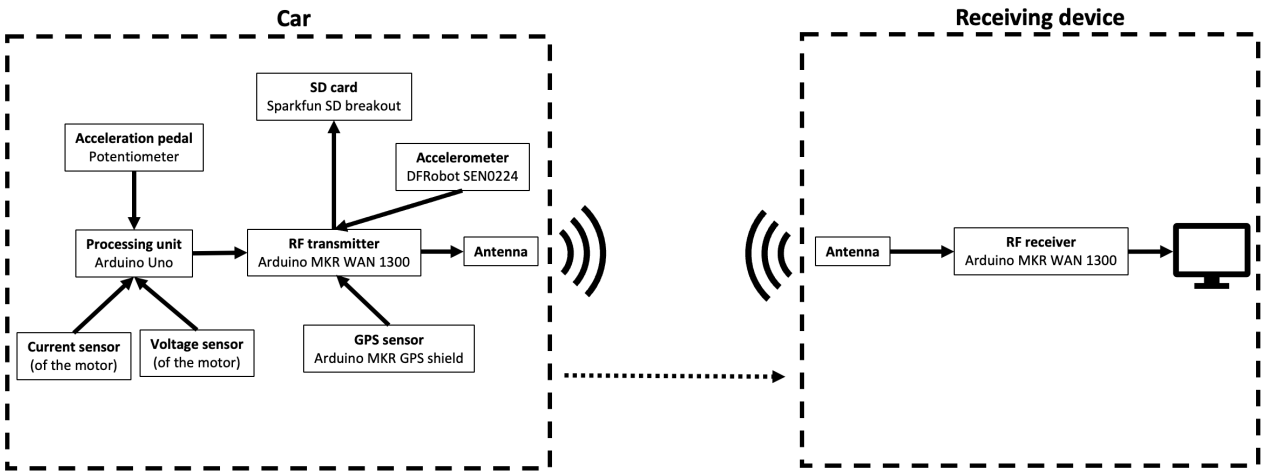

Figure 2.1: Global view of the telemetry system. The arrows show the direction taken by the flow of data

the behavior of the driver in order to define the best driving strategy as the one minimizing the usage of electrical energy to travel a specified distance.

2.2 Acceleration pedal

The pedal depression is how much the driver is pressing the pedal. This must be measured in order to study the link between this value and the speed of the vehicle as well as the quantity of energy spent. This is similar to a potentiometer: it is powered by a GND and a 5V line, and a third port is connected to an analog input port of the Arduino Uno.

2.3 Position, speed and acceleration

The link between the electric power consumed and the acceleration of the car is also interesting to study. Hence, an accelerometer has been placed among the electrical components in the car. In order to measure the position of the car on the track, a GPS sensor must as well be used. This will allow the team to monitor where the car is at all times, even when not in line of sight, which can be very useful in case of technical issues during the run.

It is also very important to measure the speed of the car, because the mean speed must according to the regulations be of at least 30km/h. Since such a measurement is very important, its values must be reliable and verified. To ensure its reliability, the speed will be computed independently using 2 separate components: the GPS sensor and the accelerometer. The GPS position is derived to get the speed, while the acceleration value is integrated.

It is expected that the speed derived from the accelerometer will be less precise because a small error on the acceleration value will have an effect on all the speed velocities computed afterward. More precisely, if $A(t_i)$ is the value of the acceleration at time t_i (without taking into account the vertical acceleration induced by gravity), the speed at time t_i is

$$S(t_i) = \int_0^{t_i} A(\tau)d\tau \approx \Delta t \sum_{j=0}^{i-1} A(t_j)$$

with $\Delta t = t_{i+1} - t_i$ the constant time interval between two samples, assuming that the speed at time $t = 0$ was 0.

On the contrary, the speed computed from the GPS position is only the derivative of these values. Therefore it is only needed to use the last few measurements and an error on a position value will only have an effect for a few seconds.

2.4 Wireless transmission

As it will be detailed in Section 3 about the choice of the components, the optimal solution for transmission is to use LoRa (long range) radio-frequency (RF) transmission in the Ultra High-Frequency (UHF) band, by opposition to using WiFi, Bluetooth, or cellular communication for example. To implement this on the car prototype, 2 Arduinos WAN 1300 mainly dedicated to communication have been added: one in the car as the transmitting device (Tx), and one outside the car to receive this data (Rx). As the accelerometer and the GPS sensor need a 3.3V supply, they are connected to this Arduino WAN, which has 3.3V digital pins.

For the Tx Arduino WAN to send data computed in the Arduino Uno such as the electrical measurements, a connection is made between them. It will be further discussed in Section 4 presenting details of implementation.

3 Choice of the electronic components

After giving a general overview of the telemetry system, this section will now dive in the details of the choice of each electrical component needed in the system.

3.1 Accelerometer

The accelerometer is one of the easiest component to choose. Indeed, 3 main points are to be considered: the current/power consumption, its precision, and its cost. With these factors in mind, the best choice seemed to be the DFRobot SEN0224, built around a LIS2DH12 chip [1]. Indeed, it is very low power (current usage of $2\mu A$), and has the advantage of having a settable precision by changing the measurement range (among $\pm 2g$, $\pm 4g$, $\pm 8g$, and $\pm 16g$, where g is the gravity acceleration). Moreover, it can have a very large output data rate (up to $5.3kHz$) which allows to read the acceleration value at a very high rate. Finally, its price very low as is it available for 5€ VAT included on Belgian websites.

This accelerometer also has the advantage of being able to read the acceleration value independently in the 3 axis. This means it will be easier to remove the contribution of the gravity acceleration for the calculation of the speed of the car.

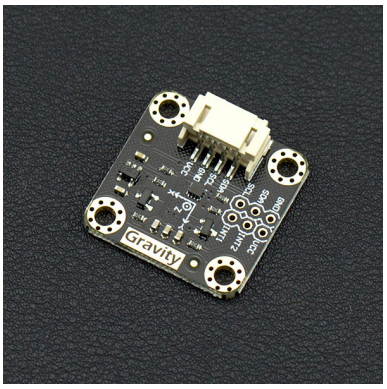

Figure 3.1: Accelerometer SEN0224. Source: dfrobot.com

3.2 GPS sensor

The GPS sensor that has been chosen is the Arduino MKR GPS Shield, based on the SAM-M8Q GPS module [2]. This choice has been done according to the following reasons, particularly important for this application :

- Compatibility with GPS, but also with GLONASS (Russian equivalent), and Galileo (European equivalent) which ensures a better reliability
- Moderate power usage
- Embedded antenna (saves costs and decreases the risks of breaking)
- Direct calculation of the speed, without needing to do the derivative ourselves
- Can contain a battery cell to save its settings and its configuration even when not powered. Since the startup time can be quite long (1 to 10 minutes) for configuration, it allows the GPS to skip this step when it has been disconnected from power supply for a short time
- Compatibility with Arduino WAN guaranteed (in terms of voltages, currents, pinout)
- Moderate price

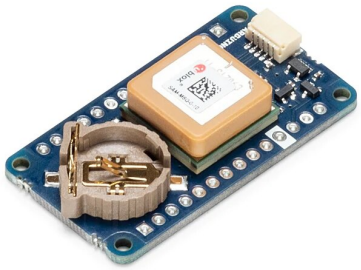

Figure 3.2: Arduino MKR GPS (GPS module shield). Source: arduino.cc

3.3 Transmission of data

The major part of the choice of the components is linked to the wireless transmission of data, because it has the largest importance in terms of power usage, costs, and it is critical to ensure a reliable communication with a large enough range and data rate.

The different communication technologies that have been considered are the following :

- WiFi
- Bluetooth
- SigFox radio communication
- "Classic" radio communication
- Cellular communication (GSM, 3G, IoT)

- LoRa radio communication (UHF band)

WiFi and Bluetooth quite clearly don't offer a sufficiently large range of communication to cover the entire track. They have therefore been quickly rejected.

SigFox is an Internet of Things (IoT) service provider, similar to a lightweight version of cellular communication (short messages, low-power, narrowband) [3]. However, it doesn't cover the entirety of Europe yet and since at this point the location of the race track was unknown, it was not possible to be sure a base station would be in range. Moreover, to comply with the sharing rules of the frequency bands used by SigFox in terms of duty cycle and power usage, there is a limitation of sending 140 messages of 12 bytes each per day. This is clearly not enough to cover for intensive usage that will be done irregularly on testing and competition days. This solution has therefore been rejected as well.

The last 3 types of communication could not be classified as a whole because they all have advantages and disadvantages. To compare them in a very complete and precise way, typical electrical components of each technology have been chosen and compared in Tables 1, 4, 2, and 3.

HC-12

The HC-12 is a small and lightweight component allowing to establish radio-frequency communication with another HC-12, through the use of 2 antennas. One HC-12 and one antenna must be connected to the Arduino Uno already in the car, and another HC-12 and antenna outside the car will receive all relevant data, and transmit them via a second Arduino (which must be bought) to a computer. This component can be used in several different modes, each having a different range, current usage, and bitrate. In table 1, mode FU4 has been chosen because it permits the largest range.

The main advantages of HC-12 is that it is cheap, lightweight, and very easy to use. However, it also has 2 big drawbacks: the range is a bit short compared to the track size, and on a policy point of view, the frequencies used by this component are legally limitating the power of transmission. Indeed, in the frequency allowed by the HC-12 the band that is usable without license with the largest allowed power is the $433.05 - 434.79$MHz in Belgium. But even for this band, the maximum allowed power is only 16.4mW EIRP (equivalent to 10mW ERP), and it is lower than the power needed to achieve the considered range. For this reason, the HC-12 has been rejected.

Type of communication	Radio-frequency communication
Range	1 to 1.8km in open-space (FU4 mode for long-distance)
Material needed and price	• 12€ for 1 HC-12 and antenna • Need 2 HC-12, 2 antennas and a new Arduino (24€ for a Uno) • $2 \times 12 + 24 = 48$€ in total
Current/power usage	• IDLE: 16mA, Max: 100mA (FU4 mode) • 3.2 to 5.5V supply $\Rightarrow$ 51 to 550mW
Bitrate	1200bps (FU4)
Interferences-prone	Subject to interferences
Frequencies usable	433.4 to 473MHz
Number of pins required	3 (supply and GND excluded)
Other advantages	Multiple modes exist, to have higher bitrates but lower range
Other drawbacks	Legal limitations on the transmission power
Sources	[4], [5], [6]

Table 1: Characteristics of HC-12

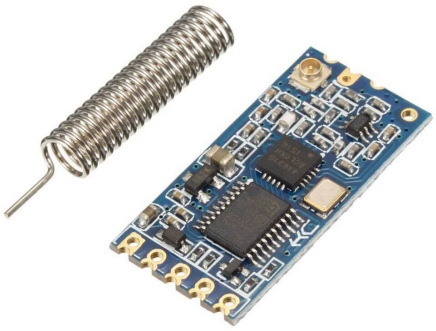

Figure 3.3: HC-12 component for radio-frequency communication. Source: arduino.cc

Arduino MKR GSM 1400

The Arduino MKR GSM 1400 is a component using cellular communication to transmit data. Just like the MKR WAN 1300/1310, and the MKR NB 1500, it is a real microcontroller, which means it could completely replaced the Arduino Uno. However, since the Arduino Uno is already at our disposal, it is best to keep the two board in parallel because it creates more free pins, more computing power, and simplifies the circuit ry. Moreover, the Uno has 5V digital pins while all the 3 MKR board have 3.3V digital pins, combining the two allows to choose among a broader range of components that can be used.

The particularity of the MKR GSM 1400 board is that it uses cellular (GSM or 3G) communication. That brings the advantages of having an unlimited range between Tx and Rx (as long as Tx is in range of any base station) and of allowing very large data rates. However, it also means the team has to buy a SIM card and regularly pay for a subscription, and most importantly, cellular transmissions use a huge amount of power. This board uses 500mA when

transmitting (with spikes up to $2A$), which translates as a power usage of $2.5W$ for the telemetry alone.

Considering that most teams achieve a result of $300 - 400$km/kWh in our category of the 2019 edition of the Eco-Marathon [7], and since the team is aiming to achieve a speed of $30 - 40$km/h, 10 hours of driving are needed to use 1kWh of energy. In this same time frame, continuously using the telemetry with a power of $2.5 - 10$W would have wasted $25 - 100$Wh, which can be as large as $2.5 - 9\%$ of the total energy usage ! This is of course too large, and for this reason the MKR GSM 1400 has been rejected as well.

Type of communication	GSM, 3G
Range	Unlimited as long as in range of a base station
Material needed and price	• $72 - 75$€ for GSM 1400 • 5€ for antenna • $0 - 3$€ for SIM card • $18 - 30$€ per year for subscription (10MB/month and service included) depending on the operator • Total: 80€ once + $18 - 30$€ per year
Current/power usage	• Max: $500 - 2000$mA • 5V supply $\Rightarrow 2.5 - 10$W
Bitrate	few MBps
Interferences-prone	Very low
Frequencies usable	850MHz for GSM
Number of pins required	2 but offers 18 new pins
Other advantages	• Don't have to consider frequency bands policy • It is a proper separate microcontroller, so it offers more pins and more computing power, and it simplifies circuitry
Other drawbacks	If the track is not in range of GSM, no communication can be made
Sources	[8], [9], [10]

Table 2: Characteristics of Arduino MKR GSM 1400

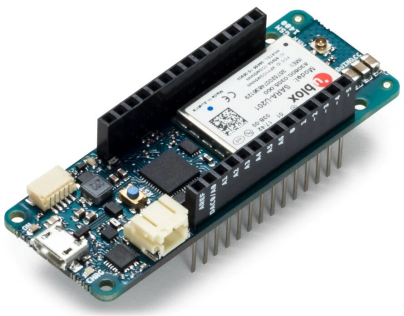

Figure 3.4: Arduino MKR GSM 1400 component for cellular communication. Source: arduino.cc

Arduino MKR NB 1500

The Arduino MKR NB 1500 board is similar to the Arduino MKR GSM 1400. The big difference between them is that the latter uses cellular data such as GSM or 3G, while the former uses lighter cellular data such as NB-IoT or LTE-M intended for Internet of Things (IoT) applications. This allows the board to save a lot of power, which becomes reasonable.

However, despite having the big advantages of having no limit on the range and a very large bitrate, this board has been discared as well. The main reason is because the coverage of NB-IoT and LTE-M is not 100%, and since the location of the track was not known at that time, it was not possible to check it was in range of a base station. Even if it was unlikely, not being in range would mean the telemetry would not work at all. Moreover, the limitation of data usage per month and the recurring fee for the Internet subscription are important drawbacks as well.

Type of communication	NB-IoT, LTE-M
Range	Unlimited as long as in range of a base station
Material needed and price	• $81 - 88$€ for NB 1500 • 10€ for antenna • $0 - 3$€ for SIM card • $10 - 20$€ per year for subscription (10MB/month and service included) depending on the operator • Total: 95€ once + $10 - 20$€ per year
Current/power usage	• Max: $60 - 190$mA • 3.3V supply $\Rightarrow 200 - 600$mW
Bitrate	$20 - 375$ kBps
Interferences-prone	Very low
Frequencies usable	$700 - 2100$ MHz
Number of pins required	2 but offers 18 new pins
Other advantages	• Don't have to consider frequency bands policy • It is a proper separate Arduino, so it offers more pins and more computing power, and it simplifies circuitry
Other drawbacks	• If the track is not in range of base station, no communication can be made • Coverage of NB-IoT and LTE-M is not as good as GSM • NB-IoT is intended for not moving applications and has a latency of few seconds, so LTE-M is preferred
Sources	[11], [9], [10], [12]

Table 3: Characteristics of Arduino MKR NB 150

Figure 3.5: Arduino MKR NB 1500 component for "light" cellular communication. Source: arduino.cc

Arduino MKR WAN 1300/1310

Finally, the board that has been chosen is the Arduino MKR WAN 1300 (or 1310). Just like the HC-12, it uses radio communication but it has a larger range and is quite immune to interferences. Like the other MKR boards, it is a microcontroller in itself, so it creates more pins and has more computing power. Moreover, it uses frequencies with larger allowed transmission power: the frequency band $869.4 - 869.65$MHz can be used with a maximum ERP of 500mW, which is large enough for this communication (with a duty cycle of 10%). It is slightly more expensive, and uses a non-negligible power, but it stays very reasonable.

The main difference between the MKR WAN 1300 and 1310 is that the 1310 uses a bit less of power. However, since it was not in stock for a few weeks, the 1300 version has been ordered. Even considering the maximum range, the maximum current drawn and a duty cycle of 100%, the maximum energy usage for continuous telemetry during 10 hours of driving should be 4.2Wh, less than 0.5% of the total energy usage of 1kWh.

Type of communication	LoRa (Long Range)
Range	Typically 15km in line of sight, $2-5$km in urban
Material needed and price	• $42-44$€ for WAN 1300 • $42-53$€ for WAN 1310 • 5€ for antenna • Need 2 WAN and 2 antennas: $2 \times 42 + 2 \times 5 = 94$€ total
Current/power usage	• Max: 128mA (for max range) • 3.3V supply $\Rightarrow$ max 422mW
Bitrate	300 to 5500 bps (depending on the SF)
Interferences-prone	Quite immune
Frequencies usable	868 and 915MHz
Number of pins required	1 but offers 18 new pins
Other advantages	• Multiple modes (called spreading factors, or SF), to have higher bitrate but lower range • It is a proper separate microcontroller, so it offers more pins and more computing power, and it simplifies circuitry
Other drawbacks	/
Sources	[13], [14], [15]

Table 4: Characteristics of Arduino MKR WAN 1300/1310

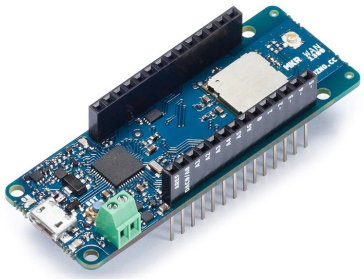

Figure 3.6: Arduino MKR WAN 1300 component for LoRa radio-frequency communication. Source: arduino.cc

4 Implementation and construction of the telemetry

4.1 Electrical values sensors and UART connection

To calculate the electrical power usage by the motor in real-time, the motor voltage and current must be measured by the Arduino. These measurements are done by the PCB designed by the control team and are translated into voltages between 0 and 5V and are directly connected to the analog inputs of the Arduino Uno. Since the motor voltage is a PWM value, the code must calculate its moving average to compute the mean voltage applied to the motor. The Arduino also records how much the pedal is pressed by the driver, as it is a very important measurement to analyze the driver's behavior and decide on a driving strategy. Then, these 3 voltage measurements are digitalized and transformed back into the voltage, current, and pedal

depression values.

In order to send the digitalized measurements towards the RF receiver and the person analyzing the data in real time, it first has to be transmitted to the Arduino MKR WAN in the car, which is the RF transmitter. This transmission can be done via 3 different communication protocols: UART, I2C, and SPI. However, as it will be explained in the next subsections, SPI must be used by the SD card writer, and the Arduino WAN is already an I2C master for communication with the accelerometer. Therefore it can't be used as an I2C slave for the communication with the Arduino Uno at the same time. It implies that the chosen communication protocol between the Arduinos is UART.

To make an UART connection, the Tx pin of the Uno is connected to the Rx pin of the WAN. Then, the digital values are sent very easily using the command *Serial.write(...)*. The only difficulty is that UART frames are 8 bits long, while measurements done by the Arduino are encoded on 10 bits. It is possible to send only the first 8 bits, but to avoid losing precision on these important values, the measurements are simply sent on 2 UART frames (one containing the first 8 bits, then one containing the last 2).

Since the Arduino WAN takes a variable amount of time between 2 readings on its UART Rx pin, it is possible that the sending frequency of the Uno is larger than the WAN reading frequency. And since data detected on the WAN Rx pin is stored in a queue before being read, it is possible that data starts to accumulate, and that the WAN reads data that was sent a while ago by the Uno. To avoid reading outdated values, the WAN discards all data left in the UART queue after each reading.

4.2 Accelerometer

The accelerometer comes with an Arduino library called *DFRobot_LIS2DH12.h* and programming it is therefore very straightforward. It uses I2C communication protocol and has a 4-pin connection: GND and VCC are directly connected to GND and VCC pins of the WAN as the accelerometer requires a 3.3V supply. The SCL and SDA pins of the accelerometer are respectively the clock and data connections used by the I2C protocol, and must be connected respectively to digital pins 12 and 11 of the WAN as they are the only ones compatible with I2C communication [15].

As stated before, the accelerometer was planned to be an alternative way to calculate the speed of the car. However, tests showed that it was not reliable at all for 2 reasons. The first one is that the resolution of the acceleration measurements was not small enough. The second one is that it is needed to remove the contribution of gravity acceleration to calculate the speed from the acceleration. It is theoretically possible because the accelerometer gives the acceleration components in the 3 directions x, y, and z, but it is difficult to achieve in practice because as soon as the accelerometer is somewhat tilted, the contribution of gravity is seen not only on the vertical axis z but also on x and y.

4.3 GPS sensor

Just like the accelerometer, the GPS sensor can be very easily programmed using the Arduino library *Arduino_MKRGPS.h*. However a dive in the code of this library showed that each time the GPS is polled using the function GPS.available(), only one byte of data is returned. As data coming from the GPS is several hundreds of bytes long, it is needed to poll the GPS

hundreds of times before being able to read just once the current coordinates of the car. In the Arduino code, the GPS is therefore polled continuously until GPS.available() returns 1, which means the location has been updated. To avoid being stuck in an infinite loop when the GPS detects no signal, there is a limit of 5000 polls after which the location update is skipped.

The GPS shield can be used either with UART or I2C. UART being already in use for the communication between the Arduinos, and since I2C protocol (already used with the accelerometer) supports multiple slaves, it has been connected to the pins 12 and 11 of the WAN as well. The GPS also has an external interrupt pin, which must be connected to the digital pin 7.

4.4 SD card

In order to make sure important data is recorded even if the RF communication is not working anymore, it is stored on an SD card before being wirelessly sent. For this, the SD card module that has been used for some time last year is again utilized: it is an SD card breakout made by Sparkfun and it requires SPI connection. SPI connection requires the next 4 wire connections :

- CLK (on SD breakout) connected to pin 9 of WAN (SCK or Spi ClocK) for the clock of the communication
- CMD connected to pin 8 (MOSI or Master Out Slave In) for the orders sent by the master (WAN) to the slave (SD breakout)
- D0 connected to pin 10 (MISO or Master In Slave Out) to send data from the SD card to the WAN
- D3 connected to any digital pin (here pin 6 has been chosen), for the slave to indicate when it is ready to communicate

The implementation in the code is again straightforward using the libraries SD.h and SPI.h. To avoid erasing previous data, the code makes sure to use a file name that doesn't exist on the SD card yet, and it writes data as a .csv file to ease interpretation and reading.

4.5 RF connection

The solution chosen to send data over the wireless connection using the library *Lora.h* is to define a C structure containing all the last recorded value of all relevant data. This structure is transformed in bytes, then sent over the Tx antenna, recieved at the Rx antenna and transformed back into the structure.

With LoRa communication, a parameter that can be tuned is the **spreading factor** and can be between 7 and 12. When the spreading factor (SF) is larger, the range of communication is larger but the maximum bitrate is lower. During tests the SF has been fixed to 8 because the range was already large enough, but it can very easily be switched to a larger value on the competition day, in order to ensure the range will be large enough to cover the entire track.

Because of this spreading factor the bitrate can be a limitating factor. In order to ensure a refresh rate as high as possible, the number of bits used to code each variable has been optimized as much as possible, while ensuring they still have a satisfying precision. With $SF = 8$, the refresh rate is between 3 and 5Hz while with $SF = 12$, the refresh rate is around $0.4Hz$. If a larger refresh rate is needed, it is possible to decrease even more the number of bits per variable

but in that case some precision is lost. For example, the motor voltage value can be stored on 8 bits instead of 10, but then its resolution is 20mV instead of 5mV.

4.6 Real-time updating graphs

To have a clear view of the state of the car at all times, the best way is to plot graphs that are updated in real-time. This can be done using the software Serial Studio [1]. To let the software know what values and graphs should be shown on the telemetry panel, a JSON file is written. It contains the outline of the telemetry panel wanted, the name of values that must be shown, their units, and whether these values must be plotted in a graph. Then the Rx Arduino just has to send received data via Serial communication to the computer, and the software takes care of plotting the graphs.

As shown on Figure 4.1, the telemetry panel first shows all the values that are measured: the left part shows the status of the telecommunication by printing the ID number of the last received data packets, and the time at which this last packet has been measured (on a 12-hour clock). The middle part shows the electrical values directly measured, or derived as it will be discussed in 4.7, the potentiometer being the amount of pressing on the pedal. The right part shows the measured speed in real-time, the average speed on the last 10 measurements (to smoothen the graph), and the average speed measured since last reset. That is useful to check that the car is going faster than 30km/h in average. After these real-time updated values, there are multiple plots of useful data to shows how they change in time. Finally, a satellite map shows the position of the car.

Finally, the measured values are also stored in a CSV file to analyze them later on, and they can be uploaded in the software to replay the evolution of the graphs afterwards. Even if the wireless transmission fails, data can still be analyzed after the tests because the CSV file written on the SD card can be uploaded in the same way in the software.

4.7 Derived calculations

Some values are very important to define a driving strategy, like the real-time power usage or the total distance traveled. Despite having no sensor for these values, they can be derived from values already measured :

- The electrical power used by the motor can be calculated as the product of the motor voltage and the motor current: $p(t_i) = v(t_i) \times i(t_i)$
- The total electrical energy used is calculated by integrating the electrical power used $e(t_i) = \sum_{i=1}^{t_i} p(t_i)(t_i - t_{i-1})$ assuming $e(0) = 0$
- Since the GPS position is only precise to a few meters, the speed is quite noisy. A smooth speed is thus derived as the moving average of the last 10 samples of the speed $s_{smooth}(t_i) = \frac{1}{10} \sum_{j=0}^{9} s(t_{i-j})$
- The average speed since last reset is directly calculated as the average of the speed since the beginning of the measurements $s_{avg}(t_i) = \frac{1}{i+1} \sum_{j=0}^{i} s(t_j)$
- The total distance traveled is the integral of the speed $d(t_i) = \sum_{j=1}^{i} s(t_j)(t_j - t_{j-1})$

[1] https://serial-studio.github.io

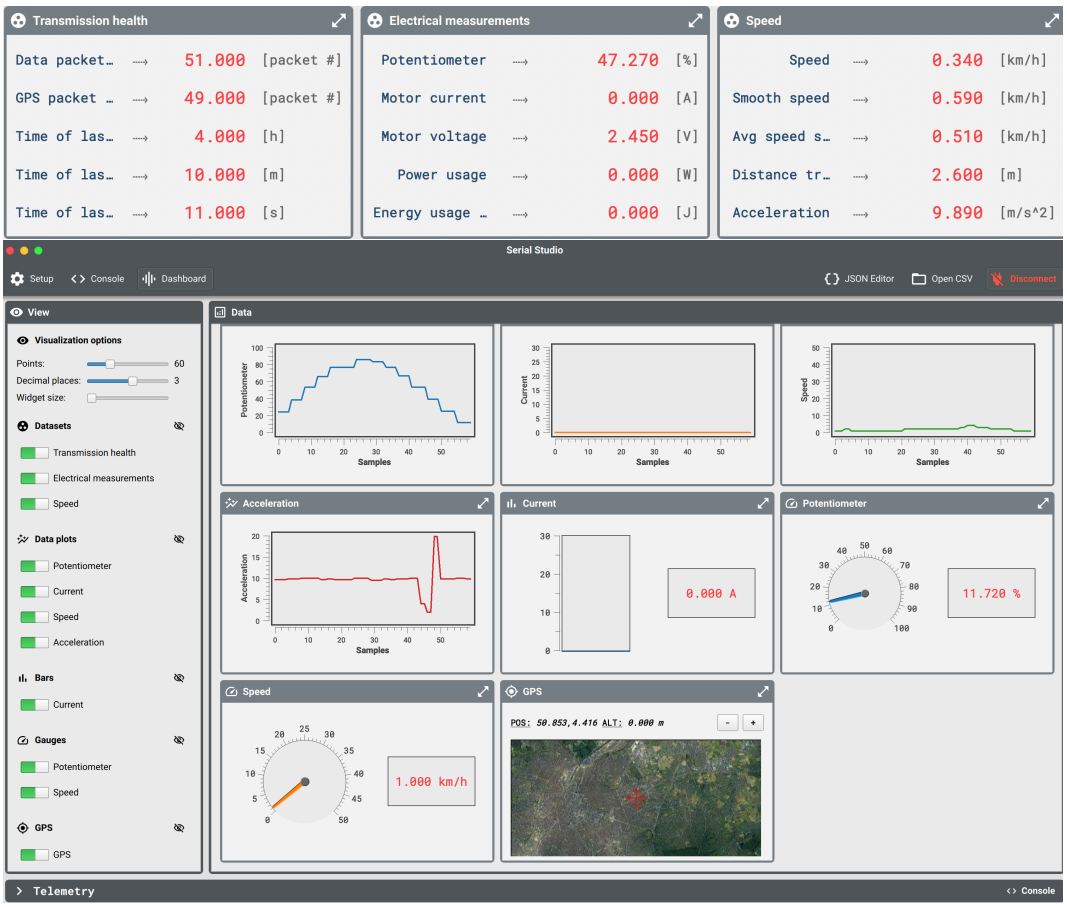

Figure 4.1: Telemetry panel updated in real-time on the computer that tracks the car

4.8 Electrical connections

Figure 4.2 sums up the entirety of electrical connections between the components that belong in the car.

5 Conclusion and possible improvements

The telemetry system has been strongly improved this year. From a system that simply recorded the value of a couple electrical values on an SD card last year, an extensive telemetry system has been designed and tested. This system allows to make the following measurements :

- How much the pedal is pressed
- Current flowing into the motor
- Voltage over the motor
- Power and energy used by the motor
- GPS position of the car (longitude, latitude, and altitude)
- Speed of the car, as well as a smooth version
- Average speed since last reset
- Distance traveled

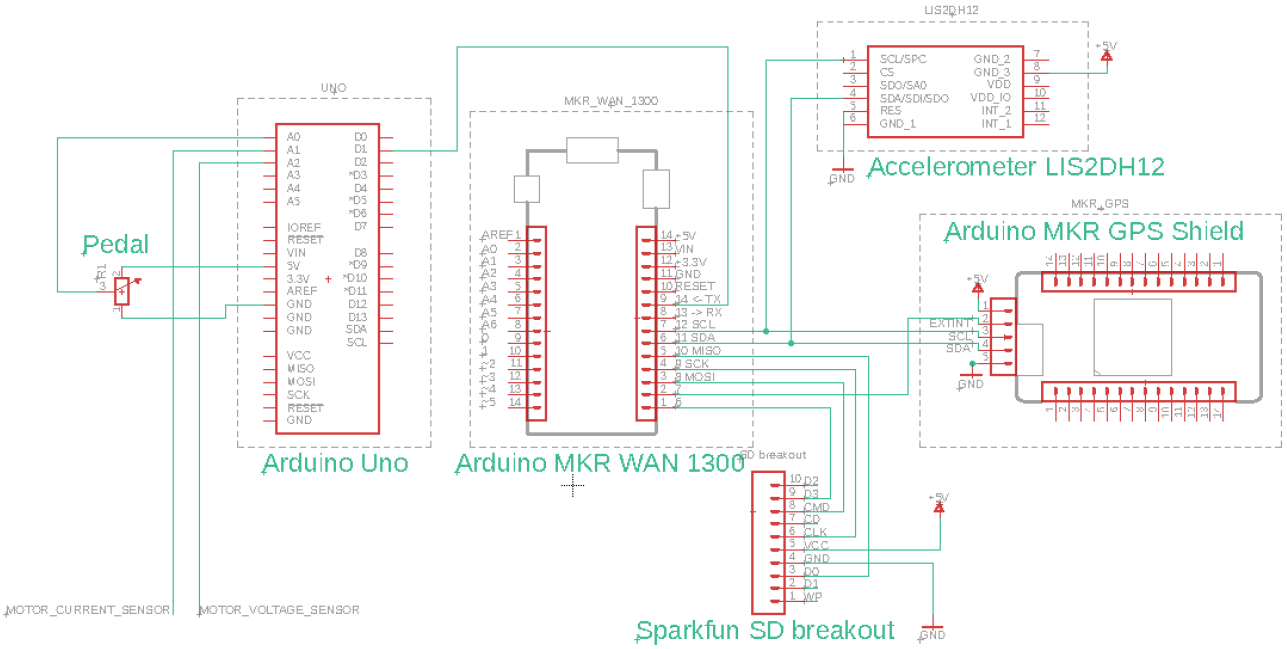

Figure 4.2: Summary of the electrical connections made between the components in the car

- Acceleration of the car
- Current time (given by the GPS sensor)

All these measurements are both stored on an SD card and sent via a wireless connection to a computer, with a couple kilometers of range in urban environment. The computer can analyze this data using raw values and plots, which allows the team to analyze and correct the driver's behavior in real time. All data are stored both on the SD card and on the computer, to make sure no data is lost.

Despite this extensive and fully-working telemetry, unfortunately no analysis of the data collected could have been done yet. Indeed, first tests in real situation planned on a track in Nivelles have not yet been performed at the time this report has been handed over. After these tests will have been performed, an in-depth analysis of collected data will allow the team to define an optimal data-driven driving strategy, which will likely save a big amount of energy during the competition.

One major improvement that can still be done to the telemetry is giving a feedback in real-time to not only the team outside the car but to the driver as well. Indeed, on the competition day the driver has to be able to correct its behavior in real time by directly giving him or her access to the data. It would also support the driver's decision-making in tricky or unexpected situations. This can for example be done by adding a Bluetooth module to the Arduino, and by developing a mobile application to receive the most important data directly on his smartphone.

References

[1] *MEMS digital output motion sensor: ultra-low-power high-performance 3-axis femto accelerometer*, Mouser, 5 2017, rev. 6. [Online]. Available: https://www.mouser.be/datasheet/2/830/en_DM00091513-2488668.pdf

[2] *Easy-to-use u-blox M8 GNSS antenna module Data Sheet*, U-Blox, 2 2019, rev. 5. [Online]. Available: https://core-electronics.com.au/attachments/localcontent/Arduino-MKR-GPS-Shield_u-blox8-SAM-M8Q_DataSheet_UBX-16012619_94827a2c23a.pdf

[3] Sigfox. Sigfox website. [Online]. Available: https://www.sigfox.com/en

[4] *HC-12 Wireless Serial Port Communication Module*, Electrow, rev. 1. [Online]. Available: https://www.elecrow.com/download/HC-12.pdf

[5] *HC-12 WIRELESS RF UART COMMUNICATION MODULE*, HC tech, 12 2016, rev. 2.4. [Online]. Available: https://statics3.seeedstudio.com/assets/file/bazaar/product/HC-12_english_datasheets.pdf

[6] B. I. for Postal services and Telecommunication. Frequency bands usage (band 433.05-434.79mhz). [Online]. Available: https://www.bipt.be/file/cc73d96153bbd5448a56f19d925d05b1379c7f21/6910c550188818dfe58c709383d142fb8f2ed801/B-EN.pdf#page=77

[7] S. Eco-marathon. Shell eco-marathon europe 2019. final results : Prototype battery-electric. [Online]. Available: https://www.shell.com/make-the-future/shell-ecomarathon/europe/results-and-awards/_jcr_content/par/expandablelist_781025105/expandablesection.stream/1563371146355/3ffc055693403502ab9323df9ff51f0dea80a2ef/europe-2019-prototype-battery-electric-full-results.pdf

[8] Arduino. Arduino mkr gsm 1400. [Online]. Available: https://docs.arduino.cc/hardware/mkr-gsm-1400

[9] Enmify. Iot data plans and pricing. [Online]. Available: https://www.emnify.com/pricing

[10] Orange. Orange plus subscription pricing. [Online]. Available: https://orange-iotshop.allthingstalk.com/product/nb-iot-plus/?v=d3dcf429c679

[11] Arduino. Arduino mkr nb 1500. [Online]. Available: https://docs.arduino.cc/hardware/mkr-nb-1500/

[12] Orange. Orange products and services. [Online]. Available: https://business.orange.be/en/it-solutions/internet-things/products-and-services-internet-things-live-objects

[13] LoRa. Lora documentation. [Online]. Available: https://lora.readthedocs.io/en/latest/

[14] L. interferences. Avoiding rf interference with lora. [Online]. Available: https://lora-developers.semtech.com/documentation/tech-papers-and-guides/interference-immunity/

[15] *Arduino MKR WAN 1300 (LoRa Connectivity)*, Arduino, 1 2018. [Online]. Available: https://www.mouser.com/datasheet/2/34/Arduino_01172018_ABX00017-1286590.pdf