ENSTA Bretagne

Second-year internship report

Development of a tracking system for a remote-controlled car

Alexandre PIOT
Engineering student in Vehicle Architecture at ENSTA Bretagne
Research Assistant Intern at TU Chemnitz

October 19, 2019

Acknowledgment

I would like to express my warmest thanks to Vladimír BUDAY, researcher at TU Chemnitz, for welcoming me and giving me the necessary advice and guidance during my internship. I would also like to thank Maxime WACH, third year student at ENSTA Bretagne and intern at TU Chemnitz, René MORITZ and Martin BIÁK, researchers at TU Chemnitz, for their support, help and interest in my project.

Abstract

Tracking technologies are necessary for land transport for several tasks such as guidance systems, autonomous driving, trajectory optimization. These technologies always combine sensors, more or less expensive and efficient, and algorithms for data fusion, more or less complex and voluminous. The aim here is to develop a tracking system for a RC car in order to optimize its trajectories during the race with certain constraints. Indeed, Arduino components have been used with their disadvantages such as connections not always reliable, noisy data and limited memory capacities. To obtain real time results, data are transmitted as quickly as possible from a transmitter on the vehicle to a receiver connected to a computer where they are merged. The implementation of a fusion algorithm for triaxial orientation coupled with a GNSS receiver has resulted in a tracking system that could be used in a passenger car, but improvements are needed to use it in an RC car.

Résumé

Les technologies de suivi sont nécessaires au transport terrestre pour plusieurs tâches telles que les systèmes de guidage, la conduite autonome, l'optimisation des trajectoires. Ces technologies combinent toujours des capteurs, plus ou moins coûteux et efficaces, et des algorithmes de fusion de données, plus ou moins complexes et volumineux. L'objectif est ici de développer un système de suivi pour une voiture radiocommandée afin d'optimiser ses trajectoires en course avec certaines contraintes. En effet, des composants bon marché compatibles Arduino ont été utilisés avec leurs inconvénients tels que des connexions pas toujours fiables, des données bruitées et des capacités mémoire limitées. Pour obtenir des résultats en temps réel, les données sont transmises le plus rapidement possible d'un émetteur sur le véhicule à un récepteur connecté à un ordinateur où les données sont fusionnées. La mise en œuvre d'un algorithme de fusion pour l'orientation triaxiale couplé à un récepteur GNSS a abouti à un système de suivi qui pourrait être utilisé sur une voiture particulière, mais des améliorations sont nécessaires pour l'utiliser sur une voiture radiocommandée.

Contents

Introduction

Work environment. Chemnitz University of Technology, located in Saxony, Germany, has a strong expertise in automotive engineering and hosts important research institutes, especially in automotive software engineering and in alternative source of energy for vehicles. This internship took place in the Alternative Energy Vehicle Department (Alternative Fahrzeugantriebe) of the Faculty of Mechanical Engineering. This department, led by Prof. Dr.-Ing. Thomas VON UNWERTH, focuses on sustainable solutions for vehicle's propulsion, especially hydrogen propulsion based on fuel cells. This laboratory houses, for example, fuel cells test benches and a chemistry laboratory. The research department works in partnership with the start-up FCP (Fuel Cell Powertrain) specialized in the design and optimization of complete hydrogen powertrains for vehicles. The infrastructure and laboratories are accessible to companies, which can rent the premises and benefit from state-of-the-art equipment. Even if this research department is located in a university, the working atmosphere could be compared to that of a start-up insofar as there are a small, fairly close-knit staff and no strictly fixed schedules. Priority is given to the completion of tasks and the working environment is designed to be friendly. The multicultural aspect is very present due to the many nationalities among researchers and trainees (German, Czech, Indian, Chinese, French, etc.) which makes the use of English very easy.

Project context. In addition to a first project which consisted in creating a test bench for a hydrogen RC car carried out by Maxime WACH, this study was carried out on the tracking of the same RC car. The final goal is to optimize driving during the race, which requires a test bench to obtain information on the energy consumption and real-time information on the trajectory. Further projects for this car would be to implement a torque vectoring system and to make it autonomous.

Objectives and challenges. The main goal of this internship is to choose suitable and inexpensive sensors and to use them in sensor fusion algorithms in order to calculate orientation and position estimation of a ground vehicle. The goal is not to create the most accurate system but to understand how these technologies work together and to establish a protocol that describes how to use the hardware and the computer codes presented in this report. Another objective is to do as many things as possible from scratch to gain knowledge and show that a restricted budget can be enough to obtain satisfying results. In addition to low cost equipment, other technical challenges are the car's radius of movement, which requires a high-resolution and highly responsive tracking system, and the size of the car, which requires the design of a compact and lightweight system. This study was conducted independently but always with the possibility to seek advice from researchers and other students.

Relevance in the professional project. It must be noticed that tracking technologies and trajectory control systems are crucial problems for recent technologies including active safety, autonomous driving, drones, tracking in video games, in augmented reality/virtual reality. These technologies are at the crossroads of information technologies, sensor technologies, signal processing, artificial intelligence. Mastering them or at least understanding them is an undeniable advantage for a future automotive engineer and a strong point to put

forward to potential recruiters. A first work experience abroad after a one-year exchange in Prague also strengthens my profile for a first position outside France after graduation.

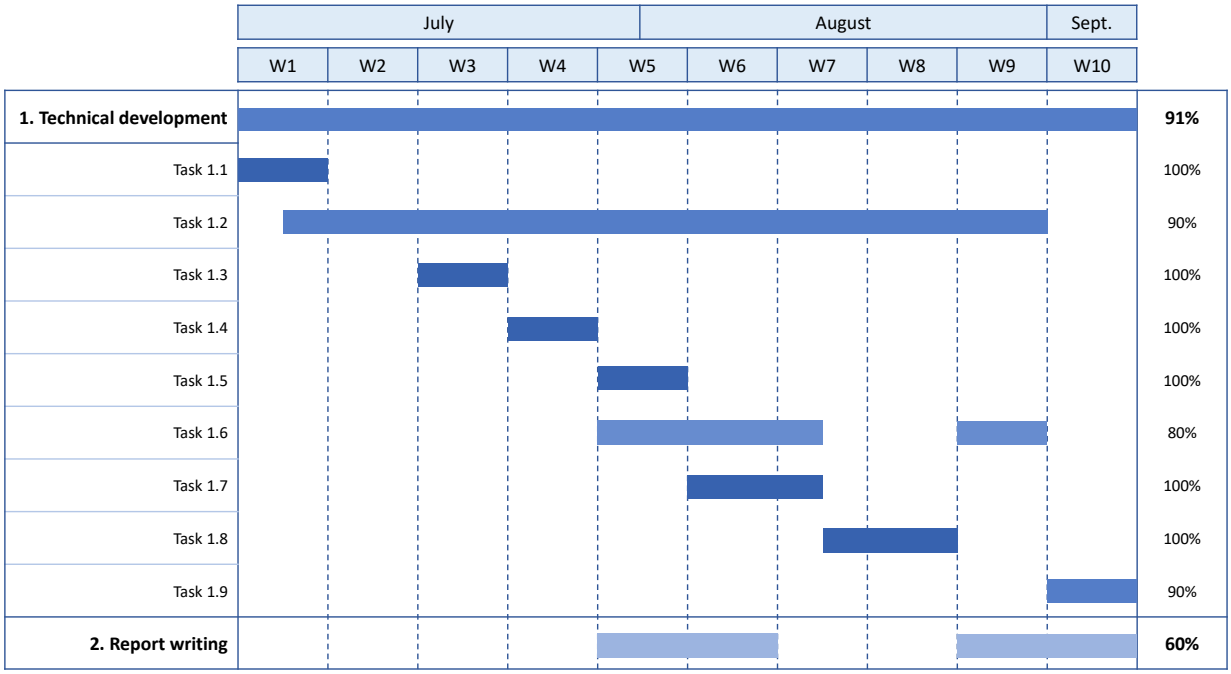


Figure 1: Gantt chart

1.1	Target objectives of the project
1.2	Find information (sensors for tracking, sensors fusion, data protocols…)
1.3	First manipulation of a MPU9250, use Arduino and MATLAB libraries to obtain first orientation results
1.4	Arduino program for the MPU9250 from scratch
1.5	Calibration procedure
1.6	Fusion algorithms (complementary filters, Madgwick, Kalman)
1.7	RF communication
1.8	Include a GNSS receiver in the assembly, extract data from it
1.9	Carry out tests

Figure 2: List of tasks

1 Hardware choices and testing procedures

In this first part are presented the components used in this project, the reason for their choice, their specification and how to test them individually before we begin to use them together.

1.1 Microcontrollers

This project requires two microcontrollers, one for the control of the transmitter block to which are connected the sensors and one for the receiver block that receives data from the transmitter.

The transmitting microcontroller must be small and light, have male pins so that it can be directly connected to a breadboard, have enough flash memory and SRAM, have a $3.3V$ and $5V$ power output, have UART, SPI and I^2C buses. The **Arduino Nano** meets these requirements and so has been chosen. It has an processor ATmega328P with a $16MHz$ frequency, the required ports, a $32kB$ flash memory and a $2kB$ SRAM.

The receiving microcontroller has less requirements insofar as it will be only used to transfer data to the computer. It must have enough flash memory and SRAM, have $5V$ power output and have an SPI bus. The **Arduino Uno** is a versatile microcontroller that meets these requirements and so has been chosen. It has the same processor as the Nano.

1.2 Sensors for orientation and position estimation

Different sensors can be involved in tracking technologies depending on our goals and requirements. Their choice depends on the type of estimation (relative or absolute) we are looking for. A relative position and orientation estimation only requires the changes in sensors output values from an initial position, which can be done with an Inertial Measurement Unit made up of an accelerometer and a gyroscope. An absolute estimation requires more devices to be used as outside observers such as a magnetometer or a GNSS receiver. This can be also done with environment detection system (camera, radar, lidar) [1]. The choice of the sensors and their parameters depends also on the context of measurements : tracking of slow or fast motions, noise, short-term or long-term tracking.

1.2.1 Inertial Measurement Unit (IMU)

Technology. The **MPU9250** 9-DOF IMU combines the 3-axis accelerometer and 3-axis gyroscope of the MPU6050 (a lighter 6-DOF IMU generation) and the 3-axis magnetometer AK8963. It has a Digital Motion Processor (DMP) based on a acceleromter-gyroscope fusion algorithm but it will not be used in this study. The MPU9250 can communicate through an I^2C bus with the Arduino board at 400MHz (A.3). This sensor has the advantage to be inexpensive and its use is highly documented throughout many studies.

The MPU9250 uses microelectronmechanical systems (MEMS) to measure static acceleration (gravity) and dynamic acceleration (outside excitation) in three axes. Any change in

the acceleration along an axis is measured by a change of the capacitance between the fixed plates and the moving mass [2].

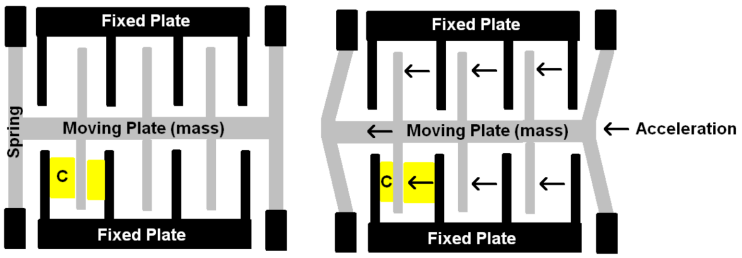
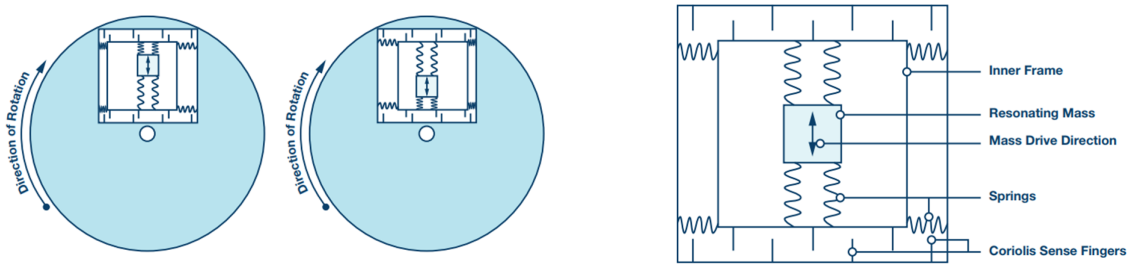


Figure 3: MEMS accelerometer principle [2]

In the same way, the sensor uses MEMS for the measurement of the rate of rotations in °/s around three axes (roll,pitch,yaw) using Coriolis effect. A resonating mass exerts a force on the frame to the right while moving toward the center of rotation and to the left while moving away from this same point. The displacement of this mass and the resulting force are translated into capacitance between the inner and outer frames of the sensor [3].



(a) MEMS gyroscope

(b) MEMS gyroscope details

Figure 4: MEMS gyroscope principle [3]

Finally, the magnetometer measures the neighboring magnetic field strength and direction along three axes. This sensor uses Hall effect. The electrons move in a conductive plate immersed in a magnetic field that exerts Lorentz force on the electrons [4] as $\vec{F} = q(\vec{E}+\vec{v}\times\vec{B})$, where q is a particle of charge, v its velocity, E the electric field and B the magnetic field. It allows us to get access to absolute heading of the system on which is fixed the IMU.

The MPU-9250 uses three 16-bit analog-to-digital converters (ADCs) for digitizing the gyroscope outputs, three for the accelerometer outputs, and three for the magnetometer outputs. The full-scale ranges of the different sensors can be adjusted by manipulating their dedicated registers. The possible choices are 250, ±500, ±1000, and ±2000°/s for the gyroscope, ±2g, ±4g, ±8g, and ±6g ($1g = 9.81 m.s^{-2}$) for the accelerometer. Magnetometer's range is set at $\pm 4800 \mu T$ but binary output length can be chosen (14 bits or 16 bits). Because the output raw values are given in arbitrary units, these parameters are used to calculate the resolution of the three sensors in order to obtain proper units. The internal sample rate of the sensors can be adjusted as well as the accelerometer and gyroscope built-in low-pass filters. Finally, a sample rate divider can be set to reduce the working frequency of the system [6].

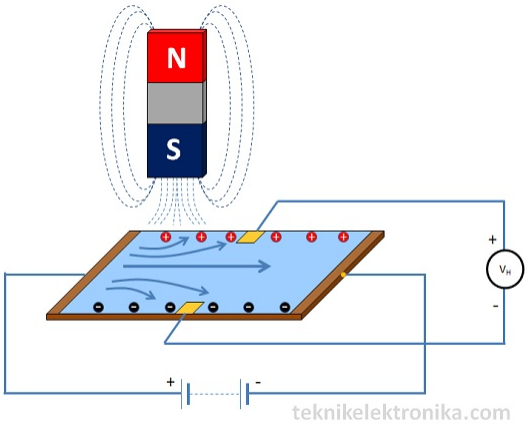


Figure 5: Hall effect sensor [5]

Test procedures. The use of an IMU requires three test procedures. The first test program checks the communication through the I^{2}C port and is called *test_I2C_scanner.ino*. This program tries to communicate with all the available addresses from 0x08 to 0x7B and displays the ones that answered positively. For the MPU-9250, we are supposed to find 0x0C for the magnetometer and 0x68 for the accelerometer and gyroscope.

```
-> ---Scanning---
-> 0x0C
-> 0x42
-> 0x68
-> 3 devices found on the bus
```

Figure 6: I^{2}C scanner output

The second and third tests are gathered in the file *test_detect_and_read.ino*. This program first performs a self-test of the IMU that returns the percentage of deviation from the factory trim values for the accelerometer and gyroscope.

```
-> ---Self Test---
-> Put the device on a flat surface and do not move it
-> x-axis self test: acceleration response within : 3.9% of factory value
-> y-axis self test: acceleration response within : 0.7% of factory value
-> z-axis self test: acceleration response within : -0.5% of factory value
-> x-axis self test: gyration response within : 0.1% of factory value
-> y-axis self test: gyration response within : -0.1% of factory value
-> z-axis self test: gyration response within : 0.1% of factory value
```

Figure 7: IMU self-test output

1.2.2 GPS chip

Technology. GPS is the most common satellites-based navigation method. It is provided by the United States Government and Air Force. Signals are transmitted from a constellation

of satellites to GPS receiver that decodes received data into position, time, velocity and heading. Other global navigation satellite systems (GNSS) exist such as GLONASS (Russian army) and Galileo (European Union). A GNSS receiver needs at least four satellites to compute a 3D position, but they can fuse data coming from more satellites to compute a more accurate position. The GNSS receiver that is needed for this project must be as precise as possible for a low price, it must be small and light, an embedded antenna is preferable for reasons of compactness. The **Sparkfun SAM-M8Q** has been chosen. This GNSS receiver has a $2.5m$ position accuracy, a maximal update rate of $10Hz$, has an embedded patch antenna and weighs only $5g$. The communication protocol of the chip can be chosen among the NMEA 0183 standard, the RTCM standard or the proprietary protocol UBX (detailed further in **??**).

Test procedures. This component can be plugged to the Arduino board through I^2C or UART interfaces. In case it is connected to the I^2C bus, the I^2C scanner can be used as seen previously, and it should find the address 0x42. It has been decided here to use the I^2C interface with the library provided by Ublox, the manufacturer of the GNSS receiver which is integrated in the Sparkfun component. This test procedure for the GPS consists in reading the raw NMEA sentences and see if there is any anomaly or irregularity in the transmitted messages that can prevent the parser to extract proper information. The NMEA standard uses a simple ASCII, serial communications protocol that defines how data are transmitted in a "sentence" from one "talker" to one or several "listeners" at a time [7]. These NMEA sentences must be treated by a parser that can return all the information that can be returned according to the specifications of the chip (position, velocity, heading, altitude etc). The test file *test_raw_GPS_i2c.ino* must return a similar structure to what can be seen in Figure 8.

```
-> $GNRMC,100846.00,A,5048.96843,N,01255.77162,E,0.242,,300819,,,A*6D
-> $GNVTG,,T,,M,0.242,N,0.448,K,A*31
-> $GNGGA,100846.00,5048.96843,N,01255.77162,E,1,07,2.85,339.1,M,44.7,M,,*41
-> $GNGSA,A,3,10,15,20,21,,,,,,,,3.37,2.85,1.81*18
-> $GNGSA,A,3,86,71,72,,,,,,,,,3.37,2.85,1.81*11
-> $GPGSV,2,1,08,04,,,27,10,66,111,39,15,11,034,31,16,28,194,18*40
-> $GPGSV,2,2,08,18,25,263,,20,48,066,28,21,24,078,28,32,,,28*47
-> $GLGSV,1,1,04,71,53,082,36,72,34,154,19,86,51,080,39,,,,31*5E
-> $GNGLL,5048.96843,N,01255.77162,E,100846.00,A,A*73
```

Figure 8: Raw NMEA sentences

If the sentences are short or even empty, it means that the receiver does not receive data from enough satellites, which is likely to happen inside a building. A GNSS receiver will also takes more time to find again the satellites in what is called a "cold start" (several hours after the last use of the receiver).

1.2.3 Other sensors and their use

Ideally, this project could use more sensors. Useful sensors for a tracking project are a speedometer and a barometer. The first one gives the velocity of the vehicle based on the rotation of mechanical components (incremental sensor or Hall effect sensor) or based on the

power supply on the motor (detection of rising slopes in the current). It gives a better idea of the instantaneous speed than the speed from a GNSS receiver which has a lower frequency and transmits its position inaccuracy to the speed. A barometer would be useful if we want to work in 3D including the altitude. Indeed, the altitude calculated by a GNSS receiver is not stable and depends entirely on the number of available satellites, but on the other hand the barometer needs to be carefully calibrated because of its sensitivity to ambient pressure changes. For further projects described in the introduction such as autonomous driving, a lot of additional sensors will be needed such as lidar, radar, ultrasonic sensors, cameras. Because of the required computing power, the electronic architecture described in this study would be obsolete.

1.3 Data transmission components

Technology. Several technologies are possible for data transmission: simple RF communication, bluetooth, Wi-Fi. The first requirement for the communication system is its range that must be at least a few tens of meters (which corresponds to the dimension of the racetrack for the RC-car) with an omnidirectional communication. Then, it must be capable to send the biggest packet that we may have to send in the programs detailed further, which is a packet of 14 integers codded on 2 bytes, which means 28 bytes. The IMU will be used at a frequency of $200Hz$, which means that in the worst case we may need a transmission rate of $44.8kbps$. Finally, the equipment must remain at an affordable price. A set of "transceiver" (two in one transmitter and receiver) **nRF24L01** equipped with an SMA connector and a duck-antenna has been chosen. It has a 1000m range in good conditions, a maximal packet length of 32 bytes, a maximal transmission rate of $2Mbps$ ($250kbps$, $1Mbps$, or $2Mbps$) and different output powers ($0dBm$, $-6dBm$, $-12dBm$ or $-18dBm$). It operates on the 2.4 GHz ISM band and has 125 frequency channels. These RF modules requires an SPI interface on the microcontrollers that will be used. It has also been decided to add **adapters modules** between the transceivers and the microcontrollers in order to stabilize the power supply and and to ease the pin management of the RF modules. The adapters are supplied with $5V$ current contrary to the RF modules that must be supplied with $3.3V$ when they are used alone.

Test procedures. It is necessary to test both transmitter and receiver modules to see if they share the same communication parameters. We also check if they are capable to detect surrounding communications in $2.4GHz$. The test procedure for a RF module is located in *test_RF_scanner.ino*. The library RF24 offers a method *printDetails()* that gives the parameters of the module.

The second part of the test is a scanner of the $2.4GHz$ band. All the channels are tested one by one and the program returns the occupancy rate of the channel from 0 (empty) to F (saturated). In an environment such as an office building with Wi-Fi it would be a sign of failure if the scanner returns 0 for all the channels.

```
-> RF24/examples/scanner/
-> STATUS           = 0x0e RX_DR=0 TX_DS=0 MAX_RT=0 RX_P_NO=7 TX_FULL=0
-> RX_ADDR_P0-1     = 0x544d52687c 0xabcdabcd71
-> RX_ADDR_P2-5     = 0xc3 0xc4 0xc5 0xc6
-> TX_ADDR          = 0x544d52687c
-> RX_PW_P0-6       = 0x20 0x20 0x00 0x00 0x00 0x00
-> EN_AA            = 0x00
-> EN_RXADDR        = 0x03
-> RF_CH            = 0x4c
-> RF_SETUP         = 0x07
-> CONFIG           = 0x0e
-> DYNPD/FEATURE    = 0x00 0x00
-> Data Rate        = 1MBPS
-> Model            = nRF24L01+
-> CRC Length       = 16 bits
-> PA Power         = PA_MAX
```

Figure 9: RF module parameters overview

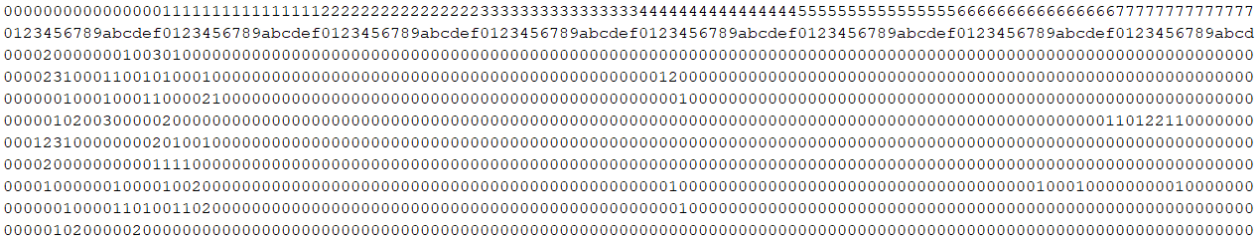


Figure 10: Scan of the $2.4GHz$ band

2 Implementation

2.1 Hardware set-up and software organization

2.1.1 Required hardware and wiring

The transmitter is built in such a way that we have two power supply lines : $5V$ for the antenna, $3.3V$ for the IMU and the GNSS receiver. The IMU is located in the center of the breadboard so that the "frame" of the breadboard when we move it matches the IMU's frame. It can be also good to move the wires as far as possible from the IMU because of its high sensitivity to external magnetic field, even those induced by the wiring. Most of the issues in the experiments presented later come from wiring and power supply, especially when we move the breadboard because wires tend to disconnect easily. That is why hardware requires a particular attention before each experiment.

2.1.2 Software organization

All the MATLAB programs that extract information from the Arduino serial port begin with the same code structure. After the program clears the workspace and closes the previous communication, a new communication is opened through the Arduino serial port at 115200bps baud rate. The program then waits to read a 'GO' on the serial port which means that the transmitter and receiver are synchronized and ready to communicate. Once this condition

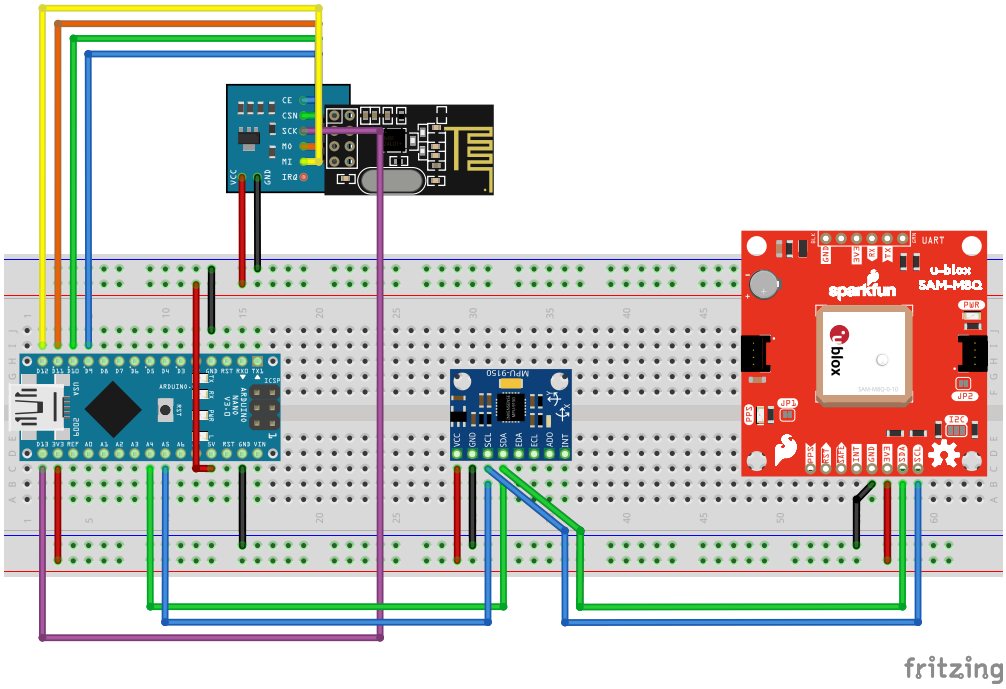


Figure 11: Transmitter block wiring

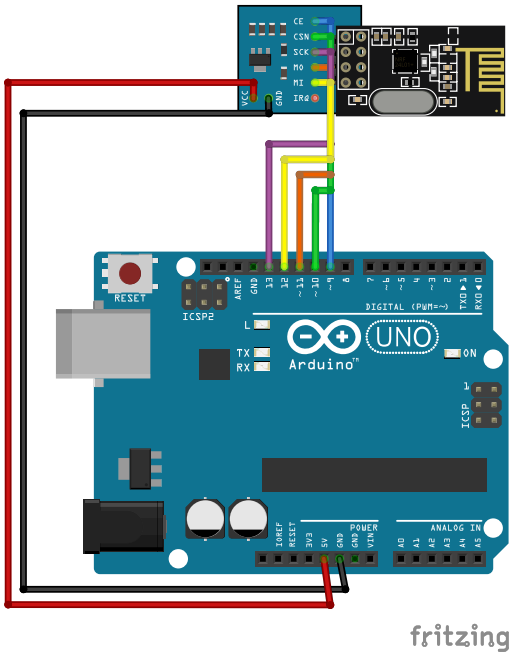


Figure 12: Receiver block wiring

is checked, we wait for data stabilization during 1s and then data packet are collected. The

standard data packet y is a string built in the following way:

$$y = a_x,\, a_y,\, a_z\ ;\ g_x,\, g_y,\, g_z\ ;\ m_x,\, m_y,\, m_z \tag{1}$$

It includes the acceleration (in g), the angular velocity (in $^\circ/s$)and the magnetic field (in mG) for the three axes of the IMU. When data is sent by the GNSS receiver, the structure changes to include this new information:

$$y = a_x,\, a_y,\, a_z\ ;\ g_x,\, g_y,\, g_z\ ;\ m_x,\, m_y,\, m_z\ ;\ lat,\, lng,\, 0\ ;\ \psi,\, v,\, 0 \tag{2}$$

The new values are the latitude and the longitude (in $^\circ$ with decimals), the heading of the receiver(in $^\circ$) and the its velocity (in m/s). We always have packets of three-values sub-packets, so that the method "str2num" in MATLAB can turns these strings in 3x3 or 5x3 matrices, which justify the use of "0" for the two last sub-packets. The use of a 115200bps baud rate can be justified considering the most demanding case in terms of transmission rate which is a 5x3 data packet and considering the frequency of the IMU which is $200Hz$, even if GPS data are actually sent at a $10Hz$ frequency. This 5x3 packet is made of 15 integers codded on 16 bits, which makes a 240 bits packet. At $200Hz$, it represents 48000 bits sent per second, so even in a hypothetical situation where GPS data are sent at this frequency, the 115200bps baud rate can still be used.

2.2 Data acquisition

Data acquisition is made in the loop function of the Arduino transmitter program as it can be seen in the following picture. GPS data acquisition is explicitly written whereas IMU data acquisition is done each time with or without GPS data in the function *sendRF*. However, each sensor needs a specific initialization detailed below.

2.2.1 IMU data

Acquiring the IMU data requires a prior manipulation of the chip's registers through the I^2C interface. This is done in the transmitter program by the functions *initMPU* for the accelerometer and gyroscope and *initMAG* for the magnetometer. Reading and writing functions are used first to read the previous settings, then to clear their contents except the reserved bits (according to the register map [8]) and finally to write the desired settings among those that can chosen as it is specified in 1.2.1. This procedure has to be done each time the transmitter program is uploaded because it has been observed that the settings disappear when the power supply is turned off. The settings can be easily modified using the section "Dashboard" in the introduction of the transmitter program without modifying the initialization functions themselves. The following table summarizes the chosen values for the range of the sensors, their sample rate before and after reduction (a sample rate divider is used to adjust more precisely the desired output frequency), the built-in low-pass filter cut-off frequency and the resolution of the magnetometer:

	Range	Sample rate (after reduction)	LPCF	Resolution
accelerometer	$4g$	$1kHz$ ($200Hz$)	$5Hz$	-
gyroscope	$500^\circ/s$	$1kHz$ ($200Hz$)	$5Hz$	-
magnetometer	-	$100Hz$	-	16 bits

```
void loop()
{
  if (millis() - lastTime2 > 5)
  {
    if (millis() - lastTime1 > 100)
    {
      lastTime1 = millis();  // Update the GPS timer
      lastTime2 = millis();  // Update the IMU timer

      // GPS output are long values so we need to cut them into two integers that
      // can be sent in our RF payload.
      coordIntFormat(lat_int, lat_dec, myGPS.getLatitude());  // given at 10^-7 °
      coordIntFormat(lng_int, lng_dec, myGPS.getLongitude());  // given at 10^-7 °
      vel_int = speedIntFormat(myGPS.getGroundSpeed());  // given in mm/s
      head_int = headIntFormat(myGPS.getHeading());  // given at 10^-5 °
      alt_int = altIntFormat(myGPS.getAltitude());  // given in m

      //Serial.println(vel_int);
      //Serial.println(alt_int);

      // send data with GPS coordinates
      sendRF(1);
      }
    else
    {
      // send data without GPS coordinates
      lastTime2 = millis();  // Update the IMU timer
      sendRF(0);
      }
    }
}
```

Figure 13: *loop()* function of the transmitter program

After the sensor initialization is done, data are read in the IMU's buffers and stored in 16 bits integers containers. The sensor has registers that indicates if new data are available or not, but we already know they are available each $5ms$, so it is more efficient in terms of number of operations to use timers rather than use a function that detects data availability.

2.2.2 GPS data

Similarly to the IMU, the GNSS receiver needs to be initialized to deliver proper information. The Ublox library that is used contains several functions that allow to configure the receiver. Thus, we choose a data output method (in our case the I^2C interface), we choose a $10Hz$ update frequency and we also configure the receiver so that it sends automatically data report containing all the information (time, position, velocity, altitude, satellites etc).

Then, data are collected every $10ms$ in the *loop()* function of the transmitter. Geographic coordinates are too long to be stored in 16 bits integer containers, that is why each coordinate is cut into two integers that will be reassembled later by the receiver program. All the information given by the parser is then treated by several functions such as *coordIntFormat* or *speedIntFormat* that will turn all the raw values into integers that are short enough to hold in 16 bits containers. The consequence is that the values sent by the transmitter block are not be in the good units, but the conversion is done later by the receiver program. For

instance, speed is given in mm/s by the parser, it is converted and sent in cm/s and finally the receiver program converts it into m/s.

2.3 Data transmission

Communication is performed with the RF antennas presented previously. These antennas use the Enhanced ShockBurst protocol detailed in A.4. The communication parameters are specified in the *setup()* function of both transmitter and receiver programs. Many operations can be done with the RF24 library, but the main operations done in these initialization blocks are the creation of a radio object, the activation of the automatic acknowledgement procedure and the configuration of the reading and writing pipes. For two nodes (a transmitter and a receiver), the pipes management only consists in opening a writing and reading pipe on each radio, with opposite addresses.

The first RF communication happens in the *setup()* function of both nodes. As it has been seen previously, the IMU data are not send with proper units. Data that are sent are 16-bits integers that contain raw data from the sensors. That is why this first communication sends the chosen resolution to the receiver so that the sensors resolutions are applied only in the receiver block after data have been received. When it is done, the receiver block displays a "GO" which is used as a marker for MATLAB because it means that usable data are coming.

After that, the transmitting antenna only sends something when new data arrive and need to be sent, while the receiving antenna is continuously in reading mode. However, the receiving block must know what packet length to expect, so it also must be specified in the receiver block (here we are working with packets of 16 integers). The function that sends data from the transmitter is *send()*. When it is only needed to send the IMU data, the function is called with the argument 0 which means that all the GNSS containers will be equal to 0. When GNSS data are available, it is called with the argument 1 which means the GNSS containers will contains the new GNSS data.

```
void sendRF(int val)
{
  readData(accelData,ACCEL_XOUT_H);
  readData(gyroData,GYRO_XOUT_H);
  readMag(magData);
  int16_t dataOut[16] = {accelData[0], accelData[1], accelData[2],
                         gyroData[0] , gyroData[1] , gyroData[2] ,
                         magData[0]  , magData[1]  , magData[2]  ,
                         lat_int*val , lat_dec*val , lng_int*val , lng_dec*val,
                         head_int*val, vel_int*val , alt_int*val};
  //Serial.println(magData[1]);
  radio.write(&dataOut, sizeof(dataOut));
}
```

Figure 14: *send()* function of the transmitter program

2.4 Calibration

2.4.1 Accelerometer and gyroscope calibration

The accelerometer and gyroscope have internal biases that we need to get rid of. These biases are the mean values that we obtain from the sensor when it is at rest and on a flat surface. There are two ways to proceed. First, we can calculate these biases in the Arduino code of the transmitter after the IMU initialization procedure and push them into hardware registers dedicated to the preservation of biases in order to subtract them automatically from the measured data. This can be done once but temperature variations affect those biases, therefore this operation is required before each use of the sensor. The second way to do it and the way that has been chosen here is to calculate the biases in the MATLAB program. One or two seconds are used between the moment data transfer begins and the moment incoming data is actually used in a fusion algorithm. It easier than manipulate the offset registers of the IMU and it allows a better control of the accelerometer calibration especially because of gravity. Gravity must not be considered as an offset to eliminate, and because the sensor is never in a perfect flat position, the acceleration along the z-axis is never really equal to $1g$. Applying a correction offset for the z-axis is an incorrect solution because we would obtain wrong values when the sensor moves. We can apply a correction factor to make the acceleration equal to $1g$ at rest and simulate a sensor in a perfect flat position when the object on which the sensor is fixed is itself in a perfect flat position. The offset coefficient are calculated as follows:

$$a_{off,j} = \frac{\sum\limits_{i \in aList} a_{ij}}{card(aList)}, \quad j \in \{x, y, z\} \tag{3}$$

$$g_{off,j} = \frac{\sum\limits_{i \in gList} g_{ij}}{card(gList)}, \quad j \in \{x, y, z\} \tag{4}$$

The calibration of the acceleration and the angular velocity is done in the following way:

$$\begin{bmatrix} a_{x,calib} \\ a_{y,calib} \\ a_{z,calib} \end{bmatrix} = \left(\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} - \begin{bmatrix} a_{off,x} \\ a_{off,y} \\ 0 \end{bmatrix} \right) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{a_{off,z}} \end{bmatrix} \tag{5}$$

$$g_{j,calib} = g_j - g_{off,j}, \quad j \in \{x, y, z\} \tag{6}$$

2.4.2 Magnetometer calibration

Types of errors. A magnetometer, especially a low cost one, needs absolutely to be calibrated properly because the Earth magnetic field is our second absolute reference with the gravity. Indeed, even a slight defect in its calibration can have visible consequences, which means that this step requires focus and patience in order to obtain satisfying results. It should be remembered that a typical value for an axis of the magnetometer that is aligned with the Earth magnetic field is $500mG$. This ideal output when the sensor is rotated is all the direction is a $500mG$ radius sphere centered in $(0, 0, 0)$. Two possible errors need to be corrected [9]:

- hard-iron effect: caused by permanently magnetized ferromagnetic components of the sensor, the consequence is an offset error of the sphere,.
- soft-iron effect: caused by an interfering magnetic field induced by the geomagnetic field onto normally unmagnetized ferromagnetic components of the sensor, the consequence is a scale error (ellipsoid deformation of the sphere) that gives different weights for three components of the magnetometer.

The calibration is written in MATLAB to save space on the Nano but above all to have more freedom in the manipulation of data and the creation of a visual rendering.

Correction calculation. Some sophisticated methods exist for the magnetometer calibration [9], but here simple calculations are performed and still offer satisfying results [10]. Offset errors are calculated as follows:

$$off_j = \frac{max(\mathbf{m_j}) + min(\mathbf{m_j})}{2}, \quad j \in \{x, y, z\} \tag{7}$$

Scale errors are calculated as follows:

$$scale_j = \frac{\sum\limits_{i \in \{x,y,z\}} (max(\mathbf{m_i}) - min(\mathbf{m_i}))}{3\,(max(\mathbf{m_j}) - min(\mathbf{m_j}))}, \quad j \in \{x, y, z\} \tag{8}$$

Finally, the calibrated values are written:

$$m_{j,calib} = \left(m_j - off_j\right) scale_j, \quad j \in \{x, y, z\} \tag{9}$$

A closer visual rendering of the calibration is presented in figure 15 where are presented raw data (red), filtered data (blue) and calibrated data (green).

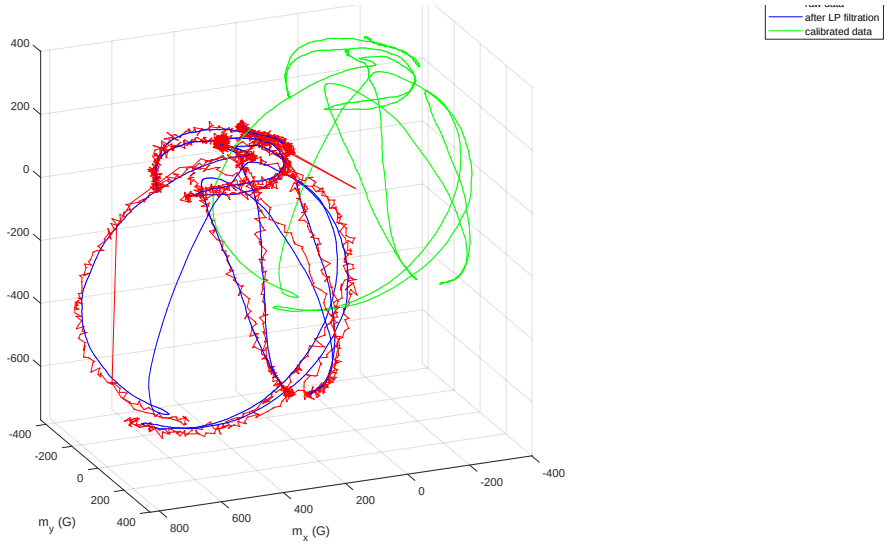


Figure 15: Complete 3-axes calibration

Code structure. After the program initialization, in the beginning of the acquisition part, the user is asked to rotate the sensor in all the directions such as the magnetometer output draws a sphere in the 3D animated figure. Each data that comes from the serial port has its format checked. Indeed, some hardware defects (such as a slightly unplugged cable, an altered pin connection while moving the IMU, etc.) can alter the format of the transmitted data, and the corrupted data shall be ignored in order to avoid a format error in the calculation part of the program. Proper data is saved before being treated. The treatment consists in a low-pass filtering step and finally the calculation of the offset and scale errors. There is a possibility to test directly the calibrated values with a 3D live plotting where we can see if the calibrated values indeed follow the surface of a sphere. Two videos showing the procedure have been recorded: the *expected MATLAB output* and the *recommended motions to calibrate the magnetometer*.

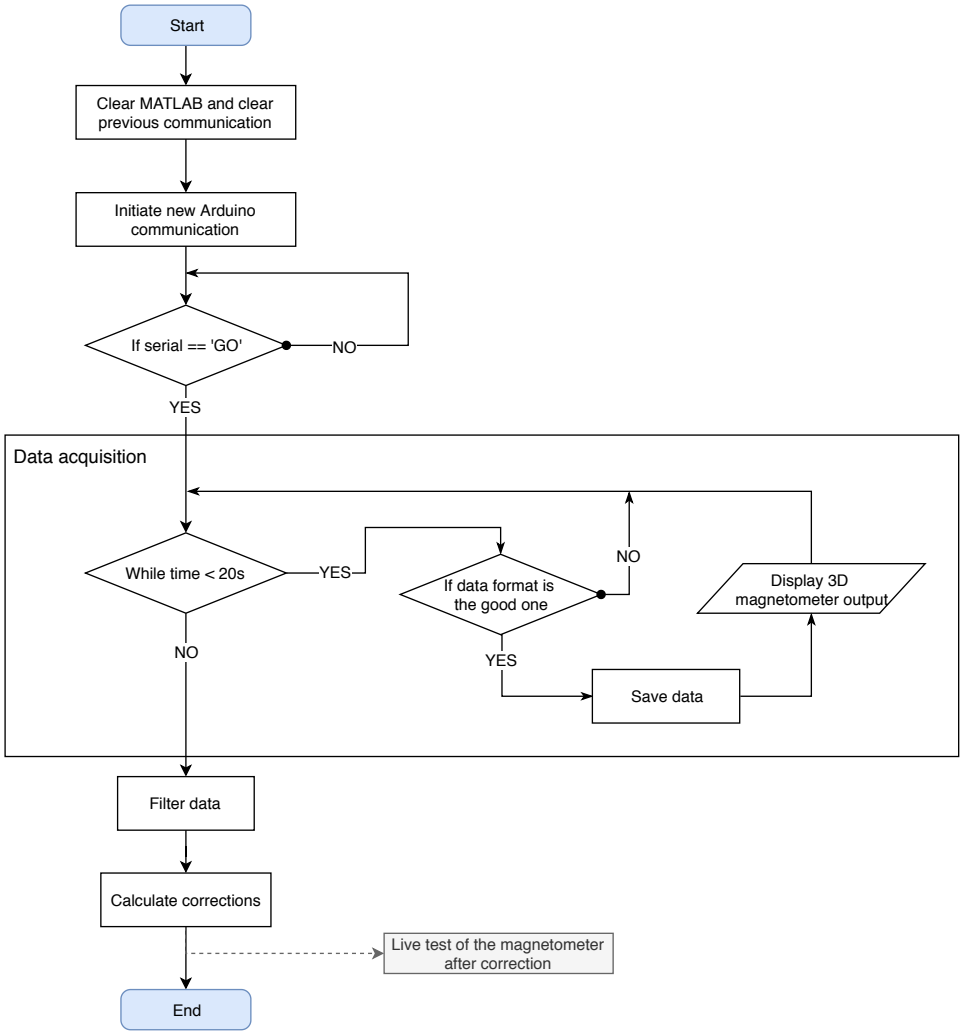


Figure 16: Calibration program architecture

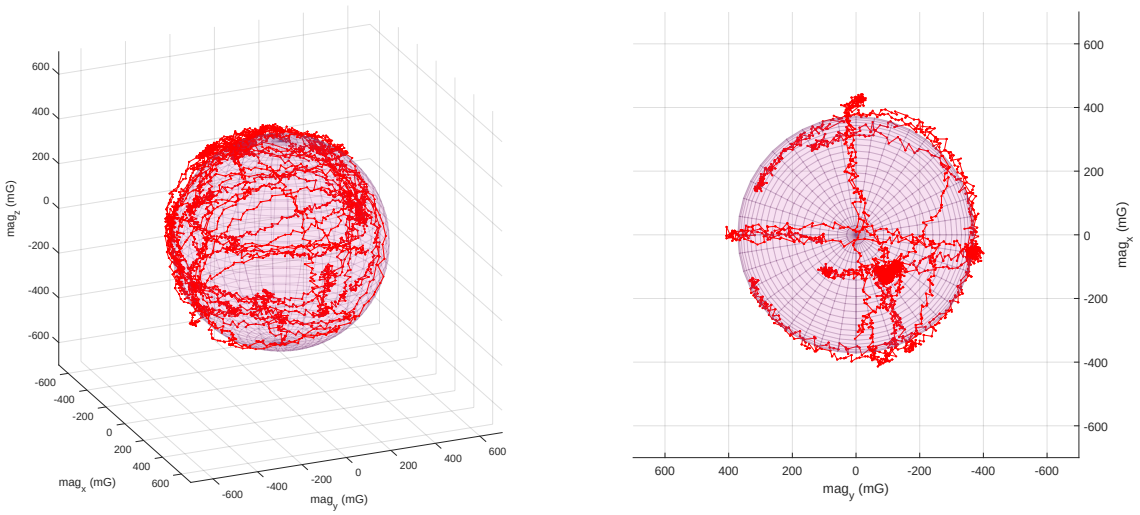


Figure 17: Test of the calibrated magnetometer

3 Sensor fusion

There are two main ways to treat a tracking issue: either buy an expensive and high-performance sensor of buy several low-range, medium-performance sensors and fuse their output data through a fusion algorithm. This method corrects the deficiencies of the individual sensors to calculate accurate position and orientation [11]. Because many research have been conducted on this subject, fusion algorithms are nowadays numerous and of different natures. For general purposes, it is preferable to use low-cost hardware and these fusion methods.

The major problem with low-range IMU is noise. Because orientation and position come from the 1st and 2nd integration of output data, the initial noise and bias will cause drifting and pulsating phenomenons that make the final result unusable. The gyroscope is reliable on the short-term but drift makes its output data unreliable on the long-term. On the contrary, accelerometer and magnetomer are not reliable on the long-term but not on the short-term due to noise. Thus, accelerometer and magnetometer can be used to determine the orientation of the IMU with respect to the Earth magnetic field to avoid drift. Then, filtering strategies must be used to smooth the output curves [1].

Many fusion algorithms and filters have been published, but will detailed only three of them, in order to study different approaches in sensor fusion issues and compare their performance. The first one is a complementary filter based on a trigonometric approach, quite popular and highly documented [12][13][14]. The second algorithm is commonly called the Madgwick algorithm as it has been written by Sebastian Madgwick from the University of Bristol [15]. It is a quaternion-based fusion algorithm that uses data from a 9-DOF IMU sensor. The final algorithm is a Kalman-based algorithm.

3.1 Definition of orientation parameters and sensor data

Before we start to develop any algorithm, it is necessary to define in which frame orientation estimation is done as well as the orientation parameters.

3.1.1 Coordinate frames

Any tracking study requires a description of the coordinate frames. In this study, the following frames are considered [16]:

- Body frame **b**: coordinate frame of the IMU and of the body it is attached to. The measurements that are performed with the IMU are expressed in this frame.

- Navigation frame **n**: local frame that lies on the Earth surface in which we navigate, that is to say we want to know the position and orientation of the **b**-frame in this local geographic frame.

- Earth-centered inertial frame **i**: stationary frame whose origin is located in the center of the Earth.

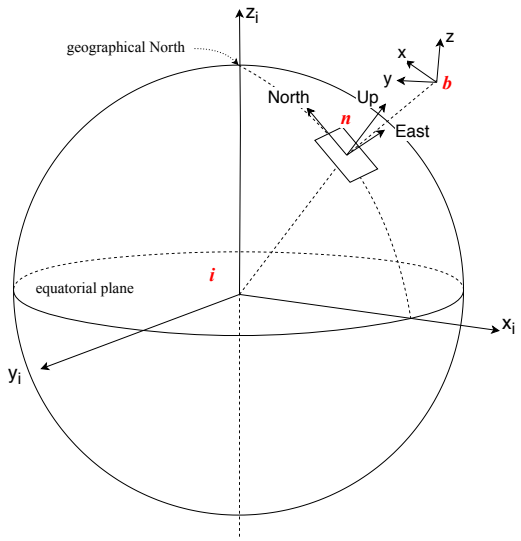


Figure 18: IMU frame with respect to the external frames

3.1.2 Orientation parametrization

Euler angles. Any rotations can be described by three consecutive rotations. The convention we use to give the Euler angles is first ψ around the z-axis, θ around the y-axis and ϕ around the x-axis. These rotations ψ,θ,ϕ are called yaw, pitch and roll. A frame **v** rotated by (ψ,θ,ϕ) in a reference frame **u** has its rotation described by the rotation matrix from the frame **u** to the frame **v** $R^{vu}_{zyx}(\phi, \theta, \psi) = R_z(\psi)R_y(\theta)R_x(\phi)$.

The use of Euler angles raises the question of the chosen sequence of rotations and the question of "Gimbal lock" (see B.1). Quaternions are a more complex mathematical solution to these issues.

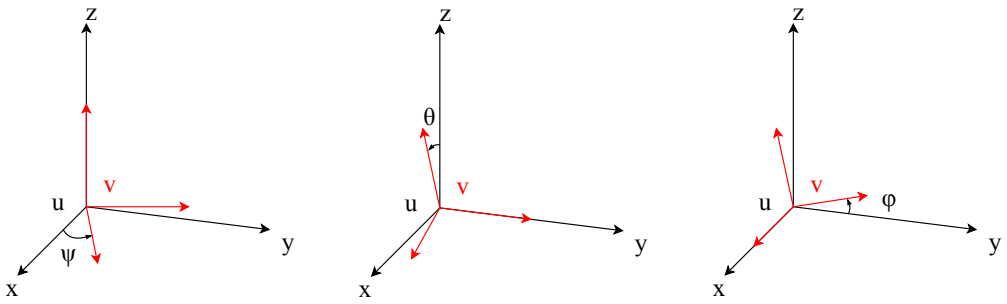


Figure 19: Euler angles

Quaternions. Quaternions are a mathematical tool developed by William Rowan Hamilton, in the 19th-century. They are an extension of the complex numbers. A quaternion has a scalar part (real part) and a vector part (imaginary part) such as $\mathbf{q} = q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$. Definitions and properties are detailed in B.2.

Any 3D rotation in the rotation space can be represented with a unit quaternion as a coordinate point (q_w, q_x, q_y, q_z) that corresponds to a rotation around the axis $\mathbf{v}$ by the angle $\alpha = 2\arccos w = 2\arcsin\sqrt{q_x^2 + q_y^2 + q_z^2}$. Here, a single rotation around vector $\mathbf{v}$ is equivalent to three consecutive rotation around the three axes of the frame as presented with the Euler angles. This is one of the solution to avoid gimbal lock.

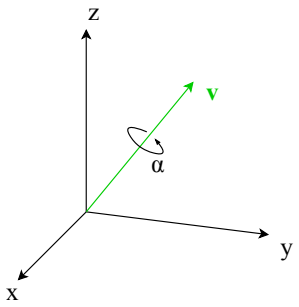


Figure 20: Quaternions applied to 3D rotation

3.1.3 Sensor output data

In this project, the following units for the sensor data are the following ones:

- Angular velocity ω in $^\circ/s$
- Acceleration a in g ($1g = 9.81m.s^{-2}$)
- Magnetic field m in mG ($m_{earth} \simeq 500mG$)

Sensor output is far from being perfect especially with low-range component. Each measurement can be usually divided in a true value, a bias and a noise that form together an approximation of the true value [12]. As it has been already underlined, these measurement

errors tend to accumulate and cause drift when data is integrated.
The gyroscope data can be written in the following way:

$$\tilde{\boldsymbol{\omega}}_k = \boldsymbol{\omega}_k + \mathbf{b}_\omega + \boldsymbol{\epsilon}_k, \quad \boldsymbol{\epsilon}_k \sim \mathcal{N}(0, \boldsymbol{\Sigma}_\omega^2) \tag{10}$$

With $\tilde{\boldsymbol{\omega}}_k$ the estimated k-th angular velocity measurement, $\boldsymbol{\omega}_k$ the true angular velocity, $\mathbf{b}_\omega$ the bias, $\boldsymbol{\epsilon}_k$ a Gaussian noise that obeys to a standard deviation matrix $\boldsymbol{\Sigma}_\omega^2 = diag(\sigma_x^2, \sigma_y^2, \sigma_z^2)$. Similarly, we have for the acceleration:

$$\tilde{\boldsymbol{a}}_k = \boldsymbol{a}_k + \mathbf{b}_a + \boldsymbol{\epsilon}_k, \quad \boldsymbol{\epsilon}_k \sim \mathcal{N}(0, \boldsymbol{\Sigma}_a^2) \tag{11}$$

Finally, the magnetometer output data can be described as follows, where the noise is neglected insofar as the magnetic field measurement is more stable:

$$\tilde{\boldsymbol{m}}_k = \boldsymbol{m}_k + \mathbf{b}_m \tag{12}$$

These measurement models show which operations are needed to obtain proper values that can be treated in a fusion algorithm, that is to say calibration and filtering operations.

3.1.4 Fusion algorithms feeding

We saw previously the structure of the data packets received by MATLAB (2.1.2). However, it is not possible to feed the following fusion algorithm directly with the packet because the accelerometer-gyroscope and the magnetometer do not share the same coordinate system so their axes and the directions of their axes are not the same. In order to feed the algorithms

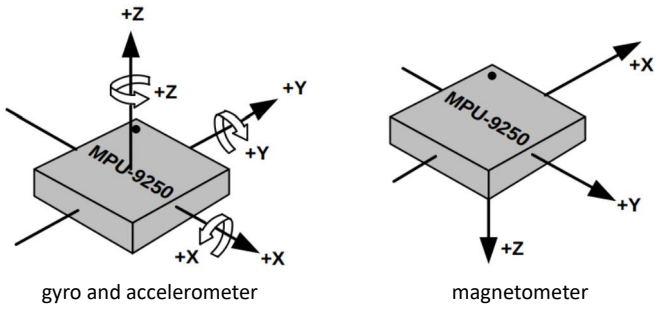


Figure 21: Differences in the coordinate systems for the MPU9250 [17]

with consistent data, the packet received from the IMU is automatically changed to:

$$y = a_x,\ a_y,\ a_z\ ;\ g_x,\ g_y,\ g_z\ ;\ m_y,\ m_x,\ -m_z \tag{13}$$

3.2 Complementary filter

3.2.1 Implementation

Principle. As a first step, a simple fusion algorithm will be used to calculate the pitch, roll and yaw. This algorithm, called in the literature complementary filter, fuses the gyroscope and accelerometer data to calculate the pitch and roll. However, the calculation of

yaw requires also the magnetometer data. Indeed, the z-axis is aligned with the gravity, so a rotation around it will not cause any change in the accelerometer output data. Because the Earth magnetic field is perpendicular to the gravitational field, a rotation around the z-axis will be visible in the magnetometer output data. This gives the yaw for a rotation in the x-y plane, but when the IMU is tilted (pitched or rolled), the magnetometer data needs to be compensated, that is why the pitch and roll previously calculated are also taken into account for the yaw calculation.

Rotation sequence. The chosen sequence of rotations shall not be neglected in the calculation of pitch and roll. Indeed, pitch and roll calculations require the projection of the gravity vector respectively on the x-axis and z-axis (pitch is a rotation around the y-axis) and on the y-axis and z-axis (roll is a rotation around the x-axis). Assuming the sensor only undergoes the gravity, the acceleration in the sensor coordinates $a^{(v)}$ is given by:

$$\frac{\mathbf{a}^{(v)}}{\|\mathbf{a}^{(v)}\|} = \frac{1}{\sqrt{a_x^2 + a_y^2 + a_z^2}} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = R(\psi, \theta, \phi) \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{14}$$

No matter the chosen sequence, we obtain a set of three equations. We also have to add the fact the sum of the acceleration components is always 1 (the acceleration vector moves on the surface of a sphere with a 1g radius). Thus, despite three values, the accelerometer output has only two degrees of freedom [13]. A system with three variables on the right is therefore unsolvable, so the sequence of rotations shall be chosen regarding the possibility to solve the system. From [13], it appears that the sequence $\{xyz\}$ gives a solvable system, while for example the sequence $\{xzy\}$ is unsolvable:

$$\{xyz\} \qquad \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = R_{xyz}(\psi, \theta, \phi) \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -\sin\theta \\ \cos\theta \sin\phi \\ \cos\theta \cos\phi \end{bmatrix} \tag{15}$$

$$\{xzy\} \qquad \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = R_{xyz}(\psi, \theta, \phi) \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -\cos\psi \sin\theta \\ \cos\theta \sin\phi + \cos\phi \sin\psi \sin\theta \\ \cos\theta \cos\phi - \sin\phi \sin\psi \sin\theta \end{bmatrix} \tag{16}$$

Euler angles calculation. From the $\{xyz\}$ sequence and the equation 15 we obtain from the accelerometer the roll angle ϕ_{acc} and the pitch angle θ_{acc}:

$$\tan\phi_{acc} = \frac{a_y}{a_z} \implies \phi_{acc} = \arctan\left(\frac{a_y}{a_z}\right) \tag{17}$$

$$\tan\theta_{acc} = \frac{-a_x}{\sqrt{a_y^2 + a_z^2}} \implies \theta_{acc} = \arctan\left(\frac{-a_x}{\sqrt{a_y^2 + a_z^2}}\right) \tag{18}$$

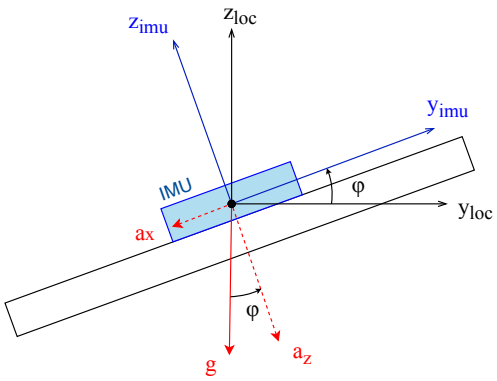


Figure 22: Trigonometric representation of the first rotation (pitch)

The calculation of the first rotation (roll) is a simple rotation around the x-axis starting from a flat position. It can be easily seen as a simple trigonometric calculation (figure 22).

A problem of instability for $\tan\phi$ can occur when $a_y = a_z = 0$. A solution commonly used for the $\{xyz\}$ sequence that gives an approximation of $\tan\phi$ is to mix the original denominator with a fraction of a_x so that the denominator and numerator can be equal to zero without any instability [13]. However, as we are working on a ground vehicle that is more stable than a drone for example, this potential problem will be neglected.

As previously indicated, yaw is calculated by fusing magnetometer measurement with pitch and roll. After calculations not detailed here (cf. [14]), we obtain the following formula:

$$\psi = \arctan\left(\frac{m_z \sin\phi - m_y \cos\phi}{m_x \cos\theta + m_y \sin\theta \sin\phi + m_z \sin\theta \cos\phi}\right) \tag{19}$$

Complementary filtering. Because it is more relevant to exploit gyroscope data on the short-term and accelerometer data on the long-term (cf. 3), we can supplement the accelerometer data for the pitch and roll by the gyroscope data with a ratio $\alpha \in [0, 1]$ [18] such as :

$$\phi = \alpha(\phi + \dot{\phi}dt) + (1 - \alpha)\phi_{acc} \tag{20}$$

$$\theta = \alpha(\theta + \dot{\theta}dt) + (1 - \alpha)\theta_{acc} \tag{21}$$

3.2.2 Results

Orientation. The result we obtain seems satisfying for pitch and roll as the output is quite smooth, but yaw is unstable even at rest. We observe an influence of pitch and roll on yaw even for small angles. The influence on yaw and the instability of yaw make this filter unusable for tracking, which is the reason of the use of the Madgwick algorithm. Two videos that show the output with animated lines and an animated car have been recorded. They are available by clicking on the following links: *animated lines* and *animated car*.

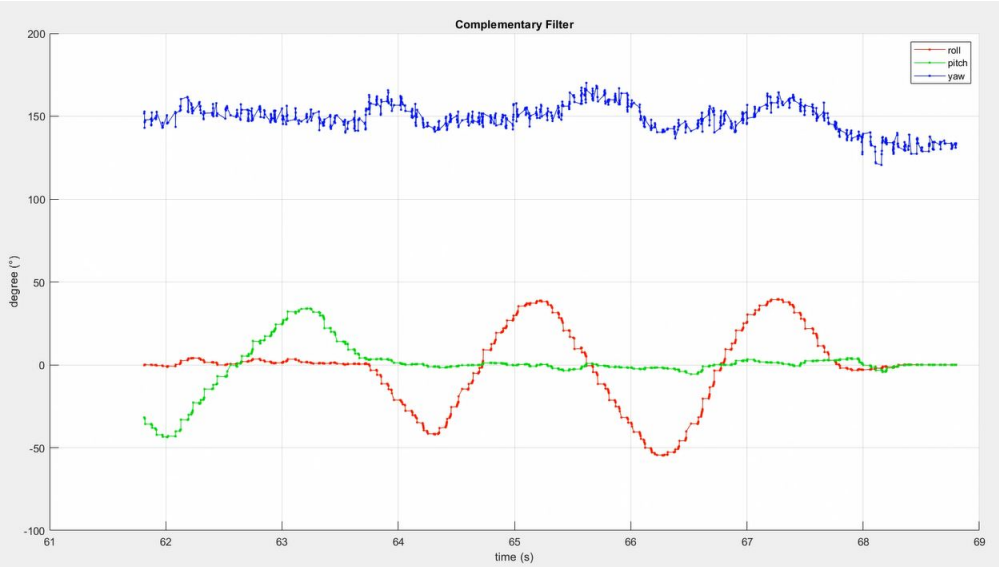


Figure 23: Animated lines visualization for the complementary filter

3.3 Madgwick algorithm

3.3.1 Implementation

This algorithm uses a 9-DOF IMU in a gradient-descent algorithm that gyroscope errors. It has the advantage to be not so expensive in terms of computation and is effective even with low samples rates, which makes this algorithm suitable for inexpensive hardware. The algorithm includes a magnetic distortion compensation block and a gyroscope bias drift compensation block as it can be seen in figure 24. The influence of the previous state through the gradient descent is controlled by the coefficient β and the influence of the gyroscope drift compensation block is controlled by the coefficient ζ. The detailed calculations can be found in the study published by Sebastian Madgwick [15]. This algorithm has been adapted in MATLAB code from the implementation in C published in Madgwick's article [15] and the implementation for Arduino published by Kris Winer from the Lawrence Livermore National Laboratory [19]. The resulting function is *fusionMadgwick()*. It takes in argument the previous quaternion, the acceleration triplet, the gyroscope triplet, the magnetometer triplet, the coefficients β and ζ and the sampling period calculated in real-time. Its output is a quaternion from which are extracted the Euler angles thanks to the conversion function *quatern2euler()*.

3.3.2 Results

We observe first a stabilization period of the filter for around $8s$. Then, we obtain a result that is much more precise and less noisy than the complementary filter. The curves are smoother, there is no difference in term of standard deviation between the yaw and the pitch or roll. Yaw is also more reliable for high pitch or roll than with the complementary filter. Yaw is in general less influenced by the changes of pitch or roll. It must also be noticed that the whole data chain from the sensor to the output of the algorithm offers a very

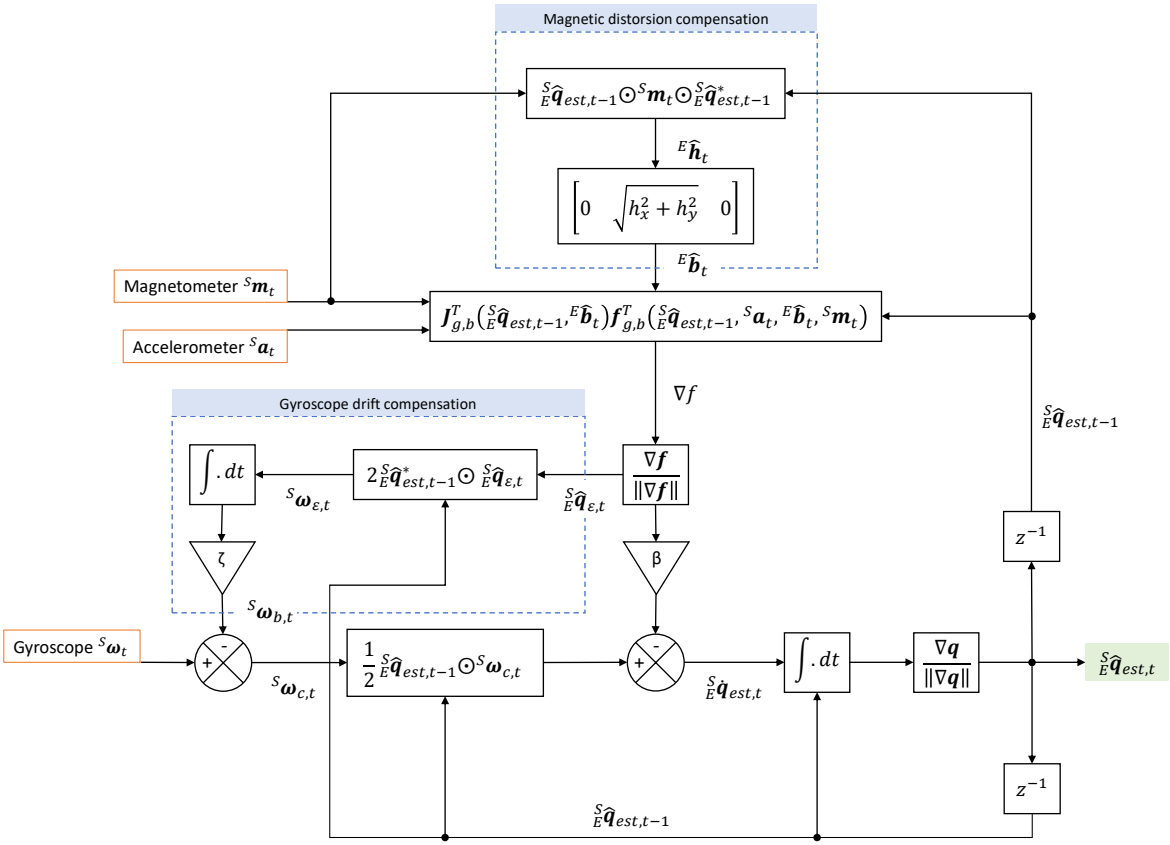


Figure 24: Block diagram of the Madgwick algorithm

satisfying responsiveness and it is hard to see a delay between the motion of the sensor and the computer display. In the same way as the complementary filter, two videos are available

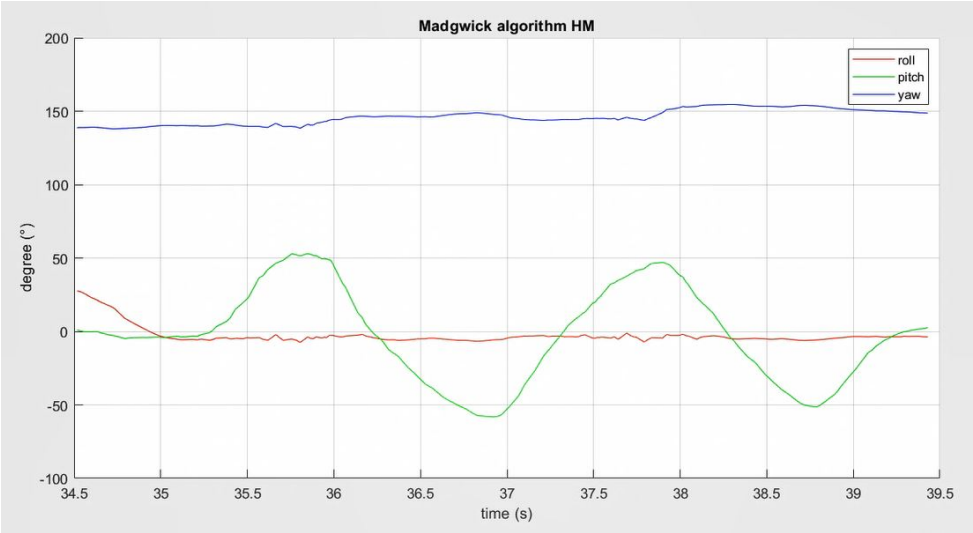


Figure 25: Animated lines visualization for the Madgwick algorithm

with the *animated lines* and the *animated car*.

3.4 Madgwick and GNSS receiver

3.4.1 Implementation

A first solution for vehicle tracking can be to compute the orientation with the Madgwick algorithm and extract the geographic coordinates from the GNSS receiver so that we can plot the position of the vehicle and its heading. This tracking approach is the simplest one insofar as there is no data fusion between the GNSS receiver and the IMU.

This solution has been tested a first time in the office but despite a good calibration, heading was wrong (indicated North was different from the one indicated by a cell phone) due the parasitic magnetic field in the office probably caused by the power supply of all the hardware. A second test has been performed in an external corridor that links two buildings of the faculty. Without parasitic magnetic fields, the orientation indicates the same North as

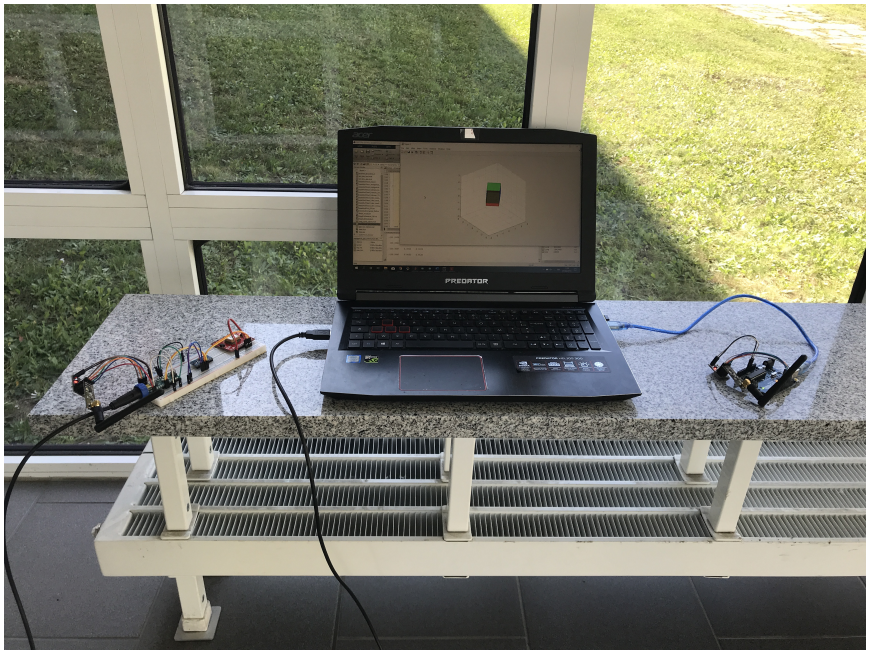
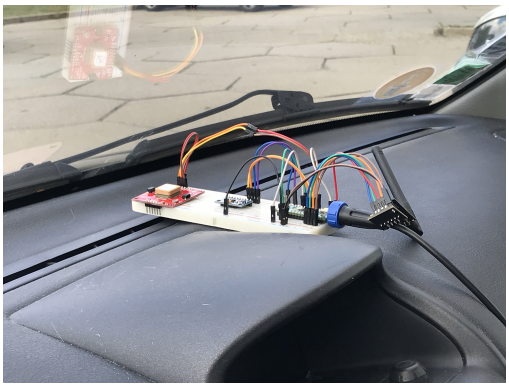


Figure 26: Madgwick + GNSS test in better conditions

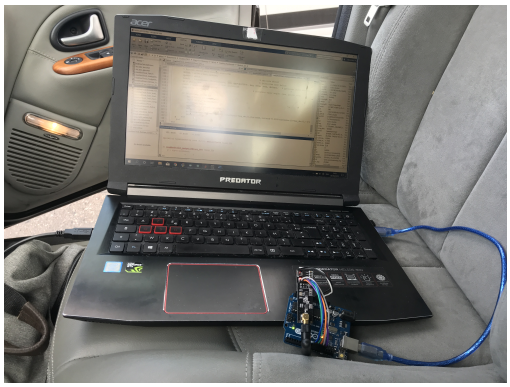
the compass of a cell phone. It must be noticed that the magnetic declination of the location (Chemnitz) has been deducted from the displayed heading values. This value can be find for example on the NOAA website [20]. The GNSS data are also more reliable and stable in the second test location than in the office due to a larger number of available satellites. The videos of the two tests can be watched by clicking on the following links: the *test in bad conditions* and the *test in better conditions*.

3.4.2 Results

These first tests proved that this system can work. The next tests have been performed on a passenger car and have been conducted on public roads around the university. The first



(a) Transmitter in front of the dashboard



(b) Receiver and computer

Figure 27: Passenger car experimentation set-up

display window offers a real-time view of the geographic coordinates in red with the vehicle heading in blue. When we decide to stop the program, a second window opens where is plotted the whole trajectory on a road map, so that we can see if the GNSS data match with the roads trajectory.

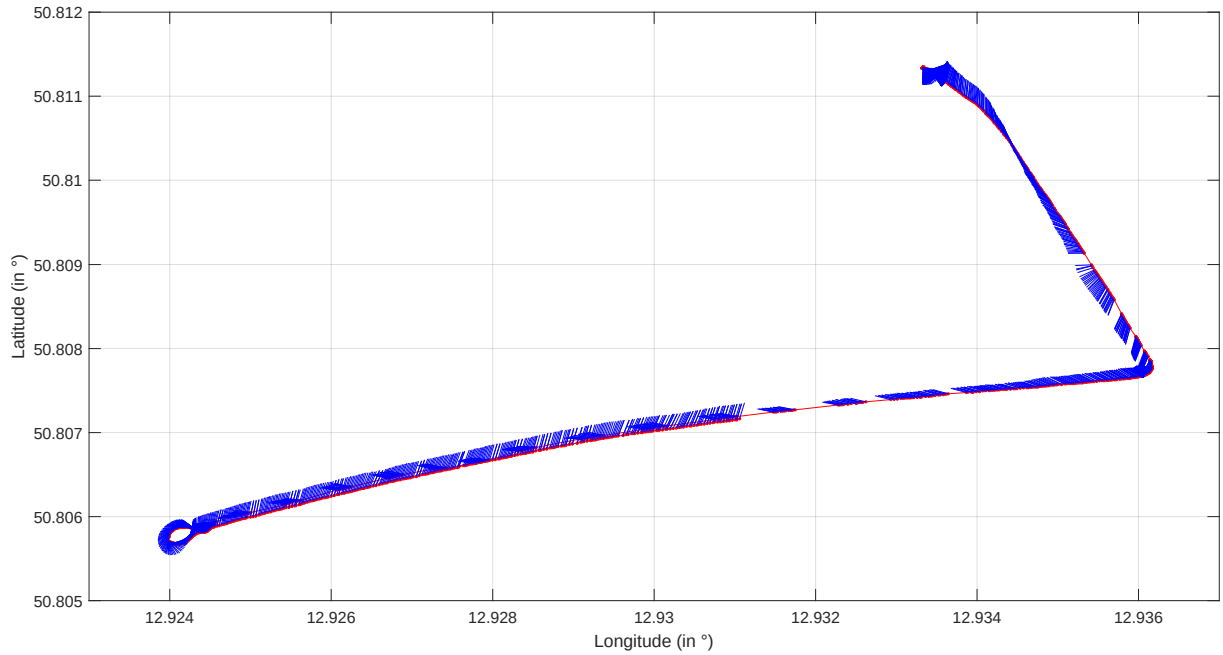


Figure 28: Passenger car experiment results

However, it appears that this solution has two limits:

- In the best conditions, the precision of the GNSS receiver is only $2.5m$ which could be enough for a passenger car but not for an RC car that moves in a radius of a few tens of meters.
- The magnetometer of this sensor is very sensitive and it is hard to isolate it from parasitic magnetic fields.

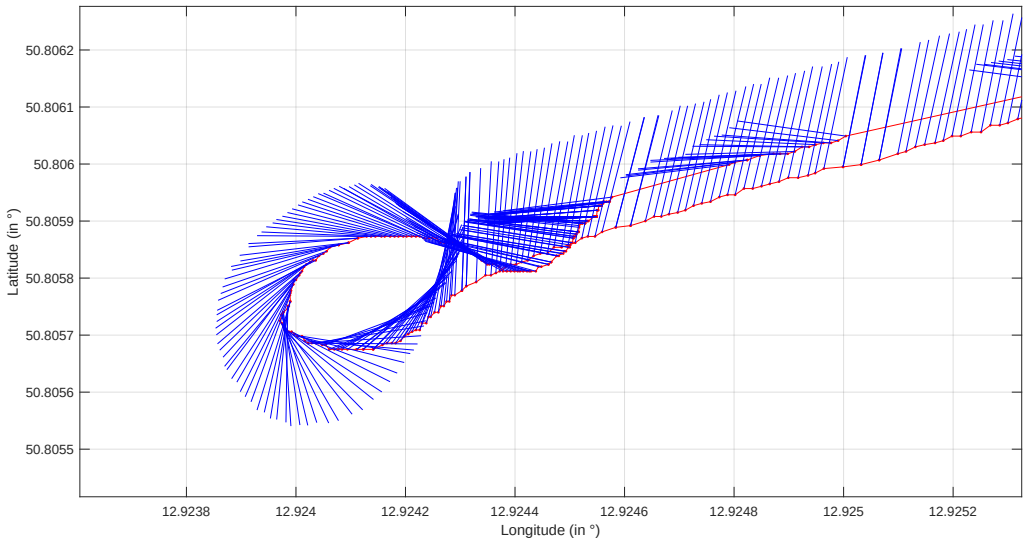


Figure 29: Zoom on a roundabout

Figure 30: Trajectory display on OpenTopoMap

Indeed, it has been clearly noticed during this on-road test that the GNSS receiver gives a sharp-edged trajectory due to its limited precision and the orientation has a strange behavior due the parasitic fields inside the car. A solution could have been to place the transmitter on the car roof but in that case it would have cut the RF communication. From the point of view on an RC car, the main problem would come from the motor and the magnetic field it generates.

It would be possible to isolate the magnetometer either by surrounding the motor with an insulating material or by placing the sensors on a platform isolated and located above the chassis. The second option would offer a better isolation but it would also lift the center of gravity.

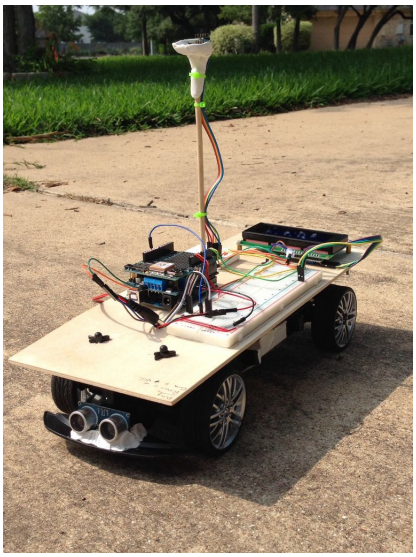


Figure 31: Example of sensor set-up for an RC car [21]

3.5 Kalman filter

The Kalman filter is a recursive filter which means the current state is updated from the previous state and from measurements that have been made in the time period. When it is used for tracking, the motion of the object is described by the state of the object. New measurements are included to keep the state estimate as accurate as possible. The filter is optimal because it minimizes the mean-square error of the state [22]. A loop in this filter

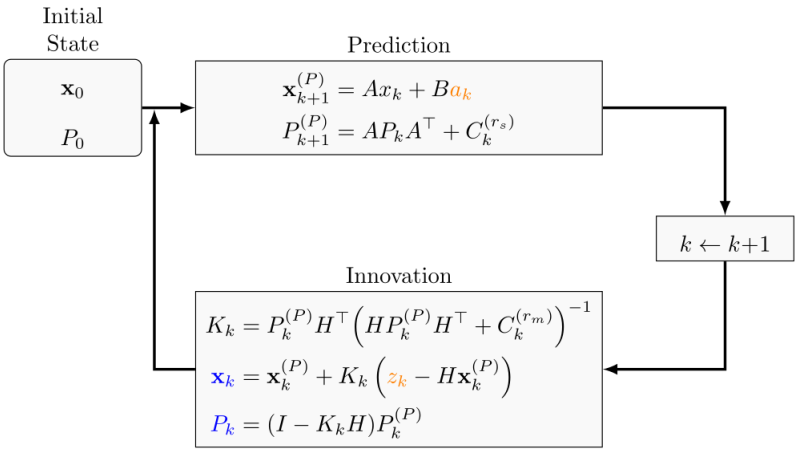


Figure 32: Block diagram of a Kalman filter [23]

is made of two main steps which are a prediction from the initial state and a correction ("innovation") of this prediction where is used the measurements z_k. From this last step we can extract the new state vector x_k.

On the basis of a previous study [24], it was attempted to implement such a filter, which required to write the algorithm that performs a calculation loop *kalman2D()*, a display program where is used the algorithm and finally two functions *coord2meter()* and *meter2coord()*

that convert the displacement in geographic coordinate into a displacement in meter and vice versa. These two functions are needed because calculations cannot be performed with coordinates, but the coordinates are necessary later to display the trajectory. However, little time was allocated to this technical solution and it resulted in failure without time to further explore the reasons for this failure. The coordinate conversion functions work correctly, which means that the problem comes from the filtering algorithm itself.

Conclusion

Despite a long time spent searching information and gaining knowledge, this 10-weeks internship resulted in many achievements. The first production is a hardware prototype of a vehicle tracker entirely built with Arduino components. Then, many test codes were written, which was very useful, not only to check if the components were working, but also to understand how they work and to understand specific protocols. The next production concerns the organization of the data chain, from their collection by sensors to their reception by a processing algorithm. This was the first challenging step as we needed a smooth, constant and fast data flow with a large amount of data, while everything had to be coded on microcontrollers with very limited memory. This pushed me to always search for any potential code optimization, which was very educative. Finally, the last technical challenge was the sensor fusion, something I never studied before and which required a lot of preliminary researches about the theory and a lot of preliminary work concerning data pre-processing such as calibration procedures. Although I obtained a first tracking solution with the Madgwick algorithm coupled with the GNSS receiver, this task is incomplete as a working Kalman filter would have been a better option with the available components for an RC car. A parallel task was to write a protocol on how to implement my work for future projects after mine. The goal was also to help people avoid the problems I encountered, especially regarding low cost hardware, means for data transmission, how to guarantee a reliable data flow, how to treat them properly and how to detect mistakes. All the codes can be found by following this *link*.

Many technical improvements can be done to this project in the future. The following list is only indicative and shows what I would have liked to do if I could continue the project.

- Use a PCB rather than a breadboard : A first and necessary improvement would be to design a PCB, for example on the software Fritzing, and to make it fabricated. Many companies offer this service by just dragging and dropping your design file to their website. This improvement is crucial if we want to have a fully usable prototype that does not have constant problems of connection and power supply. To make it more compact, this PCB could be designed for example like a stack of three boards, one for the microcontroller, one for the sensors, one for the radio transmitter.

- Increase the position accuracy : the accuracy of our tracking system must be improved. This can be done by working on a Kalman filter with additional sensors (odometer for velocity, barometer for altitude), which is maybe the most time-consuming option but which seems to give excellent results when it is performed correctly. Another option would be to buy a RTK GNSS receiver which can provide a centimeter-level accuracy in the best conditions, but this option is very expensive as an RTK GPS module costs around 200$ for Arduino (see this *link*). The third option is to implement a differential GPS (DGPS) sytem. This method uses "fixed ground-based reference stations to broadcast the difference between the positions indicated by the GPS satellite system and known fixed positions" [25] (see an implementation *here*).

- Find a way to isolate the magnetometer : as it was described previously, the magne-

tometer needs to be isolated to be usable.

- Implement a bi-directional RF communication : a bi-directional communication would be useful to configure more easily the sensors from MATLAB without touching the Arduino codes anymore.

- Design a human-machine interface : it would be interesting to design such an interface to gather the calibration process, the fusion algorithms and the possibility to choose all the sensors and Arduino parameters from a single window where would be then displayed the real-time results.

- Design a test bench for the IMU : it is not a priority but it could be interesting to build a a three-axes platform to check the accuracy of the orientation obtained thanks to fusion algorithms.

- Think about autonomous driving and torque vectoring : these projects come later in a second phase of the initial project. Autonomous driving can be explored with the help of other sensors such as lidar, radar, ultrasonic sensors or even camera in a perspective of image processing and deep learning. A torque vectoring project could start from the orientation obtained with the IMU (especially pitch and roll). If we know how the center of gravity of the car moves, it is possible to calculate the load transfers applied to each wheel. A first step could be to study *Motor Vehicle Dynamics*, Giancarlo Genta, 1997.

This internship was an opportunity for me to fully invest myself in sensors technology and electronics for 10 weeks. The knowledge I have acquired will certainly help me in the field of automotive engineering insofar as sensors and electronics are now a key element which is present everywhere in a car, especially for those with AI systems. The contact with students and researchers who work in the alternative propulsion department allowed me to learn about test bench technologies (acquisition cards, PID controller), but also fuel cells and their use in a vehicle's power chain. This supports my desire to work in a general scientific environment where mechanics, embedded systems, sensors, are all present at the same time. My interest also turned more to alternative propulsion systems and their integration in the vehicle.

Glossary

ASCII American Standard Code for Information Interchange. 10

CRC Cyclic Redundancy Check. 42

DMP Digital Motion Processor. 7

DOF Degree Of Freedom. 7, 26

GNSS Global Navigation Satellite System. 2, 3, 7, 10–12, 14–16, 28–30, 33, 36

GPS Global Positioning System. 9, 10, 14, 33

I^2C Inter-Integrated Circuit. 4, 7, 9, 10, 14, 15, 36, 40–43

IMU Inertial Measurement Unit. 7–9, 11, 12, 14–16, 19–21, 23, 26, 28, 34, 36, 44

ISM (Industrial, Scientific and Medical. 11

MEMS MicroelEctronMechanical Systems. 7, 8, 36

NMEA National Marine Electronics Association. 10, 36

NOAA National Oceanic and Atmospheric Administration. 28

PCB Printed Circuit Board. 33

PCF Packet Control Field. 42

RTCM (Radio Technical Commission for Maritime Services. 10

RTK Real-Time Kinematic. 33

SCL Serial CLock. 42

SDA Serial DAta. 42

SMA SubMiniature version A. 11

SPI Serial Peripheral Interface. 4, 7, 11, 36, 40, 41

SRAM Static Random Access Memory. 7

UART Universal Asynchronous Receiver Transmitter. 4, 7, 10, 40

List of Figures

References

[1] Charlotte Treffers and Luc van Wietmarschen. Position and orientation determination of a probe with use of the imu mpu9250 and a atmega328 microcontroller. 2016.

[2] Adxl335 accelerometer module. https://www.electronicwings.com/sensors-modules/adxl335-accelerometer-module, unpublished manuscript.

[3] Jeff Watson. Mems gyroscope provides precision inertial sensing in harsh, high temperature environments — analog devices. https://www.analog.com/en/technical-articles/mems-gyroscope-provides-precision-inertial-sensing.html, 2016.

[4] Hall effect magnetometer - wikid, the industrial design engineering wiki. http://wikid.io.tudelft.nl/WikID/index.php/Hall_effect_Magnetometer, unpublished manuscript.

[5] Pengertian sensor efek hall (hall effect sensor) dan prinsip kerjanya - teknik elektronika. https://teknikelektronika.com/pengertian-sensor-efek-hall-hall-effect-sensor-prinsip-kerja-efek-hall/, unpublished manuscript.

[6] Ps-mpu-9250a-01-v1.1.pdf. https://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf, unpublished manuscript.

[7] Nmea 0183 - wikipedia. https://en.wikipedia.org/wiki/NMEA_0183, unpublished manuscript.

[8] Rs-mpu-6000a-00. https://cdn.sparkfun.com/assets/learn_tutorials/5/5/0/MPU-9250-Register-Map.pdf, unpublished manuscript.

[9] Calibrating an ecompass in the presence of hard and soft iron interference. http://cache.freescale.com/files/sensors/doc/app_note/AN4246.pdf, unpublished manuscript.

[10] Atmega48/88/168. https://www.sparkfun.com/datasheets/Components/SMD/ATMega328.pdf, unpublished manuscript.

[11] Faq — kionix - global. https://www.kionix.com/faq, unpublished manuscript.

[12] lecture9.pdf. https://stanford.edu/class/ee267/lectures/lecture9.pdf, unpublished manuscript.

[13] Tilt sensing using linear accelerometers. https://www.nxp.com/docs/en/application-note/AN3461.pdf, unpublished manuscript.

[14] Implementing a tilt-compensated ecompass using accelerometer and magnetometer sensors. https://cache.freescale.com/files/sensors/doc/app_note/AN4248.pdf, unpublished manuscript.

[15] Sebastian Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. *Report x-io and University of Bristol (UK)*, 25:113–118, 2010.

[16] 1704.06053.pdf. https://arxiv.org/pdf/1704.06053.pdf, unpublished manuscript.

[17] Ps-mpu-9250a-01-v1.1.pdf. http://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf, unpublished manuscript.

[18] Flight instruments. http://stickyfish.dk/Flight%20Instruments.html, unpublished manuscript.

[19] kriswiner/mpu9250: Arduino sketches for mpu9250 9dof with ahrs sensor fusion. https://github.com/kriswiner/MPU9250, unpublished manuscript.

[20] Ncei geomagnetic calculators. https://www.ngdc.noaa.gov/geomag/calculators/magcalc.shtml, unpublished manuscript.

[21] Arduino powered autonomous vehicle: 12 steps (with pictures). https://www.instructables.com/id/Arduino-Powered-Autonomous-Vehicle/, unpublished manuscript.

[22] Linear kalman filters - matlab & simulink. https://www.mathworks.com/help/driving/ug/linear-kalman-filters.html, unpublished manuscript.

[23] File:kalman-filter en.svg - wikimedia commons. https://commons.wikimedia.org/wiki/File:Kalman-filter_en.svg#globalusage, unpublished manuscript.

[24] Niklas Magnusson and Tobias Odenman. Improving absolute position estimates of an automotive vehicle using gps in sensor fusion. 2012.

[25] Differential gps - wikipedia. https://en.wikipedia.org/wiki/Differential_GPS#European_DGPS_Network, unpublished manuscript.

[26] Uart explained – digilent inc. blog. https://blog.digilentinc.com/uart-explained/, unpublished manuscript.

[27] Introduction to spi interface — analog devices. https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html, unpublished manuscript.

[28] 'i2c' tag wiki - stack overflow. https://stackoverflow.com/tags/i2c/info, unpublished manuscript.

[29] Understanding the i2c bus. http://www.ti.com/lit/an/slva704/slva704.pdf, unpublished manuscript.

[30] Serial peripheral interface - wikipedia. https://en.wikipedia.org/wiki/Serial_Peripheral_Interface#Data_transmission, unpublished manuscript.

[31] Arduino nrf24l01+ communications - ben fraser - medium. https://medium.com/@benjamindavidfraser/arduino-nrf24l01-communications-947e1acb33fb, unpublished manuscript.

[32] Euler angles - wikipedia. https://en.wikipedia.org/wiki/Euler_angles#Tait-Bryan_angles, unpublished manuscript.

[33] Gimbal lock - wikipedia. https://en.wikipedia.org/wiki/Gimbal_lock, unpublished manuscript.

[34] attitude.pdf. https://www.astro.rug.nl/software/kapteyn-beta/_downloads/attitude.pdf, unpublished manuscript.

A Communication interfaces and protocols

Here are presented briefly the three multi-wire serial data transmission formats that are used in this project.

A.1 UART interface

UART (Universal Asynchronous Receiver Transmitter) refers to an asynchronous serial communication with configurable data format and transmission speeds. A UART system provides a full-duplex communication with only two signals: Tx (transmitted serial data), Rx (received serial data).

Figure 33: UART interface principle

In contrast to other protocols such as SPI and I^2C, no clock signal is required. Indeed, both receiver and transmitter have internal clock signals that govern how the changing logic levels are generated (on the Tx side) and interpreted (on the Rx side). UART uses "Start Bits" and "Stop Bits" to signify the beginning and end of a bit-wise data packet. These bits do not carry any of the transmitting/receiving data. The signal lines typically idle at a logic high level, so the Start Bit is typically a logic low level and the Stop bit is typically the logic high level. When the receiver detects the drop in voltage, it knows to start collecting data. At the end of the data stream, the signal is driven back to a high logic level to indicate the end of the data packet [26].

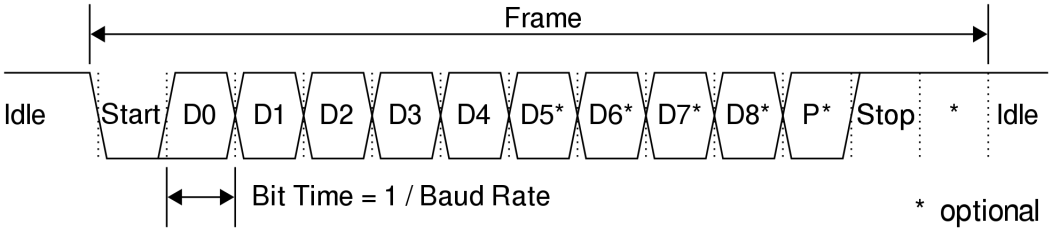


Figure 34: UART frame structure [26]

A.2 SPI interface

SPI (Serial Peripheral Interface) is a four-wire serial bus with a master-slave architecture. It can be also described as a synchronous serial interface contrary to the UART communication

interface for which there is no guarantee that both sides are working at the same rate. A synchronous data bus means that it uses separate lines for data and a clock that keeps both sides synchronized. The clock is an oscillating signal that tells the receiver exactly when to sample the bits on the data line. This could be the rising (low to high) or falling (high to low) edge of the clock signal. When the receiver detects that edge, it will immediately look at the data line to read the next bit [27]. The SPI bus uses four logic signals as follows [30]:

- SCK : Serial Clock (output from master). The side that generates the clock is the master, and the other side is the slave.
- MOSI : Master Output Slave Input (data output from master). This line is used when data is sent from the master to a slave.
- MISO : Master Input Slave Output (data output from master). This line is used when data is sent from a slave to the master.
- CS or SS : Chip Select or Slave Select (data output from master). This line is used instead of an addressing concept. It tells the slave that it should wake up and receive or send data.

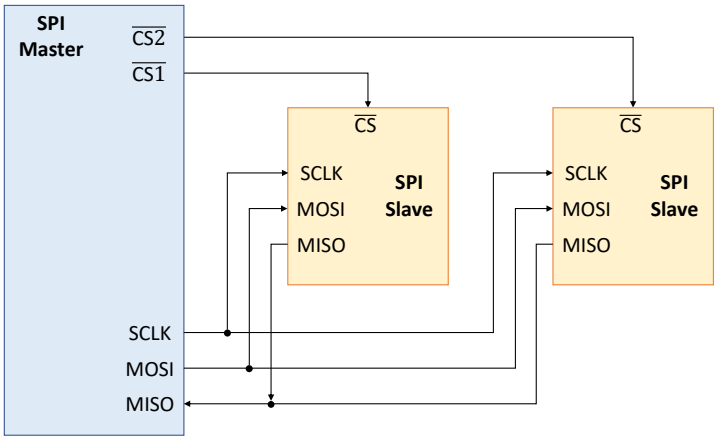


Figure 35: Multislave SPI configuration

In the SPI interface, it is possible to select the clock polarity and clock phase. The "CPOL" bit sets the polarity of the clock signal during the idle state. The idle state is defined as the period when CS is high and transitioning to low at the start of the transmission and when CS is low and transitioning to high at the end of the transmission. Data exchange can be done only is this idling state. The "CPHA" bit selects the clock phase. Depending on the CPHA bit, the rising or falling clock edge is used to sample and/or shift the data [27].

A.3 I^2C interface

The I^2C (Inter-Integrated Circuit) bus has been designed to ease the communication between several components on the same board. It is a bidirectional interface that uses a controller,

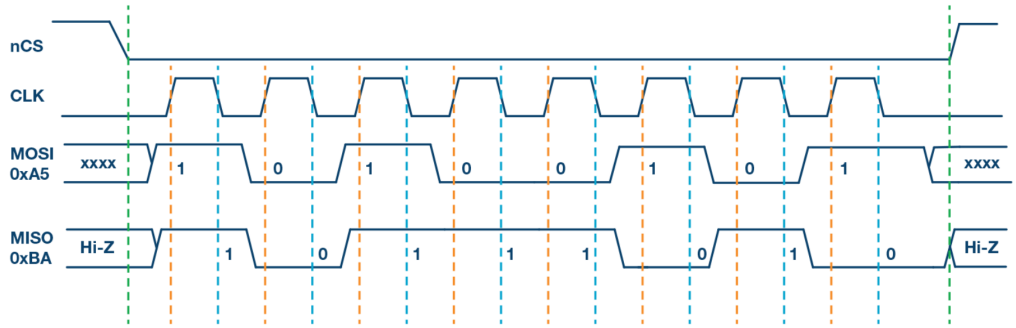


Figure 36: Example of SPI bus configuration (idle state is low, data is sampled on the falling edge) [27]

known as the master, to communicate with slave devices. A slave may not transmit data unless it has been addressed by the master. Each device on the bus has a specific device address to differentiate between other devices. A device can have one or multiple registers where data is stored, written or read [29]. The physical I^2C interface consists of the serial clock (SCL) and serial data (SDA) lines. Both SDA and SCL lines must be connected to V_{CC} through a pull-up resistor.

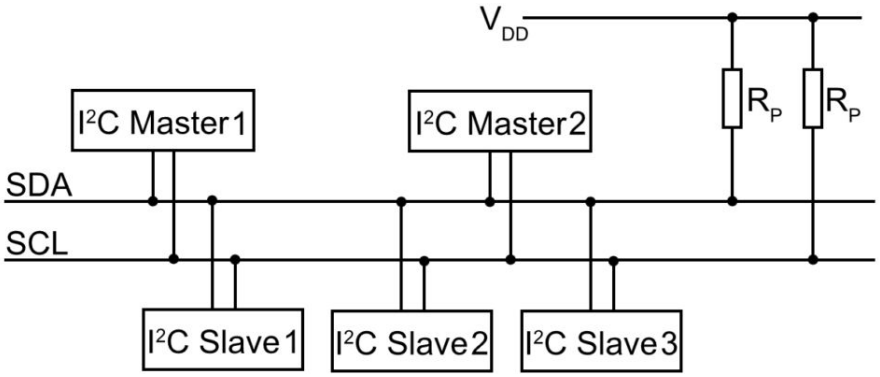


Figure 37: I^2C bus [28]

Data transfer may be initiated only when the bus is idle. A bus is considered idle if both SDA and SCL lines are high after a STOP condition [29].

With I^2C, data is transferred in messages. Messages are broken up into frames of data. Each message has an address frame that contains the binary address of the slave, and one or more data frames that contain the data being transmitted.

The message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame [28].

A.4 Enhanced ShockBurst protocol

In addition to the preamble, the address, the payload and the Cyclic Redundancy Check (CRC), the packet structure of this protocol also includes a Packet Control Field (PCF). This (PCF) contains itself a 6-bits payload length, a 2-bits packet ID and a 1-bit acknowledgment.

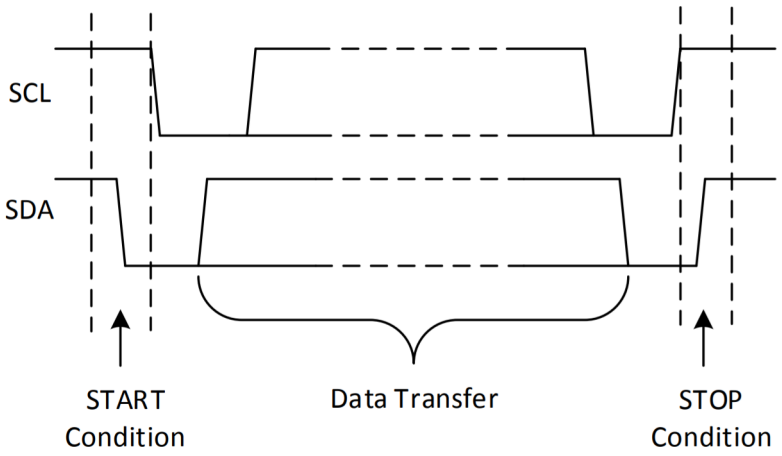


Figure 38: Communication idling condition [29]

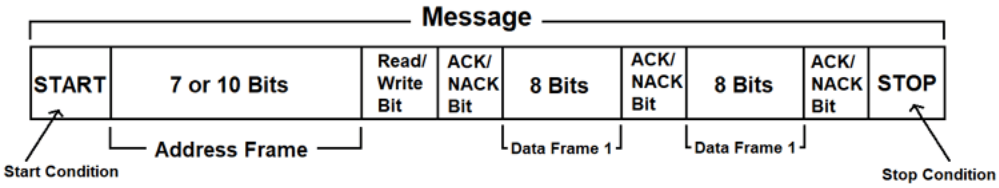


Figure 39: Complete I^2C frame [29]

This specific feature allows "variable length payloads with a payload length specifier, meaning payloads can vary from 1 to 32 bytes. Secondly, it provides each sent packet with a packet ID, which allows the receiving device to determine whether a message is new or whether it has been re-transmitted (and thus can be ignored). Finally, and most importantly, each message can request an acknowledgement to be sent when it is received by another device. This acknowledgement message can contain a pre-loaded payload, meaning a system of bi-directional communications can be established entirely using a master device that transmits data requests to each slave device in succession" [31].

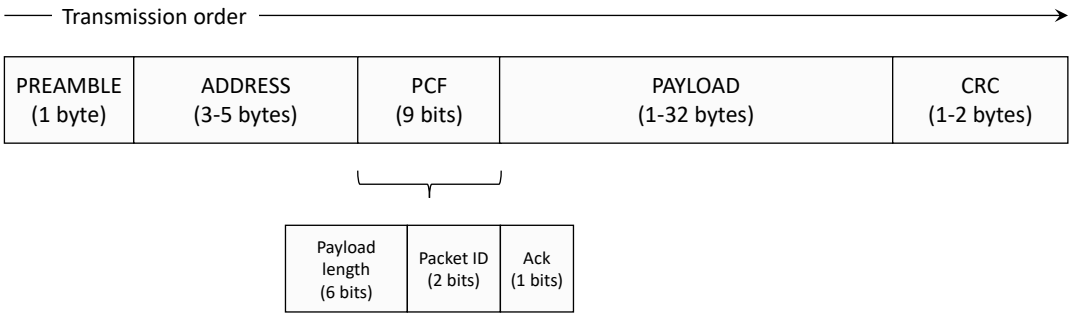


Figure 40: Enhanced ShockBurst packet frame

B Sensor fusion theory

B.1 Euler angles

Rotation matrix.

$$\begin{aligned} R_{zyx}^{vu}(\phi,\theta,\psi) &= R_z(\psi)R_y(\theta)R_x(\phi) \\ &= \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \end{aligned} \tag{22}$$

Rotation sequences. There are twelve possible sequences for these rotations, divided in two categories:

- Proper Euler angles: $\{xyx, xzx, yxy, yzy, zxz, zyz\}$
- Tait-Bryan angles: $\{xyz, xzy, yxz, yzx, zxy, zyx\}$

Each of these sequences can be expressed in terms of intrinsic rotations, which means that the elementary rotations occur in the coordinate system of the moving body, or in terms of extrinsic rotations, which means that the elementary rotations are made in the reference coordinate system [32]. The choice of the sequence directly influences the overall rotation matrix R^{uv} and will be discussed in 3.2. Despite the fact that Euler angles and the use of rotation matrices can be directly used to determine the orientation of our IMU, this system can suffer from "gimbal lock". It is the loss of a degree of freedom in a 3-gimbals system that occurs when two gimbals are successively rotated so that they are aligned on the same plane. For instance, a plane that pitches up 90° so the pitch and yaw gimbals become aligned and the consequence is that any change in the yaw or the roll will cause a rotation around a same axis. Even if some technologies exist to replace gimbals and this defect in gyroscope, the use of numerical tools and especially quaternion methods are the preferred methods to get rid of gimbal lock [33].

B.2 Quaternions

Definition. Quaternions can be described as follows:

$$\mathbf{q} = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}^T = \begin{bmatrix} w \\ \mathbf{v} \end{bmatrix}, \quad \mathbf{q} \in \mathbb{H},\ w \in \mathbb{R},\ \mathbf{v} = \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix} \in \mathbb{R}^3 \tag{23}$$

Multiplication. The quaternion multiplication which will be written $\odot$ is defined as follows:

$$\mathbf{q}, \mathbf{p} \in \mathbb{H}, \quad \mathbf{q} \odot \mathbf{p} = \begin{bmatrix} q_0 p_0 - \mathbf{q}_{1:3}^T \mathbf{p}_{1:3} \\ q_0 \mathbf{p}_{1:3} + p_0 \mathbf{q}_{1:3} + \mathbf{q}_{1:3} \times \mathbf{p}_{1:3} \end{bmatrix}, \tag{24}$$

Unit quaternions. Unit quaternions such as $\|\mathbf{q}\| = 1$ are commonly used in three-dimensional mechanics and especially used in orientation parametrization.

Description of a rotation. Considering a vector $\mathbf{e}^{(\mathbf{u})}$ in the reference coordinates and a vector $\mathbf{e}^{(\mathbf{v})}$ in the body-fixed coordinates, the transformation is given by the following relation [34]:

$$\begin{bmatrix} 0 \\ \mathbf{e}^{(\mathbf{v})} \end{bmatrix} = \mathbf{q} \odot \begin{bmatrix} 0 \\ \mathbf{e}^{(\mathbf{u})} \end{bmatrix} \odot \mathbf{q}^* \tag{25}$$

With the conjugate quaternion $\mathbf{q}^* = \begin{bmatrix} q_w & -\mathbf{v} \end{bmatrix}^T$.

Conversion from Euler angles to quaternions. Considering for example the Euler angle sequence $\{xyz\}$, it is possible to establish formulas between quaternions and these angles [16]. The quaternion can be expressed first in terms of Euler angles:

$$\mathbf{q}_{xyz}(\phi, \theta, \psi) = \begin{bmatrix} c_{\phi/2} c_{\theta/2} c_{\psi/2} + s_{\phi/2} s_{\theta/2} s_{\psi/2} \\ -c_{\phi/2} s_{\theta/2} s_{\psi/2} + c_{\phi/2} c_{\theta/2} s_{\psi/2} \\ c_{\phi/2} c_{\theta/2} s_{\psi/2} + s_{\phi/2} c_{\theta/2} s_{\psi/2} \\ c_{\phi/2} c_{\theta/2} s_{\psi/2} - s_{\phi/2} c_{\theta/2} s_{\psi/2} \end{bmatrix} \tag{26}$$

Conversion from quaternions to Euler angles. The orientation vector that results from the transformation can be then calculated from the quaternion elements:

$$e_{xyz} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \arctan \frac{2(q_0 q_1 + q_2 q_3)}{q_0^2 - q_1^2 - q_2^2 + q_3^2} \\ -\arcsin(2(q_1 q_3 - q_0 q_2)) \\ \arctan \frac{2(q_0 q_3 + q_1 q_2)}{q_0^2 + q_1^2 - q_2^2 - q_3^2} \end{bmatrix} \tag{27}$$

C Codes

In this section are presented the most important programs written for this project. Secondary programs, algorithms and functions are available *here*.

C.1 Arduino codes

C.1.1 Transmitter code

```
1  //  ========== TRANSMITTER ==========
2
3  /**
4   * In order to make this operating file lighter on the Arduino Nano, it contains
5   * the operations that do not need any manipulation of the registers and/or the
6   * operations that calculate and send values to the receiver :
7   *  - RF initialization
8   *  - magnetometer initialization (calculation of its sensitivity adjustement values)
9   *  - get the full scale parameters written in the dedicated registers by the init file
10  *  - send the mag sensitivity and the full scale parameters to the receiver
11  *  - send raw IMU and GPS data
12  */
13
14
15 /**  ||Inspiration||
16  *  - Kris Winer
17  *  - https://forum.hobbycomponents.com/viewtopic.php?f=73&t=1956
18  *  - http://arduinolearning.com/code/arduino-mpu-9250-example.php
19  *  - https://playground.arduino.cc/Main/I2cScanner/
20  *  - https://github.com/sparkfun/SparkFun_Ublox_Arduino_Library
21  *  - https://tmrh20.github.io/RF24/pingpair_ack_8ino-example.html
22  */
23
24
25 /**  ||I2C connections||
26  MPU9250 Breakout --------- Arduino
27  VDD ---------------------- 3.3V
28  SDA ---------------------- A4
29  SCL ---------------------- A5
30  GND ---------------------- GND
31
32  Note: The MPU9250 is an I2C sensor and uses the Arduino Wire library.
33  Because the sensor is not 5V tolerant, we are using a 3.3 V 8 MHz Pro Mini or a 3.3 V
        Teensy 3.1.
34  We have disabled the internal pull-ups used by the Wire library in the Wire.h/twi.c
        utility file.
35  We are also using the 400 kHz fast I2C mode by setting the TWI_FREQ  to 400000L /twi.h
         utility file.
36  */
37
38
39 /**  ||Built-in low-pass filter configuration||
40  * The DLPF_CFG parameter sets the digital low pass filter configuration. It
41  * also determines the internal sampling rate used by the device as shown in
42  * the table below.
43  *
```

```
 * Note: The accelerometer output rate is 1kHz. This means that for a Sample
 * Rate greater than 1kHz, the same accelerometer sample may be output to the
 * FIFO, DMP, and sensor registers more than once.
 *
 * <pre>10
 *          |   ACCELEROMETER    |           GYROSCOPE
 * DLPF_CFG | Bandwidth | Delay  | Bandwidth | Delay  | Sample Rate
 * ---------+-----------+--------+-----------+--------+-------------
 * 0        | 260Hz     | 0ms    | 256Hz     | 0.98ms | 8kHz
 * 1        | 184Hz     | 2.0ms  | 188Hz     | 1.9ms  | 1kHz
 * 2        | 94Hz      | 3.0ms  | 98Hz      | 2.8ms  | 1kHz
 * 3        | 44Hz      | 4.9ms  | 42Hz      | 4.8ms  | 1kHz
 * 4        | 21Hz      | 8.5ms  | 20Hz      | 8.3ms  | 1kHz
 * 5        | 10Hz      | 13.8ms | 10Hz      | 13.4ms | 1kHz
 * 6        | 5Hz       | 19.0ms | 5Hz       | 18.6ms | 1kHz
 * 7        |   -- Reserved --   |   -- Reserved --   | Reserved
 */

/**  ||IMU registers management||
 *   Scan only valid addresses (8 to 0x7B)
 *   0x00 - Reserved - General Call Address
 *   0x01 - Reserved for CBUS Compatibility
 *   0x02 - Reserved for I2C-compatible Bus Variants
 *   0x03 - Reserved for Future Use
 *   0x04, 0x05, 0x06, 0x07 - Reserved for Hs-mode Master
 *   0x78 0x79 0x7A 0x7B - Reserved for 10-bit I2C Addressing
 *   x7C 0x7D 0x7E 0x7F - Reserved for Future Purposes
 */


/**  ||RF transmission : TRANSMITTER||
 *   3.3V
 *   GND
 *   CE ---------------------- D9
 *   CSN --------------------- D10
 *   MOSI -------------------- D11
 *   MISO -------------------- D12
 *   SCK --------------------- D13
 *   IRQ --------------------- disconnected
 */


//Include Libraries
#include <Wire.h>
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <SparkFun_Ublox_Arduino_Library.h>

// Registers definition
#define MPU9250_ADDRESS    0x68
#define AK8963_ADDRESS     0x0C

#define INFO               0x01
#define AK8963_ST1         0x02  // data ready status bit 0
#define AK8963_XOUT_L      0x03  // data
#define AK8963_XOUT_H      0x04
#define AK8963_YOUT_L      0x05
#define AK8963_YOUT_H      0x06
```

```
103 #define AK8963_ZOUT_L      0x07
104 #define AK8963_ZOUT_H      0x08
105 #define AK8963_ST2         0x09  // Data overflow bit 3 and data read error status bit
        2
106 #define AK8963_CNTL        0x0A  // Power down (0000), single-measurement (0001), self-
        test (1000) and Fuse ROM (1111) modes on bits 3:0
107 #define AK8963_ASTC        0x0C  // Self test control
108 #define AK8963_I2CDIS      0x0F  // I2C disable
109 #define AK8963_ASAX        0x10  // Fuse ROM x-axis sensitivity adjustment value
110 #define AK8963_ASAY        0x11  // Fuse ROM y-axis sensitivity adjustment value
111 #define AK8963_ASAZ        0x12  // Fuse ROM z-axis sensitivity adjustment value
112
113 // Other registers
114 #define XG_OFFSET_H        0x13  // User-defined trim values for gyroscope
115 #define XG_OFFSET_L        0x14
116 #define YG_OFFSET_H        0x15
117 #define YG_OFFSET_L        0x16
118 #define ZG_OFFSET_H        0x17
119 #define ZG_OFFSET_L        0x18
120 #define SMPLRT_DIV         0x19  // Sample rate divider
121 #define CONFIG             0x1A  // Gyroscope built-in LPF
122 #define GYRO_CONFIG        0x1B  // Gyroscope scale configuration
123 #define ACCEL_CONFIG       0x1C  // Accelerometer scale configuration
124 #define ACCEL_CONFIG2      0x1D  // Accelerometer built-in LPF
125 #define FIFO_EN            0x23
126 #define I2C_MST_CTRL       0x24
127 #define I2C_MST_STATUS     0x36
128 #define INT_PIN_CFG        0x37
129 #define INT_ENABLE         0x38
130 #define DMP_INT_STATUS     0x39  // Check DMP interrupt
131 #define INT_STATUS         0x3A
132 #define ACCEL_XOUT_H       0x3B
133 #define ACCEL_XOUT_L       0x3C
134 #define ACCEL_YOUT_H       0x3D
135 #define ACCEL_YOUT_L       0x3E
136 #define ACCEL_ZOUT_H       0x3F
137 #define ACCEL_ZOUT_L       0x40
138 #define TEMP_OUT_H         0x41
139 #define TEMP_OUT_L         0x42
140 #define GYRO_XOUT_H        0x43
141 #define GYRO_XOUT_L        0x44
142 #define GYRO_YOUT_H        0x45
143 #define GYRO_YOUT_L        0x46
144 #define GYRO_ZOUT_H        0x47
145 #define GYRO_ZOUT_L        0x48
146 #define USER_CTRL          0x6A  // Bit 7 enable DMP, bit 3 reset DMP
147 #define PWR_MGMT_1         0x6B  // Device defaults to the SLEEP mode
148 #define PWR_MGMT_2         0x6C
149 #define DMP_BANK           0x6D  // Activates a specific bank in the DMP
150 #define DMP_RW_PNT         0x6E  // Set read/write pointer to a specific start address
        in specified DMP bank
151 #define DMP_REG            0x6F  // Register in DMP from which to read or to which to
        write
152 #define DMP_REG_1          0x70
153 #define DMP_REG_2          0x71
154 #define FIFO_COUNTH        0x72
155 #define FIFO_COUNTL        0x73
156 #define FIFO_R_W           0x74
157 #define WHO_AM_I_MPU9250   0x75  // Should return 0x71
```

```
158 #define XA_OFFSET_H         0x77
159 #define XA_OFFSET_L         0x78
160 #define YA_OFFSET_H         0x7B
161 #define YA_OFFSET_L         0x7C
162 #define ZA_OFFSET_H         0x7D
163 #define ZA_OFFSET_L         0x7E
164
165 // Choice of the data range for gyroscope on bits 3 and 4: register 27 (0x1B)
166 #define GYRO_FULL_SCALE_250_DPS 0x00   // 00
167 #define GYRO_FULL_SCALE_500_DPS 0x08   // 01
168 #define GYRO_FULL_SCALE_1000_DPS 0x10  // 10
169 #define GYRO_FULL_SCALE_2000_DPS 0x18  // 11
170
171 // Choice of the data range for accelerometer on bits 3 and 4: register 28 (0x1C)
172 #define ACC_FULL_SCALE_2_G 0x00        // 00
173 #define ACC_FULL_SCALE_4_G 0x08        // 01
174 #define ACC_FULL_SCALE_8_G 0x10        // 10
175 #define ACC_FULL_SCALE_16_G 0x18       // 11
176
177 // Choice for magnetometer parameters
178 #define MAG_RES_14 0x00    // 14 bits magnetometer resolution
179 #define MAG_RES_16 0x01    // 16 bits  magnetometer resolution
180 #define MAG_FREQ_8 0x02    // 8Hz continuous magnetometer data read
181 #define MAG_FREQ_100 0x06  // 100Hz continuous magnetometer data read
182
183 // Initial parameters
184 float response[6];                                          // holds results of gyro
        and accelerometer self test
185 int16_t accelData[3], gyroData[3], magData[3];
186 float gyroBias[3] = {0, 0, 0}, accelBias[3] = {0, 0, 0};    // bias corrections for
        gyro and accelerometer
187 float listSensMag[3] = {0, 0, 0};                           // factory mag calibration
        and mag bias
188
189 // Dashboard
190 uint8_t Gscale = GYRO_FULL_SCALE_500_DPS;  // Choose gyroscope full scale
191 uint8_t Ascale = ACC_FULL_SCALE_4_G;       // Choose accelerometer full scale
192 uint8_t Mscale = MAG_RES_16;               // Choose either 14-bit or 16-bit
        magnetometer resolution
193 uint8_t Mmode = MAG_FREQ_100;              // 2 for 8 Hz, 6 for 100 Hz continuous
        magnetometer data read
194 int aResOut, gResOut, mResOut;             // scale resolutions per LSB for the sensors
195
196 // Create an RF24 object and parameters
197 RF24 radio(9, 10);  // CE, CSN
198 const uint64_t pipes[2] = { 0xABCDABCD71LL, 0x544d52687CLL };     // radio pipe
        addresses for the 2 nodes to communicate.
199
200 // Create a GPS object and parameters
201 SFE_UBLOX_GPS myGPS;
202 long latitude, longitude, speed, heading, SIV;
203 int lat_int, lat_dec, lng_int, lng_dec, vel_int, head_int, alt_int;
204 long lastTime1 = 0; //Simple local timer. Limits amount if I2C traffic to Ublox module.
205 long lastTime2 = 0; //Simple local timer. Limits amount if I2C traffic to Ublox module.
206 long startTime = 0; //Used to calc the actual update rate.
207
208
209 //////////////////////////////////////////////////////////////////////////////////
210
```

```
211
212 void calibrateMPU(float * dest1, float * dest2)
213 // Function that accumulates gyro and accelerometer data after device initialization.
        It calculates the average
214 // of the at-rest readings and then loads the resulting offsets into accelerometer and
        gyro bias registers.
215 {
216   uint8_t data[12]; // data array to hold accelerometer x, y, z and gyro x, y, z, data
217   uint16_t ii, packet_count, fifo_count;
218   int32_t gyro_bias[3]  = {0, 0, 0}, accel_bias[3] = {0, 0, 0};
219
220  // Reset device is done in register 107 (0x6B)
221   I2CwriteByte(MPU9250_ADDRESS, PWR_MGMT_1, 0x80);  // Write a one to bit 7 reset bit
         (1000 0000)
222   delay(100);
223
224  // Get stable time source is done in register 107 (0x6B)
225  // else use the internal oscillator, bits 2:0 = 001
226   I2CwriteByte(MPU9250_ADDRESS, PWR_MGMT_1, 0x01);  // Write a one to bit 1 (0000 0001)
        to auto select the best available clock source
227   I2CwriteByte(MPU9250_ADDRESS, PWR_MGMT_2, 0x00);  // Set on accelerometer x, y, z and
        gyroscope x, y, z
228   delay(200);
229
230 // Configure device for bias calculation
231   I2CwriteByte(MPU9250_ADDRESS, INT_ENABLE, 0x00);   // Disable all interrupts
232   I2CwriteByte(MPU9250_ADDRESS, FIFO_EN, 0x00);      // Disable FIFO
233   I2CwriteByte(MPU9250_ADDRESS, PWR_MGMT_1, 0x00);   // Turn on internal clock source
234   I2CwriteByte(MPU9250_ADDRESS, I2C_MST_CTRL, 0x00); // Disable I2C master
235   I2CwriteByte(MPU9250_ADDRESS, USER_CTRL, 0x00);    // Disable FIFO and I2C master
         modes
236   I2CwriteByte(MPU9250_ADDRESS, USER_CTRL, 0x0C);    // Reset FIFO and DMP
237   delay(15);
238
239 // Configure gyroscope and accelerometer for bias calculation
240   I2CwriteByte(MPU9250_ADDRESS, CONFIG, 0x01);       // Set low-pass filter to 188 Hz
241   I2CwriteByte(MPU9250_ADDRESS, SMPLRT_DIV, 0x00);   // Set sample rate to 1 kHz
242   I2CwriteByte(MPU9250_ADDRESS, GYRO_CONFIG, 0x00);  // Set gyro full-scale to 250
         degrees per second, maximum sensitivity
243   I2CwriteByte(MPU9250_ADDRESS, ACCEL_CONFIG, 0x00); // Set accelerometer full-scale to
        2 g, maximum sensitivity
244
245   uint16_t  gyrosensitivity  = 131;   // = 131 LSB/degrees/sec at gyro full scale
         GFS_SEL = 00 (250dps)
246   uint16_t  accelsensitivity = 16384;  // = 16384 LSB/g at acc full scale AFS_SEL = 00
         (2g)
247
248   // Configure FIFO to capture accelerometer and gyro data for bias calculation
249   I2CwriteByte(MPU9250_ADDRESS, USER_CTRL, 0x40);   // Enable FIFO
250   I2CwriteByte(MPU9250_ADDRESS, FIFO_EN, 0x78);     // Enable gyro and accelerometer
         sensors for FIFO  (max size 512 bytes in MPU-9150)
251   delay(40); // accumulate 40 samples in 40 milliseconds = 480 bytes
252
253 // At end of sample accumulation, turn off FIFO sensor read
254   I2CwriteByte(MPU9250_ADDRESS, FIFO_EN, 0x00);            // Disable gyro and
         accelerometer sensors for FIFO
255   I2CreadBytes(MPU9250_ADDRESS, FIFO_COUNTH, 2, &data[0]);  // Read FIFO sample count
256   fifo_count = ((uint16_t)data[0] << 8) | data[1];
```

```
257   packet_count = fifo_count/12;// How many sets of full gyro and accelerometer data for
          averaging
258   for (ii = 0; ii < packet_count; ii++)
259   {
260     int16_t accel_temp[3] = {0, 0, 0}, gyro_temp[3] = {0, 0, 0};
261     I2CreadBytes(MPU9250_ADDRESS, FIFO_R_W, 12, &data[0]); // Read data for averaging
262     accel_temp[0] = (int16_t) (((int16_t)data[0] << 8) | data[1]  ) ;  // Form signed
            16-bit integer for each sample in FIFO
263     accel_temp[1] = (int16_t) (((int16_t)data[2] << 8) | data[3]  ) ;
264     accel_temp[2] = (int16_t) (((int16_t)data[4] << 8) | data[5]  ) ;
265     gyro_temp[0]  = (int16_t) (((int16_t)data[6] << 8) | data[7]  ) ;
266     gyro_temp[1]  = (int16_t) (((int16_t)data[8] << 8) | data[9]  ) ;
267     gyro_temp[2]  = (int16_t) (((int16_t)data[10] << 8) | data[11]) ;
268
269     accel_bias[0] += (int32_t) accel_temp[0]; // Sum individual signed 16-bit biases to
            get accumulated signed 32-bit biases
270     accel_bias[1] += (int32_t) accel_temp[1];
271     accel_bias[2] += (int32_t) accel_temp[2];
272     gyro_bias[0]  += (int32_t) gyro_temp[0];
273     gyro_bias[1]  += (int32_t) gyro_temp[1];
274     gyro_bias[2]  += (int32_t) gyro_temp[2];
275     }
276
277     accel_bias[0] /= (int32_t) packet_count; // Normalize sums to get average count
            biases
278     accel_bias[1] /= (int32_t) packet_count;
279     accel_bias[2] /= (int32_t) packet_count;
280     gyro_bias[0]  /= (int32_t) packet_count;
281     gyro_bias[1]  /= (int32_t) packet_count;
282     gyro_bias[2]  /= (int32_t) packet_count;
283
284   if(accel_bias[2] > 0L) {accel_bias[2] -= (int32_t) accelsensitivity;}  // Remove
          gravity from the z-axis accelerometer bias calculation
285   else {accel_bias[2] += (int32_t) accelsensitivity;}
286
287 // Construct the gyro biases for push to the hardware gyro bias registers, which are
        reset to zero upon device startup
288   data[0] = (-gyro_bias[0]/4  >> 8) & 0xFF; // Divide by 4 to get 32.9 LSB per deg/s to
          conform to expected bias input format
289   data[1] = (-gyro_bias[0]/4)       & 0xFF; // Biases are additive, so change sign on
          calculated average gyro biases
290   data[2] = (-gyro_bias[1]/4  >> 8) & 0xFF;
291   data[3] = (-gyro_bias[1]/4)       & 0xFF;
292   data[4] = (-gyro_bias[2]/4  >> 8) & 0xFF;
293   data[5] = (-gyro_bias[2]/4)       & 0xFF;
294
295 // Push gyro biases to hardware registers
296   I2CwriteByte(MPU9250_ADDRESS, XG_OFFSET_H, data[0]);
297   I2CwriteByte(MPU9250_ADDRESS, XG_OFFSET_L, data[1]);
298   I2CwriteByte(MPU9250_ADDRESS, YG_OFFSET_H, data[2]);
299   I2CwriteByte(MPU9250_ADDRESS, YG_OFFSET_L, data[3]);
300   I2CwriteByte(MPU9250_ADDRESS, ZG_OFFSET_H, data[4]);
301   I2CwriteByte(MPU9250_ADDRESS, ZG_OFFSET_L, data[5]);
302
303 // Output scaled gyro biases for display in the main program
304   dest1[0] = (float)gyro_bias[0]/(float)gyrosensitivity;
305   dest1[1] = (float)gyro_bias[1]/(float)gyrosensitivity;
306   dest1[2] = (float)gyro_bias[2]/(float)gyrosensitivity;
307
```

```
308 // Construct the accelerometer biases for push to the hardware accelerometer bias
        registers. These registers contain
309 // factory trim values which must be added to the calculated accelerometer biases; on
        boot up these registers will hold
310 // non-zero values. In addition, bit 0 of the lower byte must be preserved since it is
        used for temperature
311 // compensation calculations. Accelerometer bias registers expect bias input as 2048
        LSB per g, so that
312 // the accelerometer biases calculated above must be divided by 8.
313   int16_t accel_bias_reg[3] = { 0, 0, 0 };   // A place to hold the factory
        accelerometer trim biases
314   int16_t mask_bit[3] = { 1, 1, 1 };         // Define array to hold mask bit for each
        accelerometer bias axis
315   I2CreadBytes(MPU9250_ADDRESS, XA_OFFSET_H, 2, &data[0]);   // Read factory
        accelerometer trim values
316   accel_bias_reg[0] = ((int16_t)data[0] << 8) | data[1];
317   I2CreadBytes(MPU9250_ADDRESS, YA_OFFSET_H, 2, &data[0]);
318   accel_bias_reg[1] = ((int16_t)data[0] << 8) | data[1];
319   I2CreadBytes(MPU9250_ADDRESS, ZA_OFFSET_H, 2, &data[0]);
320   accel_bias_reg[2] = ((int16_t)data[0] << 8) | data[1];
321
322   // Construct total accelerometer bias, including calculated average accelerometer
        bias from above
323   for (int i = 0; i < 3; i++)
324   {
325     if (accel_bias_reg[i] % 2)
326     {
327       mask_bit[i] = 0;
328       }
329     accel_bias_reg[i] -= accel_bias[i] >> 3; // Subtract calculated averaged
          accelerometer bias scaled to 2048 LSB/g
330     if (mask_bit[i])
331     {
332       accel_bias_reg[i] = accel_bias_reg[i] & ~mask_bit[i]; // Preserve temperature
            compensation bit
333       }
334     else
335     {
336       accel_bias_reg[i] = accel_bias_reg[i] | 0x0001; // Preserve temperature
            compensation bit
337      }
338   }
339
340   data[0] = (accel_bias_reg[0] >> 8) & 0xFF;
341   data[1] = (accel_bias_reg[0]) & 0xFF;
342   data[2] = (accel_bias_reg[1] >> 8) & 0xFF;
343   data[3] = (accel_bias_reg[1]) & 0xFF;
344   data[4] = (accel_bias_reg[2] >> 8) & 0xFF;
345   data[5] = (accel_bias_reg[2]) & 0xFF;
346
347 // IT DOES NOT WORK
348 //  // Push accelerometer biases to hardware registers
349 //  I2CwriteByte(MPU9250_ADDRESS, XA_OFFSET_H, data[0]);
350 //  I2CwriteByte(MPU9250_ADDRESS, XA_OFFSET_L, data[1]);
351 //  I2CwriteByte(MPU9250_ADDRESS, YA_OFFSET_H, data[2]);
352 //  I2CwriteByte(MPU9250_ADDRESS, YA_OFFSET_L, data[3]);
353 //  I2CwriteByte(MPU9250_ADDRESS, ZA_OFFSET_H, data[4]);
354 //  I2CwriteByte(MPU9250_ADDRESS, ZA_OFFSET_L, data[5]);
355
```

```
356 // Output scaled accelerometer biases for display in the main program
357    dest2[0] = (float)accel_bias[0]/(float)accelsensitivity;
358    dest2[1] = (float)accel_bias[1]/(float)accelsensitivity;
359    dest2[2] = (float)accel_bias[2]/(float)accelsensitivity;
360 }
361
362
363 void initMPU()
364 // Procedure:
365 // - read previous settings
366 // - clear everything except reserved bits
367 // - write new settings
368 {
369   // wake up device
370   I2CwriteByte(MPU9250_ADDRESS, PWR_MGMT_1, 0x00);  // clear sleep mode bit (6), enable
          all sensors
371   delay(100); // wait for all registers to reset
372
373   // get stable time source
374   I2CwriteByte(MPU9250_ADDRESS, PWR_MGMT_1, 0x01);  // auto selects the best available
         clock source, PLL if ready, else use the Internal oscillator
375   delay(200);
376
377   // Set accelerometers low pass filter at 5Hz and sample rate at 1kHz. Accelerometer
         built-in LPF is controlled in register 29 (0x1D)
378   uint8_t b = I2CreadByte(MPU9250_ADDRESS, ACCEL_CONFIG2);
379   // ! bit [4:7] are reserved !
380   I2CwriteByte(MPU9250_ADDRESS, ACCEL_CONFIG2, b & ~0x0F);    // Clear accel_fchoice_b
         bit [3] and A_DLPFG bits [2:0]
381   I2CwriteByte(MPU9250_ADDRESS,ACCEL_CONFIG2, 0x06);      // Set accelerometer rate to
         1 kHz and bandwidth to 5 Hz
382   //I2CwriteByte(MPU9250_ADDRESS,ACCEL_CONFIG2, (0xF0 & b) | 0x0E);      // Set
         accelerometer to normal mode and rate to 1 kHz and bandwidth to 5 Hz ((11110000 &
          b)|1110))
383
384   // Set gyroscope low pass filter bw = 5Hz and sample rate at 1kHz. Gyroscope built-in
          LPF is controlled in register 26 (0x1A)
385   uint8_t a = I2CreadByte(MPU9250_ADDRESS, CONFIG);
386   // ! bit [7] is reserved !
387   I2CwriteByte(MPU9250_ADDRESS,CONFIG,0x06);
388   //I2CwriteByte(MPU9250_ADDRESS, CONFIG, (0x40 & a) | 0x06);  //((10000000 & a)|110)
389
390   // Configure gyroscope scale. Scale selection is done in register 27 (0x1B)
391   uint8_t c = I2CreadByte(MPU9250_ADDRESS, GYRO_CONFIG);    // Set gyroscope rate to 1
         kHz and bandwidth to 5 Hz
392   // ! bit [2] is reserved !
393   I2CwriteByte(MPU9250_ADDRESS, GYRO_CONFIG, c & ~0xE0);          // Clear self-test
         bits [7:5]
394   I2CwriteByte(MPU9250_ADDRESS, GYRO_CONFIG, c & ~0x02);          // Clear Fchoice bits
          [1:0]
395   I2CwriteByte(MPU9250_ADDRESS, GYRO_CONFIG, c & ~0x18);          // Clear AFS bits
         [4:3]
396   //I2CwriteByte(MPU9250_ADDRESS, GYRO_CONFIG, Gscale);    // Set full scale range for
         the gyro
397   I2CwriteByte(MPU9250_ADDRESS, GYRO_CONFIG, (0x04 & c) | Gscale);    // Set full scale
          range for the gyro ((100 & c)|1000)
398
399   // Configure accelerometer scale. Scale selection is done in register 28 (0x1C)
400   uint8_t d = I2CreadByte(MPU9250_ADDRESS, ACCEL_CONFIG);
```

```
401   // ! bit [2:0] are reserved !
402   I2CwriteByte(MPU9250_ADDRESS, ACCEL_CONFIG, d & ~0xE0);         // Clear self-test
          bits [7:5]
403   I2CwriteByte(MPU9250_ADDRESS, ACCEL_CONFIG, d & ~0x18);         // Clear AFS bits
          [4:3]
404   //I2CwriteByte(MPU9250_ADDRESS,ACCEL_CONFIG, Ascale);    // Set full scale range for
          the accelerometer
405   I2CwriteByte(MPU9250_ADDRESS,ACCEL_CONFIG, (0x07 & d) | Ascale);    // Set full scale
           range for the accelerometer ((111 & d)|1000)
406
407   // Divides the internal sample rate (done in register 25=0x19) to generate the sample
           rate that controls sensor data output rate, FIFO sample rate.
408   I2CwriteByte(MPU9250_ADDRESS, SMPLRT_DIV, 0x04);
409     // Set sample rate = gyroscope output rate/(1 + SMPLRT_DIV) = 200Hz
410
411   // The accelerometer and gyro are set to 1 kHz sample rates, but these rates
412   // are further reduced by a factor of 5 to 200 Hz because of the SMPLRT_DIV setting
413
414   // Set by pass mode (done in register 55=0x37).
415   I2CwriteByte(MPU9250_ADDRESS,INT_PIN_CFG,0x22);
416     // 0x02: When asserted, the i2c_master interface pins (ES_CL and ES_DA) will go
           into "bypass mode" when the i2c master interface is disabled.
417     // 0x20: INT pin level held until interrupt status is cleared.
418   I2CwriteByte(MPU9250_ADDRESS, INT_ENABLE, 0x01);  // Enable data ready (bit 0)
          interrupt
419   delay(100);
420 }
421
422
423 void initMAG(float * destination)
424 {
425   // First extract the factory calibration for each magnetometer axis
426   uint8_t rawData[3];  // x, y, z mag sensitivity values stored here
427   I2CwriteByte(AK8963_ADDRESS, AK8963_CNTL, 0x00); // Power down magnetometer
428   delay(100);
429   I2CwriteByte(AK8963_ADDRESS, AK8963_CNTL, 0x0F); // Enter Fuse ROM access mode
430   delay(100);
431   I2CreadBytes(AK8963_ADDRESS, AK8963_ASAX, 3, &rawData[0]);  // Read the x-axis, y-
          axis, and z-axis calibration values
432
433   // Return x-axis, y-axis, and z-axis sensitivity adjustment values (formula in
          datasheet)
434   destination[0] =  (float)(rawData[0] - 128)/256. + 1.;
435   destination[1] =  (float)(rawData[1] - 128)/256. + 1.;
436   destination[2] =  (float)(rawData[2] - 128)/256. + 1.;
437   I2CwriteByte(AK8963_ADDRESS, AK8963_CNTL, 0x00); // Power down magnetometer
438   delay(100);
439   // Configure the magnetometer for continuous read and highest resolution
440   // set Mscale bit 4 to 1 (0) to enable 16 (14) bit resolution in CNTL register,
441   // and enable continuous mode data acquisition Mmode (bits [3:0]), 0010 for 8 Hz and
          0110 for 100 Hz sample rates
442   I2CwriteByte(AK8963_ADDRESS, AK8963_CNTL, Mscale << 4 | Mmode); // Set magnetometer
          data resolution and sample ODR
443   delay(10);
444 }
445
446
447 void getAres()
448 // Possible accelerometer scales (and their register bit settings) are:
```

```
// 2 Gs (00), 4 Gs (01), 8 Gs (10), and 16 Gs  (11).
{
  byte a = I2CreadByte(MPU9250_ADDRESS,ACCEL_CONFIG);
  switch (a)
  {
    case ACC_FULL_SCALE_2_G:
          aResOut = 2;
          break;
    case ACC_FULL_SCALE_4_G:
          aResOut = 4;
          break;
    case ACC_FULL_SCALE_8_G:
          aResOut = 8;
          break;
    case ACC_FULL_SCALE_16_G:
          aResOut = 16;
          break;
  }
}


void getGres()
// Possible gyro scales (and their register bit settings) are:
// 250 DPS (00), 500 DPS (01), 1000 DPS (10), and 2000 DPS  (11).
{
  byte b = I2CreadByte(MPU9250_ADDRESS, GYRO_CONFIG);
  switch (b)
  {
    case GYRO_FULL_SCALE_250_DPS:
          gResOut = 250;
          break;
    case GYRO_FULL_SCALE_500_DPS:
          gResOut = 500;
          break;
    case GYRO_FULL_SCALE_1000_DPS:
          gResOut = 1000;
          break;
    case GYRO_FULL_SCALE_2000_DPS:
          gResOut = 2000;
          break;
  }
}


void getMres()
// Possible magnetometer scales (and their register bit settings) are
// 14 bit resolution (0) and 16 bit resolution (1)
{
  switch (Mscale)
  {
    case MAG_RES_14:
          mResOut = 14; // Proper scale to return G (1G=100microT)
          break;
    case MAG_RES_16:
          mResOut = 16; // Proper scale to return G (1G=100microT)
          break;
  }
}

```

```
508
509 // This function read Nbytes bytes from I2C device at address Address.
510 // Put read bytes starting at register Register in the Data array.
511 void I2CreadBytes(uint8_t Address, uint8_t Register, uint8_t Nbytes, uint8_t* Data)
512 {
513   // Set register address
514   Wire.beginTransmission(Address);
515   Wire.write(Register);
516   Wire.endTransmission();
517
518   // Read Nbytes
519   Wire.requestFrom(Address, Nbytes);
520   uint8_t index=0;
521   while (Wire.available()){
522     Data[index++]=Wire.read();
523     }
524 }
525
526
527 uint8_t I2CreadByte(uint8_t address, uint8_t subAddress)
528 {
529   uint8_t data; // `data` will store the register data
530   Wire.beginTransmission(address);         // Initialize the Tx buffer
531   Wire.write(subAddress);                  // Put slave register address in Tx buffer
532   Wire.endTransmission(false);             // Send the Tx buffer, but send a restart to
        keep connection alive
533   Wire.requestFrom(address, (uint8_t) 1);  // Read one byte from slave register address
534   data = Wire.read();                      // Fill Rx buffer with result
535   return data;                             // Return data read from slave register
536 }
537
538
539 void readMag(int16_t * destination)
540 {
541   uint8_t rawData[7];  // x, y, z magnetometer data and register ST2
542   if(I2CreadByte(AK8963_ADDRESS, AK8963_ST1) & 0x01)  // Wait for magnetometer data
        ready bit DRDY to be set (=1)
543   {
544     I2CreadBytes(AK8963_ADDRESS, AK8963_XOUT_L, 7, &rawData[0]);  // Read the six raw
          data and ST2 registers sequentially
545     uint8_t c = rawData[6]; // End data read by reading ST2 register
546     if(!(c & 0x08))  // Check if magnetic sensor overflow set (bit [3] HOFL of ST2
          register = 1 if overflow), if not then report data
547     {
548       destination[0] = ((int16_t)rawData[1] << 8) | rawData[0] ;  // Turn the MSB and
            LSB into a signed 16-bit value
549       destination[1] = ((int16_t)rawData[3] << 8) | rawData[2] ;
550       destination[2] = ((int16_t)rawData[5] << 8) | rawData[4] ;
551     }
552   }
553 }
554
555
556 void readData(int16_t * destination, int16_t reg)
557 {
558   uint8_t rawData[6];  // x/y/z accel register data stored here
559   I2CreadBytes(MPU9250_ADDRESS, reg, 6, rawData);  // Read the six raw data registers
        into data array
```

```
  destination[0] = ((int16_t)rawData[0] << 8) | rawData[1] ;  // Turn the MSB and LSB
      into a signed 16-bit value
  destination[1] = ((int16_t)rawData[2] << 8) | rawData[3] ;
  destination[2] = ((int16_t)rawData[4] << 8) | rawData[5] ;
}


// Write a byte (Data) in device (Address) at register (Register)
void I2CwriteByte(uint8_t Address, uint8_t Register, uint8_t Data)
{
  // Set register address
  Wire.beginTransmission(Address);
  Wire.write(Register);
  Wire.write(Data);
  Wire.endTransmission();
  }


// GPS treatment and sending functions
void coordIntFormat(int &leftInt, int &rightInt, long coord)
{
  float coord_temp = (float)coord/100000.0;
  leftInt = floor(coord_temp);
  rightInt = (coord_temp-leftInt)*10000;
}


int speedIntFormat(long v)
{
  int v_int = floor((float)v/10.0); // in cm/s
  return v_int;
}


int headIntFormat(long h)
{
  int h_int = floor((float)h/10000.0); // in tenth of degrees
  return h_int;
}


int altIntFormat(long a)
{
  int h_int = floor((float)a/1000.0); // in m
  return h_int;
}


void sendRF(int val)
{
  readData(accelData,ACCEL_XOUT_H);
  readData(gyroData,GYRO_XOUT_H);
  readMag(magData);
  int16_t dataOut[16] = {accelData[0], accelData[1], accelData[2],
                         gyroData[0] , gyroData[1] , gyroData[2] ,
                         magData[0]  , magData[1]  , magData[2]  ,
                         lat_int*val , lat_dec*val , lng_int*val , lng_dec*val,
                         head_int*val, vel_int*val , alt_int*val};
  //Serial.println(magData[1]);
```

```
618   radio.write(&dataOut, sizeof(dataOut));
619 }
620
621
622 ////////////////////////////////////////////////////////////////////////////////
623
624
625 void setup()
626 {
627   // Initialization of communication (Serial, I2C, GPS)
628   Serial.begin(115200);
629   while (!Serial)
630   Serial.println(F("---I2C and RF initialization---"));
631   Wire.begin();
632   if (myGPS.begin() == false)  //Connect to the Ublox module using Wire port
633   {
634     Serial.println(F("Ublox GPS not detected at default I2C address. Please check
          wiring. Freezing."));
635     while (1);
636     }
637   myGPS.setI2COutput(COM_TYPE_UBX);      // set the I2C port to output UBX only (turn
       off NMEA noise)
638   myGPS.setNavigationFrequency(10);      // set output to 10 times a second
639   myGPS.setAutoPVT(true);                // enable automatic PVT (Position,Velocity,
       Time etc) reports at the navigation frequency
640   myGPS.saveConfiguration();             // save the current settings to flash and BBR
641
642   byte rate = myGPS.getNavigationFrequency();  // check the update rate of this module
643   Serial.print("Current update rate:");
644   Serial.println(rate);
645
646   // RF communication parameters
647   radio.begin();
648   radio.setAutoAck(1);                    // ensure autoACK is enabled
649   radio.setPALevel(RF24_PA_MAX);          // power amplifier level
650   radio.enableAckPayload();               // allow optional ack payloads
651   radio.setRetries(0,15);                 // smallest time between retries, max no. of
       retries
652   radio.openWritingPipe(pipes[0]);        // both radios listen on the same pipes by
       default, and switch when writing
653   radio.openReadingPipe(1,pipes[1]);
654   radio.stopListening();                  // stop listening
655   Serial.println();
656
657   // Accelerometer-Gyroscope Calibration
658   Serial.println("---Accelerometer and Gyroscope Calibration---");
659   calibrateMPU(gyroBias, accelBias); // Calibrate gyroscope and accelerometer, load
       biases in bias registers
660
661   // Accelerometer-Gyroscope initialization
662   Serial.println(F("---Accelerometer and Gyroscope Initialization ---"));
663   initMPU();
664
665   // Magnetometer initialization
666   initMAG(listSensMag);
667
668   // Transmit calculation parameters (in the transmission file we work with integers,
669   // resolutions and other factors are transmitted here and used in the receiver
       program)
```

```
  Serial.println(F("---Parameters RF transmission---"));
  getAres();
  getGres();
  getMres();
  byte counter = 0;
  int count = 0;
  byte oldByte = 0;
  int array2send[3] = {aResOut, gResOut, mResOut};
  while(count < 3){
    Serial.print(F("Sending: ")); Serial.print((int)array2send[counter]);
    int data2send = array2send[counter];
    if (!radio.write( &data2send, sizeof(data2send) )){
      Serial.println(F("failed."));
    }
    else{
      if(!radio.available()){
        Serial.println();
      }
      else{
        while(radio.available() ){
          byte ackByte;
          radio.read( &ackByte, 1 );
          Serial.print(F(", Response: ")); Serial.println(ackByte);
          if(ackByte != oldByte){
            oldByte = ackByte;
            count++;
            counter++;
          }
        }
      }
    }
    delay(100);
  }

  delay(500);
  Serial.println(F("GO"));

  startTime = millis();
}


void loop()
{
  if (millis() - lastTime2 > 5)
  {
    if (millis() - lastTime1 > 100)
    {
      lastTime1 = millis();  // Update the GPS timer
      lastTime2 = millis();  // Update the IMU timer

      // GPS output are long values so we need to cut them into two integers that can
         be sent in our RF payload.
      coordIntFormat(lat_int, lat_dec, myGPS.getLatitude());  // given at 10^-7 degrees
      coordIntFormat(lng_int, lng_dec, myGPS.getLongitude());  // given at 10^-7
         degrees
      vel_int = speedIntFormat(myGPS.getGroundSpeed());  // given in mm/s
      head_int = headIntFormat(myGPS.getHeading());  // given at 10^-5 degrees
      alt_int = altIntFormat(myGPS.getAltitude());  // given in m

```

```
      //Serial.println(vel_int);
      //Serial.println(alt_int);

      // send data with GPS coordinates
      sendRF(1);
      }
    else
    {
      // send data without GPS coordinates
      lastTime2 = millis();  // Update the IMU timer
      sendRF(0);
      }
    }
}
```

C.1.2 Receiver code

```
// === RECEIVER ===

/**  ||Inspiration||
 *  - Kris Winer
 *  - https://tmrh20.github.io/RF24/pingpair_ack_8ino-example.html
 */

/** ||connection||
* 3.3V
* GROUND
* CE = 9
* SS (CSN) = 10
* MOSI = 11
* MISO = 12
* SCK = 13
* IRQ = DISCONNECTED
*/


//Include Libraries
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

//create an RF24 object
RF24 radio(9, 10);  // CE, CSN

//addresses through which two modules communicate.
const uint64_t pipes[2] = { 0xABCDABCD71LL, 0x544d52687CLL };     // radio pipe
    addresses for the 2 nodes to communicate.
const byte address[6] = "00001";


// Calculation parameters
float aResIn, gResIn, mResIn;
float aRes, gRes, mRes;
float listSensMAGX = 1.23047, listSensMAGY = 1.22656, listSensMAGZ = 1.18359;


```

```
39 ////////////////////////////////////////////////////////////////////////////////////
40
41
42 void getAres() {
43   switch ((int)aResIn)
44   {
45   // Possible accelerometer scales (and their register bit settings) are:
46   // 2 Gs (00), 4 Gs (01), 8 Gs (10), and 16 Gs  (11).
47     case 2:
48           aRes = 2.0/32768.0;
49           break;
50     case 4:
51           aRes = 4.0/32768.0;
52           break;
53     case 8:
54           aRes = 8.0/32768.0;
55           break;
56     case 16:
57           aRes = 16.0/32768.0;
58           break;
59   }
60 }
61
62
63 void getGres()
64 // Possible gyro scales (and their register bit settings) are:
65 // 250 DPS (00), 500 DPS (01), 1000 DPS (10), and 2000 DPS  (11).
66 {
67   switch ((int)gResIn)
68   {
69     case 250:
70           gRes = 250.0/32768.0;
71           break;
72     case 500:
73           gRes = 500.0/32768.0;
74           break;
75     case 1000:
76           gRes = 1000.0/32768.0;
77           break;
78     case 2000:
79           gRes = 2000.0/32768.0;
80           break;
81   }
82 }
83
84
85 void getMres()
86 // Possible magnetometer scales (and their register bit settings) are
87 // 14 bit resolution (0) and 16 bit resolution (1)
88 {
89   switch ((int)mResIn)
90   {
91     case 14:
92           mRes = 10*4912./8190.; // Proper scale to return mG (1microT=0.01G)
93           break;
94     case 16:
95           mRes = 10*4912./32760.0; // Proper scale to return mG (1microT=0.01G)
96           break;
97   }
```

```
}


////////////////////////////////////////////////////////////////////////////////


void setup()
{
  // Initialization of communication (Serial)
  Serial.begin(115200);
  while (!Serial)
  Serial.println("---RF initialization---");

  // RF communication parameters
  radio.begin();
  radio.setAutoAck(1);                    // ensure autoACK is enabled
  radio.setPALevel(RF24_PA_MAX);          // power amplifier level
  radio.enableAckPayload();               // allow optional ack payloads
  radio.setRetries(0,15);                 // smallest time between retries, max no. of
      retries
  radio.openWritingPipe(pipes[1]);        // both radios listen on the same pipes by
      default, and switch when writing
  radio.openReadingPipe(1,pipes[0]);
  radio.startListening();                 // start listening
  radio.printDetails();                   // dump the configuration of the rf unit for
      debugging


  // Receive calculation parameters
  Serial.println("---Parameters RF transmission---");
  int count = 0;
  int oldByte = 0;
  byte ackByte = 1;
  int array2receive[3];
  while(count < 3){
    byte pipeNo;
    int data2receive;                                   // Dump the payloads until
        we've gotten everything
    while(radio.available(&pipeNo)){
      delay(100);
      radio.read( &data2receive, sizeof(data2receive) );
      radio.writeAckPayload(pipeNo,&ackByte, 1 );
      if(data2receive != oldByte){
        Serial.println(data2receive);
        oldByte = data2receive;
        ackByte++;
        array2receive[count] = data2receive;
        count++;
      }
    }
  }

  aResIn = array2receive[0];
  gResIn = array2receive[1];
  mResIn = array2receive[2];
  getAres();
  Serial.println(aRes, 4);
  getGres();
  Serial.println(gRes, 4);
```

```
153   getMres();
154   Serial.println(mRes, 4);
155
156   Serial.println("GO");
157 }
158
159
160 void loop()
161 {
162   int16_t dataIn[16];
163   if (radio.available())
164   {
165     radio.read( &dataIn, sizeof(dataIn));
166
167     // Accelerometer
168     Serial.print((float)dataIn[0]*aRes);  // North -1
169     Serial.print (",");
170     Serial.print((float)dataIn[1]*aRes);  // East -0
171     Serial.print (",");
172     Serial.print((float)dataIn[2]*aRes);   // Down -2
173     Serial.print (";");
174
175     // Gyroscope
176     Serial.print((float)dataIn[3]*gRes);  // -4
177     Serial.print (",");
178     Serial.print((float)dataIn[4]*gRes);   // 3
179     Serial.print (",");
180     Serial.print((float)dataIn[5]*gRes);  //-5
181     Serial.print (";");
182
183     // Magnetometer
184     Serial.print((float)dataIn[6]*mRes*listSensMAGX);  //6
185     Serial.print (",");
186     Serial.print((float)dataIn[7]*mRes*listSensMAGY);  //7
187     Serial.print (",");
188     Serial.print((float)dataIn[8]*mRes*listSensMAGZ);   //-8
189     Serial.print (";");
190
191     // GPS
192     if (dataIn[9]!= 0)
193     {
194       Serial.print((float)dataIn[9]/100.0 + (float)dataIn[10]/1000000.0, 6);  // in
              degree
195       Serial.print (",");
196       Serial.print((float)dataIn[11]/100.0 + (float)dataIn[12]/1000000.0, 6);  // in
              degree
197       Serial.print (",");
198       Serial.print(0);
199       Serial.print (";");
200       Serial.print((float)dataIn[13]/10.0);  // in degree
201       Serial.print (",");
202       Serial.print((float)dataIn[14]/100.0);  // in m/s
203       Serial.print (",");
204       Serial.print(dataIn[15]);  // in m
205       }
206
207     // End of line
208     Serial.println("");
209   }
```

```
}
```

C.2 Matlab codes

C.2.1 Calibration code

```
clear;
close all;
clc;
addpath('mytoolbox');

% "doc" to see a function's way of working
% "edit" to see how is written a function


% Port reset:
% instrfind reads serial port objects from memory to MATLAB workspace
if not(isempty(instrfind))
    fclose(instrfind);
    delete(instrfind);
end

%--------------------------------------------------------------------------
%% Serial open
arduino=serial('COM3','BaudRate',250000);
fopen(arduino);
fs = 200;
T = 1/fs;
fc = 1.5;  % cut off frequency for LPF

y = char.empty;
while strcmp(y,char([71,79,13,10])) ~= 1
    y = char(fscanf(arduino));
    disp(y);
end

%--------------------------------------------------------------------------
%% Magnetometer calibration
% Initialize figure
figure('units','normalized','outerposition',[0.2 0.2 0.8 0.8])
l = animatedline('Color','r','Marker','.');
view(3)
grid on

% Wait for stabilization
t0 = clock;
s = 0;
while s<1
    y = fgets(arduino);
    temp = str2num(y);
    s = etime(clock,t0);
```

```
end

% Data acquisition
disp('==Rotate the device around all the axes==');
mag = zeros([0 3]);
s = 0;
t0 = clock;
while s < 30
    y = fgets(arduino);
    temp = str2num(y);
    size_temp = size(temp);
    if (size_temp >= [3,3])
        mag = cat(1,mag,temp(3,:));
        addpoints(l,mag(end,1),mag(end,2),mag(end,3));
        drawnow limitrate
        s = etime(clock,t0);
    end
end
disp('==STOP==');

% Filter
[b,a] = butter_synth(1,fc,fs);  % from mytoolbox
magB = filter(b,a,mag);         % built-in function

magF = magB;
for i=1:50
    magF(1,:)=[];
end

scalex = (max(magF(:,1))-min(magF(:,1)))/2;
scaley = (max(magF(:,2))-min(magF(:,2)))/2;
scalez = (max(magF(:,3))-min(magF(:,3)))/2;
scale_avg = (scalex+scaley+scalez)/3;
scalex_avg = scale_avg/scalex;
scaley_avg = scale_avg/scaley;
scalez_avg = scale_avg/scalez;
x_avg = (max(magF(:,1))+min(magF(:,1)))/2;
y_avg = (max(magF(:,2))+min(magF(:,2)))/2;
z_avg = (max(magF(:,3))+min(magF(:,3)))/2;

save('mag_calib_values','x_avg','y_avg','z_avg','scalex_avg','scaley_avg','
    scalez_avg');

figure('units','normalized','outerposition',[0.2 0.2 0.8 0.8])
plot3(mag(:,1),mag(:,2),mag(:,3),'Color','r');
hold on
plot3(magF(:,1),magF(:,2),magF(:,3),'Color','b');
hold on
plot3((magF(:,1)-x_avg)*scalex_avg,(magF(:,2)-y_avg)*scaley_avg,(magF(:,3)-
    z_avg)*scalez_avg,'Color','g');
xlabel('m_x (mG)');
ylabel('m_y (mG)');
legend('raw data','after LP filtration','calibrated data');
axis equal
```

```
grid on

%--------------------------------------------------------------------------
%% Test
% Initialization figure
figure('units','normalized','outerposition',[0.2 0.2 0.8 0.8])

[x,y,z] = sphere(40);
x = x*scale_avg; y = y*scale_avg; z = z*scale_avg;
h = surf(x, y, z);
set(h,'edgecolor',[0.5 0.3 0.5],'EdgeAlpha',0.2,'facecolor',[0.8 0.1 0.6],'
    FaceAlpha',.07)
hold on

magplot = animatedline('Color','r','Marker','.'); %,'MaximumNumPoints',2000);
xlim([-700 700]);
ylim([-700 700]);
zlim([-700 700]);
xlabel('mag_x (mG)');
ylabel('mag_y (mG)');
zlabel('mag_z (mG)');
view(3);
grid on
pbaspect([1 1 1]);

% Stabilization
y = char.empty;
s = 0;
t0 = clock;
while s < 1
    y = fgets(arduino);
    temp = str2num(y);
    disp(temp);
    s = etime(clock,t0);
end

% Read
while true
    y = fgets(arduino);
    temp = str2num(y);
    size_temp = size(temp);
    if (size_temp >= [3,3])
        mag = temp(3,:);
        mag = offset_scale(mag,x_avg,y_avg,z_avg,scalex_avg,scaley_avg,
            scalez_avg);
        addpoints(magplot,mag(1),mag(2),mag(3));
        disp(mag);
        drawnow limitrate
    end
end
disp('==STOP==');
```

C.2.2 Madgwick algorithm

```
function result = fusionMadgwick(result, acc, gyr, mag, beta, zeta, deltat)
    q1 = result(1); q2 = result(2); q3 = result(3); q4 = result(4); g_bx =
        result(5); g_by = result(6); g_bz = result(7);   % short name local
        variable for readability

    ax = acc(1);
    ay = acc(2);
    az = acc(3);
    gx = gyr(1);
    gy = gyr(2);
    gz = gyr(3);
    mx = mag(2);
    my = mag(1);
    mz = -mag(3);

    % Auxiliary variables to avoid repeated arithmetic
    two_q1 = 2 * q1;
    two_q2 = 2 * q2;
    two_q3 = 2 * q3;
    two_q4 = 2 * q4;
    two_q1q3 = 2 * q1 * q3;
    two_q3q4 = 2 * q3 * q4;
    q1q1 = q1 * q1;
    q1q2 = q1 * q2;
    q1q3 = q1 * q3;
    q1q4 = q1 * q4;
    q2q2 = q2 * q2;
    q2q3 = q2 * q3;
    q2q4 = q2 * q4;
    q3q3 = q3 * q3;
    q3q4 = q3 * q4;
    q4q4 = q4 * q4;

    % Normalise accelerometer measurement
    norm = sqrt(ax * ax + ay * ay + az * az);
    if (norm == 0)
        error('Norm error calculation');
    end
    ax = ax / norm;
    ay = ay / norm;
    az = az / norm;

    % Normalise magnetometer measurement
    norm = sqrt(mx * mx + my * my + mz * mz);
    if (norm == 0)
        error('Norm error calculation');
    end
    mx = mx / norm;
    my = my / norm;
    mz = mz / norm;
```

```
49
50      % Reference direction of Earth's magnetic field
51      two_q1mx = 2 * q1 * mx;
52      two_q1my = 2 * q1 * my;
53      two_q1mz = 2 * q1 * mz;
54      two_q2mx = 2 * q2 * mx;
55      hx = mx * q1q1 - two_q1my * q4 + two_q1mz * q3 + mx * q2q2 + two_q2 * my *
            q3 + two_q2 * mz * q4 - mx * q3q3 - mx * q4q4;
56      hy = two_q1mx * q4 + my * q1q1 - two_q1mz * q2 + two_q2mx * q3 - my * q2q2
            + my * q3q3 + two_q3 * mz * q4 - my * q4q4;
57      two_bx = sqrt(hx * hx + hy * hy);
58      two_bz = -two_q1mx * q3 + two_q1my * q2 + mz * q1q1 + two_q2mx * q4 - mz *
            q2q2 + two_q3 * my * q4 - mz * q3q3 + mz * q4q4;
59      four_bx = 2 * two_bx;
60      four_bz = 2 * two_bz;
61
62      % Gradient decent algorithm corrective step
63      s1 = -two_q3 * (2 * q2q4 - two_q1q3 - ax) + two_q2 * (2 * q1q2 + two_q3q4
            - ay) - two_bz * q3 * (two_bx * (0.5 - q3q3 - q4q4) + two_bz * (q2q4 -
            q1q3) - mx) + (-two_bx * q4 + two_bz * q2) * (two_bx * (q2q3 - q1q4) +
            two_bz * (q1q2 + q3q4) - my) + two_bx * q3 * (two_bx * (q1q3 + q2q4) +
            two_bz * (0.5 - q2q2 - q3q3) - mz);
64      s2 = two_q4 * (2 * q2q4 - two_q1q3 - ax) + two_q1 * (2 * q1q2 + two_q3q4 -
             ay) - 4 * q2 * (1 - 2 * q2q2 - 2 * q3q3 - az) + two_bz * q4 * (two_bx
            * (0.5 - q3q3 - q4q4) + two_bz * (q2q4 - q1q3) - mx) + (two_bx * q3 +
            two_bz * q1) * (two_bx * (q2q3 - q1q4) + two_bz * (q1q2 + q3q4) - my) +
             (two_bx * q4 - four_bz * q2) * (two_bx * (q1q3 + q2q4) + two_bz * (0.5
             - q2q2 - q3q3) - mz);
65      s3 = -two_q1 * (2 * q2q4 - two_q1q3 - ax) + two_q4 * (2 * q1q2 + two_q3q4
            - ay) - 4 * q3 * (1 - 2 * q2q2 - 2 * q3q3 - az) + (-four_bx * q3 -
            two_bz * q1) * (two_bx * (0.5 - q3q3 - q4q4) + two_bz * (q2q4 - q1q3) -
             mx) + (two_bx * q2 + two_bz * q4) * (two_bx * (q2q3 - q1q4) + two_bz *
             (q1q2 + q3q4) - my) + (two_bx * q1 - four_bz * q3) * (two_bx * (q1q3 +
             q2q4) + two_bz * (0.5 - q2q2 - q3q3) - mz);
66      s4 = two_q2 * (2 * q2q4 - two_q1q3 - ax) + two_q3 * (2 * q1q2 + two_q3q4 -
             ay) + (-four_bx * q4 + two_bz * q2) * (two_bx * (0.5 - q3q3 - q4q4) +
            two_bz * (q2q4 - q1q3) - mx) + (-two_bx * q1 + two_bz * q3) * (two_bx *
             (q2q3 - q1q4) + two_bz * (q1q2 + q3q4) - my) + two_bx * q2 * (two_bx *
             (q1q3 + q2q4) + two_bz * (0.5 - q2q2 - q3q3) - mz);
67      norm = sqrt(s1 * s1 + s2 * s2 + s3 * s3 + s4 * s4);    % normalise step
            magnitude
68      s1 = s1 / norm;
69      s2 = s2 / norm;
70      s3 = s3 / norm;
71      s4 = s4 / norm;
72
73  %%%%%% It somehow works better without this compensation block
74  %       % compute angular estimated direction of the gyroscope error
75  %       g_err_x = two_q1 * s2 - two_q2 * s1 - two_q3 * s4 + two_q4 * s3;
76  %       g_err_y = two_q1 * s3 + two_q2 * s4 - two_q3 * s1 - two_q4 * s2;
77  %       g_err_z = two_q1 * s4 - two_q2 * s3 + two_q3 * s2 - two_q4 * s1;
78  %
79  %       % compute and remove the gyroscope baises
80  %       g_bx = g_bx + g_err_x * deltat * zeta;
```

```
%     g_by = g_by + g_err_y * deltat * zeta;
%     g_bz = g_bz + g_err_z * deltat * zeta;
%     gx = gx - g_bx;
%     gy = gy - g_by;
%     gz = gz - g_bz;
%%%%%%

    % Compute rate of change of quaternion
    qDot1 = 0.5 * (-q2 * gx - q3 * gy - q4 * gz) - beta * s1;
    qDot2 = 0.5 * (q1 * gx + q3 * gz - q4 * gy) - beta * s2;
    qDot3 = 0.5 * (q1 * gy - q2 * gz + q4 * gx) - beta * s3;
    qDot4 = 0.5 * (q1 * gz + q2 * gy - q3 * gx) - beta * s4;

    % Integrate to yield quaternion
    q1 = q1 + qDot1 * deltat;
    q2 = q2 + qDot2 * deltat;
    q3 = q3 + qDot3 * deltat;
    q4 = q4 + qDot4 * deltat;

    norm = sqrt(q1 * q1 + q2 * q2 + q3 * q3 + q4 * q4);    % normalise
       quaternion
    result(1) = q1 / norm;
    result(2) = q2 / norm;
    result(3) = q3 / norm;
    result(4) = q4 / norm;
```

C.2.3 Madgwick lines and car display code

```
clear;
close all;
clc;
addpath('mytoolbox');

% "doc" to see a function's way of working
% "edit" to see how is written a function

% Port reset:
% instrfind reads serial port objects from memory to MATLAB workspace
if not(isempty(instrfind))
    fclose(instrfind);
    delete(instrfind);
end

%--------------------------------------------------------------------------
%% Serial open
arduino=serial('COM3','BaudRate',115200);
fopen(arduino);
fs = 200;  % 200Hz sample rate (refer to arduino code)
T = 1/fs;  % sample period

y = char.empty;
```

```
while strcmp(y,char([71,79,13,10])) ~= 1
    % while y~='GO', stay in the loop
    y = char(fscanf(arduino));
    disp(y);
end

%--------------------------------------------------------------------------
%% Read data and 2D (animated lines)
% Initialize data
load('mag_calib_values');
gyrList = zeros([0 3]);
accList = zeros([0 3]);

% Initialize fusion
beta = 0.6046;
zeta = 0.005;
result = [1 0 0 0 0 0 0];

% Initialize figure
figure('units','normalized','outerposition',[0.2 0.2 0.7 0.7])
rO = animatedline('Color','r','MaximumNumPoints',1000);
hold on
pI = animatedline('Color','g','MaximumNumPoints',1000);
hold on
yA = animatedline('Color','b','MaximumNumPoints',1000);
grid on
title('Madgwick algorithm HM');
legend('roll','pitch','yaw');
xlabel('time (s)');
ylabel('degree');

% Wait for stabilization
t = 0;
while t<1
    y = fgets(arduino);
    temp = str2num(y);
    t = t + T;
end

% Data acquisition for calibration
t = 0;
t0 = clock;
while t<1.5
    y = fgets(arduino);
    temp = str2num(y);
    size_temp = size(temp);
    if (size_temp >= [3,3])
        acc = temp(1,:);
        gyr = temp(2,:);

        % Build lists before offset calculation
        accList = cat(1,accList,acc);
        gyrList = cat(1,gyrList,gyr);

```

```
        t = clock - t0;
    end
end

% Calibration calculation
accOff = mean(accList);
gyrOff = mean(gyrList);

% Read and treat
t = 0;
update = clock;
while true
    y = fgets(arduino);
    temp = str2num(y);
    size_temp = size(temp);
    if (size_temp >= [3,3])
        acc = (temp(1,:) - [accOff(1),accOff(2),0])./[1,1,accOff(3)];
        gyr = temp(2,:) - gyrOff;
        mag = temp(3,:)-[x_avg,y_avg,z_avg];

        % Process sensor data through algorithm
        for ii = 1:1
            deltat = etime(clock,update);                                   %
               calclate deltat
            update = clock;                                                 %
               initialize next deltat
            result = fusionMadgwick(result, acc, gyr*(pi/180), mag, beta, zeta
               , deltat);    % gyroscope units must be radians
            q0 = result(1:4);  % initialize next quaternion
            euler = quatern2euler(q0)*(180/pi);                             %
               calculate euler angles [yaw, pitch, roll]

            % display
            t = t + deltat;
            addpoints(rO,t,euler(3));
            addpoints(pI,t,euler(2));
            addpoints(yA,t,euler(1));
            disp(euler);
            drawnow limitrate
        end
    end
end

%-------------------------------------------------------------------------
%% Read data and 3D (animated car)
% Initialize data
load('mag_calib_values');
gyrList = zeros([0 3]);
accList = zeros([0 3]);

% Initialize filter
beta = 0.6046;
zeta = 0.005;
result = [1 0 0 0 0 0 0];
```

```
128
129 % Initialize figure
130 figure('units','normalized','outerposition',[0.2 0.2 0.8 0.8])
131 clf
132 xlim([-3 3]);
133 ylim([-3 3]);
134 zlim([-3 3]);
135 view(3);
136 grid on
137 pbaspect([1 1 1]);
138
139 p(1) = patch('YData',[-0.85,-0.85,-0.85,-0.85,-0.85],'XData'
    ,[1.75,1.75,0.25,-1.75,-1.75],'ZData',[0,0.4,1.4,1.4,0], 'FaceColor', 'b','
    FaceAlpha',1);
140 p(2) = patch('YData',[0.85,0.85,0.85,0.85,0.85],'XData'
    ,[1.75,1.75,0.25,-1.75,-1.75],'ZData',[0,0.4,1.4,1.4,0], 'FaceColor', 'b','
    FaceAlpha',1);
141 p(3) = patch('YData',[-0.85,-0.85,0.85,0.85],'XData',[1.75,1.75,1.75,1.75],'
    ZData',[0,0.4,0.4,0], 'FaceColor', 'r','FaceAlpha',1);
142 p(4) = patch('YData',[-0.85,-0.85,0.85,0.85],'XData',[1.75,0.25,0.25,1.75],'
    ZData',[0.4,1.4,1.4,0.4], 'FaceColor', 'k','FaceAlpha',.5);
143 p(5) = patch('YData',[-0.85,-0.85,0.85,0.85],'XData',[0.25,-1.75,-1.75,0.25],'
    ZData',[1.4,1.4,1.4,1.4], 'FaceColor', 'g','FaceAlpha',1);
144 p(6) = patch('YData',[-0.85,-0.85,0.85,0.85],'XData'
    ,[-1.75,-1.75,-1.75,-1.75],'ZData',[0,1.4,1.4,0], 'FaceColor', 'r','
    FaceAlpha',1);
145 p(7) = patch('YData',[-0.85,-0.85,0.85,0.85],'XData'
    ,[-1.75,+1.75,+1.75,-1.75],'ZData',[0,0,0,0], 'FaceColor', 'y','FaceAlpha'
    ,1);
146
147 c = hgtransform;
148 set(p,'Parent',c);
149
150 % Wait for stabilization
151 t = 0;
152 while t<1
153     y = fgets(arduino);
154     temp = str2num(y);
155     t = t + T;
156 end
157
158 % Data acquisition for calibration
159 t = 0;
160 t0 = clock;
161 while t<1.5
162     y = fgets(arduino);
163     temp = str2num(y);
164     size_temp = size(temp);
165     if (size_temp >= [3,3])
166         acc = temp(1,:);
167         gyr = temp(2,:);
168
169         % Build lists before offset calculation
170         accList = cat(1,accList,acc);
```

```
        gyrList = cat(1,gyrList,gyr);

        t = clock - t0;
    end
end

% Calibration calculation
accOff = mean(accList);
gyrOff = mean(gyrList);

% Read and treat
t = 0;
update = clock;
while true
    y = fgets(arduino);
    temp = str2num(y);
    size_temp = size(temp);
    if (size_temp >= [3,3])
        acc = (temp(1,:) - [accOff(1),accOff(2),0])./[1,1,accOff(3)];
        gyr = temp(2,:) - gyrOff;
        mag = temp(3,:)-[x_avg,y_avg,z_avg];

        % Process sensor data through algorithm
        for ii = 1:1
            deltat = etime(clock,update);                                    %
               calclate deltat
            update = clock;                                                  %
               initialize next deltat
            result = fusionMadgwick(result, acc, gyr*(pi/180), mag, beta, zeta
               , deltat);    % gyroscope units must be radians
            q0 = result(1:4);  % initialize next quaternion
            euler = quatern2euler(q0);                                % calculate
               euler angles [yaw, pitch, roll]

            % display
            c.Matrix = makehgtform('xrotate',euler(3),'yrotate',euler(2),'
               zrotate',euler(1));
            %c.Matrix = makehgtform('zrotate',euler(1));
            disp(euler.*(180/pi));
            drawnow limitrate
        end
    end
end
```

C.2.4 Madgwick + GNSS display code

```
clear;
close all;
clc;
addpath('mytoolbox');

```

```
% "doc" to see a function's way of working
% "edit" to see how is written a function

% Port reset:
% instrfind reads serial port objects from memory to MATLAB workspace
if not(isempty(instrfind))
    fclose(instrfind);
    delete(instrfind);
end

%--------------------------------------------------------------------------
%% Serial open
arduino=serial('COM3','BaudRate',115200);
fopen(arduino);
fs = 200;  % 200Hz sample rate (refer to arduino code)
T = 1/fs;  % sample period

y = char.empty;
while strcmp(y,char([71,79,13,10])) ~= 1
    % while y~='GO', stay in the loop
    y = char(fscanf(arduino));
    disp(y);
end

%--------------------------------------------------------------------------
%% Read data and 3D
% Initialize data
load('mag_calib_values');
gyrList = zeros([0 3]);
accList = zeros([0 3]);
mag_dec = -3.92*(pi/180); % use https://www.ngdc.noaa.gov/geomag/calculators/
    magcalc.shtml with [50.8160, 12.9293]
lat_list = zeros([0 1]);
lng_list = zeros([0 1]);
yaw_list = zeros([0 1]);

% Initialize filter
beta = 0.6046;
zeta = 0.005;
result = [1 0 0 0 0 0 0];

% Initialize figure
figure('units','normalized','outerposition',[0.2 0.2 0.8 0.8])
l = animatedline('Color','r','Marker','.');
grid on
pbaspect([1 1 1]);

% Initialize map
name = 'opentopomap';     % Define the name that you will use to specify your
    custom basemap.
url = 'a.tile.opentopomap.org';     % Specify the website that provides the
    map data
copyright = char(uint8(169));     % Create an attribution to display on the
    map that gives credit to the provider of the map data
```

```
attribution = [ ...
      "map data:  " + copyright + "OpenStreetMap contributors,SRTM", ...
      "map style: " + copyright + "OpenTopoMap (CC-BY-SA)"];
displayName = 'Open Topo Map';     % Define the name that will appear
addCustomBasemap(name,url,'Attribution',attribution,'DisplayName',displayName)
        % Add the custom basemap to the list of basemap layers available.

% Initialize data
lat = zeros([0 1]);
lng = zeros([0 1]);

% Wait for stabilization
t = 0;
t0 = clock;
while t<1
    y = fgets(arduino);
    temp = str2num(y);
    t = clock - t0;
end

% Data acquisition for calibration
t = 0;
t0 = clock;
while t<1.5
    y = fgets(arduino);
    temp = str2num(y);
    size_temp = size(temp);
    if (size_temp >= [3,3])
        acc = temp(1,:);
        gyr = temp(2,:);

        % Build lists before offset calculation
        accList = cat(1,accList,acc);
        gyrList = cat(1,gyrList,gyr);

        t = clock - t0;
    end
end

% Calibration calculation
accOff = mean(accList);
gyrOff = mean(gyrList);

% Read and treat
t = 0;
update = clock;
while true
    y = fgets(arduino);
    temp = str2num(y);
    size_temp = size(temp);
    if (size_temp >= [3,3])
        acc = (temp(1,:) - [accOff(1),accOff(2),0])./[1,1,accOff(3)];
        gyr = temp(2,:) - gyrOff;
        mag = temp(3,:)-[x_avg,y_avg,z_avg];
```

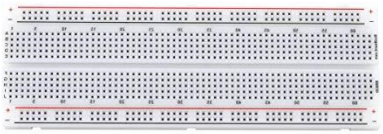
```
109
110         % Process sensor data through algorithm
111         deltat = etime(clock,update);                                   %
                calclate deltat
112         update = clock;                                                 %
                initialize next deltat
113         result = fusionMadgwick(result, acc, gyr*(pi/180), mag, beta, zeta,
                deltat);    % gyroscope units must be radians
114         q0 = result(1:4);  % initialize next quaternion
115         euler = quatern2eulerWiner(q0);                                 % calculate
                euler angles [yaw, pitch, roll]
116         disp(euler.*(180/pi));
117
118         if (size(temp) >= [4,3])
119             lat = temp(4,1);
120             lng = temp(4,2);
121             lat_list = cat(1,lat_list,temp(4,1));
122             lng_list = cat(1,lng_list,temp(4,2));
123             yaw_list = cat(1,yaw_list,euler(1));
124             disp([lat,lng]);
125             addpoints(l,lng(end),lat(end));
126             hold on
127             plot([lng,lng+0.0001*cos(euler(1)+mag_dec)],[lat,lat+0.0001*sin(
                    euler(1)+mag_dec)],'-b');
128             %xlswrite('data',[lat,lng,euler(1)],'append');
129             drawnow limitrate
130         end
131     end
132 end
133
134 save('madgwick_gps_values3','lat_list','lng_list','yaw_list');
135
136 webmap opentopomap      % Open a web map
137 wmline(lat_list,lng_list,'LineWidth',3,'Color','r')     % Plot the glider path
        track on the basemap
```

IMPLEMENTATION PROTOCOL

CONTENTS

HARDWARE AND SOFTWARE

HARDWARE

Component	Quantity	Picture
Arduino Uno	1	
USB cable type A/B	1	
Arduino Nano	1	
USB mini B cable	1	
Breadboard	1	
MPU-9250	1	
Sparkfun SAM-M8Q	1	
nRF24L01 with SMA connector and duck antenna	2	
nRF24L01 adapters	2	

Wires Male/Female	14	
Wires Male/Male	12	

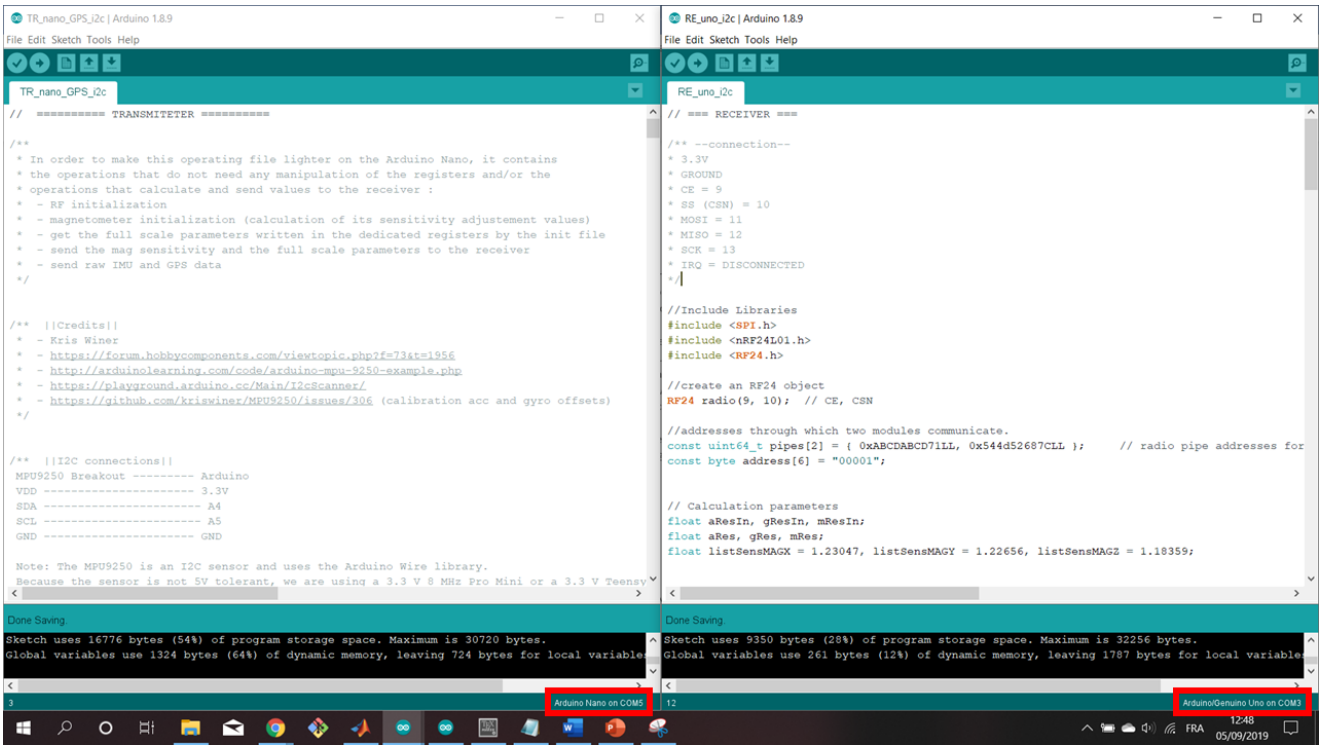
SOFTWARE

Name	Purpose	Additional libraries
Arduino IDE	Collect data	➢ SPI (built-in) ➢ printf (built-in) ➢ SoftwareSerial (built-in) ➢ RF24 (download it from the Libraries Manager) If the I²C interface is used : ➢ Wire (built-in) ➢ SparkFun_Ublox_Arduino_Library (download it from https://github.com/sparkfun/SparkFun_Ublox_Arduino_Library) If the UART interface is used : ➢ SPI (built-in) ➢ RF24 (download it from the Libraries Manager) ➢ NeoGPS (download it from https://github.com/SlashDevin/NeoGPS) ➢ NeoSWSerial (download it from https://github.com/SlashDevin/NeoSWSerial)
MATLAB	Read the serial port and treat data	➢ Mapping Toolbox (download it from the Add-On Explorer, optional library only for GPS coordinates display)
Ublox U-center	Test the GPS (optional)	

GENERAL ADVICE

Respect these few tips to avoid basic errors :

- Check regularly the wiring. A wire can easily disconnect from the breadboard while moving it.
- Choose the good card and the good port in Arduino IDE before you upload an Arduino code.
- Choose the same baud rate in the serial monitor (when you use it) and in MATLAB (when you use it) as the baud rate specified in your Arduino code.
- Make sure the power supply of the sensors is appropriate (3.3V or 5V).
- Make sure you have previously installed the libraries required by the software.
- When you need to open two different windows of Arduino IDE, do it by by double-clicking twice on the program icon (do not open Arduino IDE once and then a second window from the first one). One is for the transmitter (choose Nano and the good port in “Tools”), the other one is for the receiver (choose Uno and the good port in “Tools”).



- After uploading new programs on the Arduino board and before using a MATLAB file that uses the communication between these boards, it is recommended to use first the serial monitor to see if the output is good and if there is any problem on the Arduino's side. If there is an error message in MATLAB, you will know that the problem does not come from the Arduino boards.

Specific advice before using the UART interface with the NeoGPS and NeoSWSerial libraries:

If you want to use the UART interface for the GPS, the library NeoGPS requires a few internal manipulation. The library offers several possibilities for the serial communication using other libraries more efficient then the Arduino built-in SoftwareSerial. Moreover, it allows to change the RX and TX pins rather than the default D0 and D1, which is quite interesting insofar as when a device is connected to D0 and D1 it is not possible to upload a new program on the Arduino without disconnecting each time the RX pin. In the code that has been written, it has been chosen to use the port D4 and D3 as the serial ports with the library NeoSWSerial and with an interruption procedure to avoid lag while the availability of GPS data is being checked.

- In NMEAGPS_cfg located in C:\...\Arduino\libraries\NeoGPS\src, uncomment
  `#define NMEAGPS_INTERRUPT_PROCESSING`
- In GPSport located in C:\...\Arduino\libraries\NeoGPS\src, choose the following communication method

```
#include <NeoSWSerial.h>
NeoSWSerial gpsPort(4, 3);
#define GPS_PORT_NAME "NeoSWSerial(4,3)"
#define DEBUG_PORT Serial
```

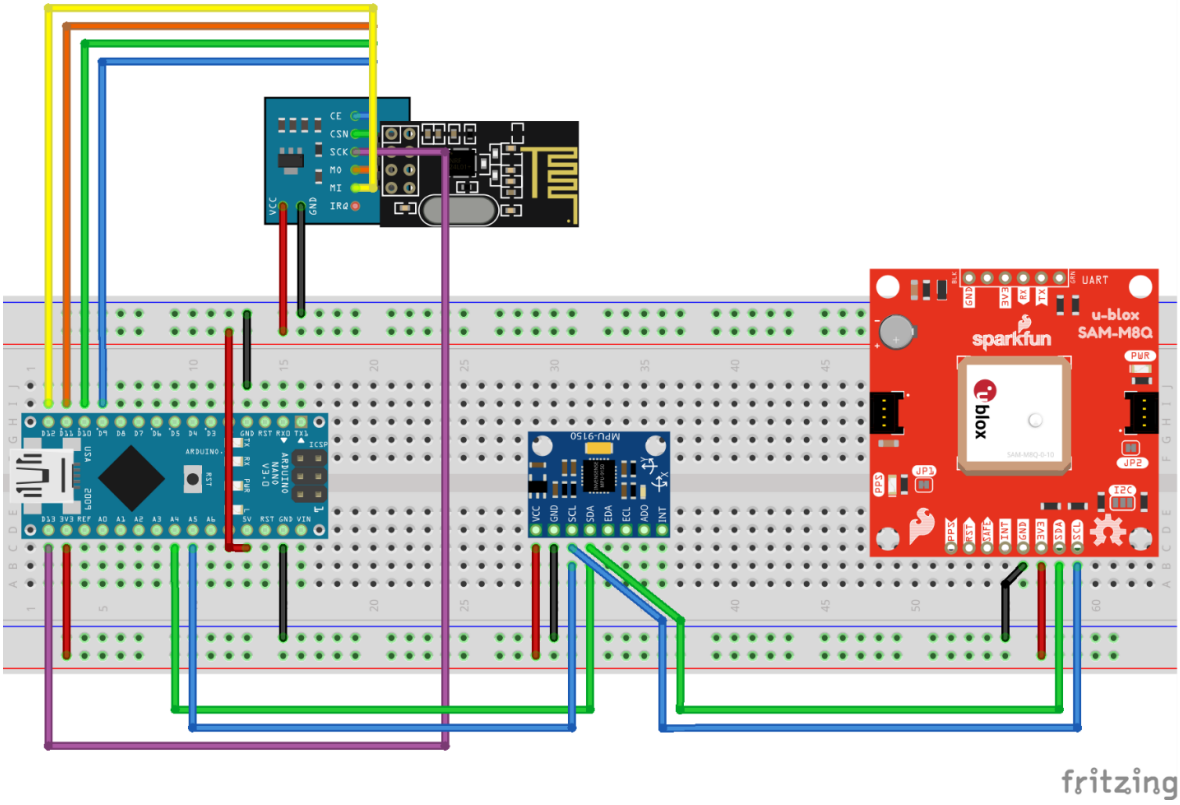
Keyboard shortcuts:

- Arduino IDE : Upload an Arduino file : CRTL + U
- Arduino IDE : Open the serial monitor : SHIFT + CRTL + M
- MATLAB : execute a whole file : F5
- MATLAB : execute a section : CRTL + ENTER
- MATLAB : execute the selected lines : F9

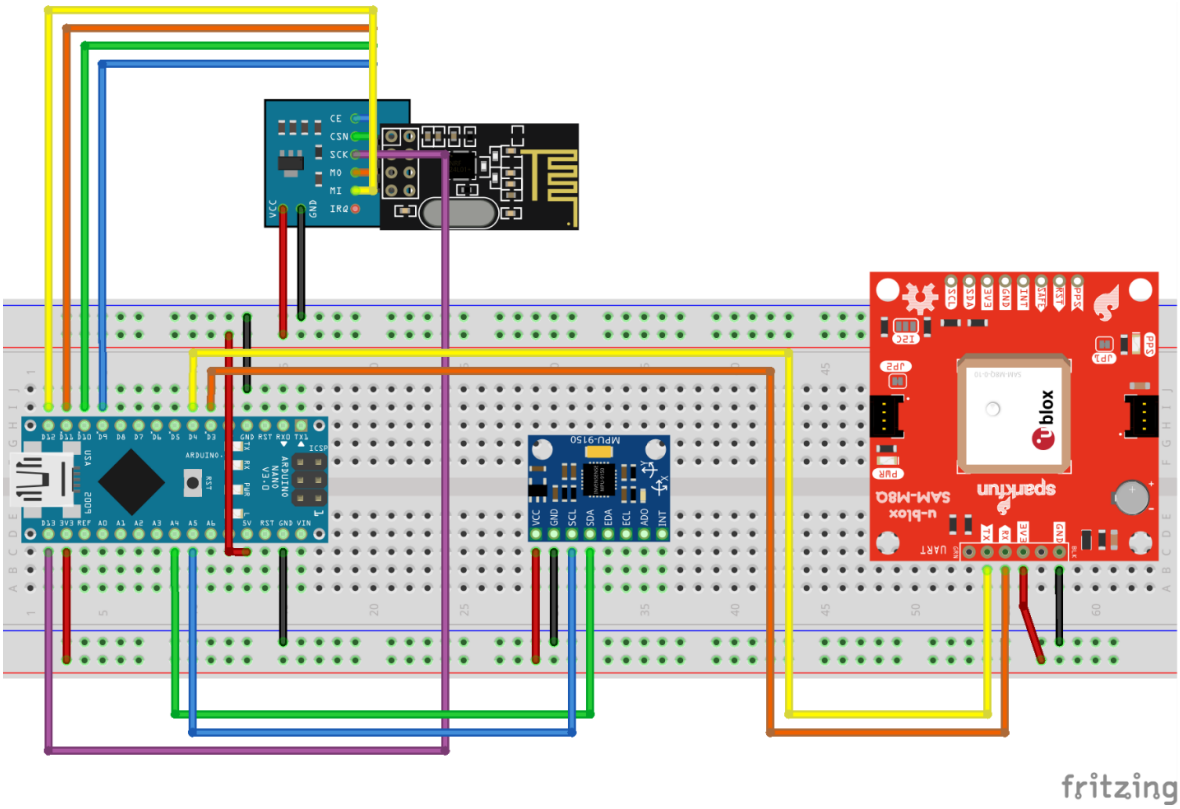
ELECTRICAL ASSEMBLY

TRANSMITTER

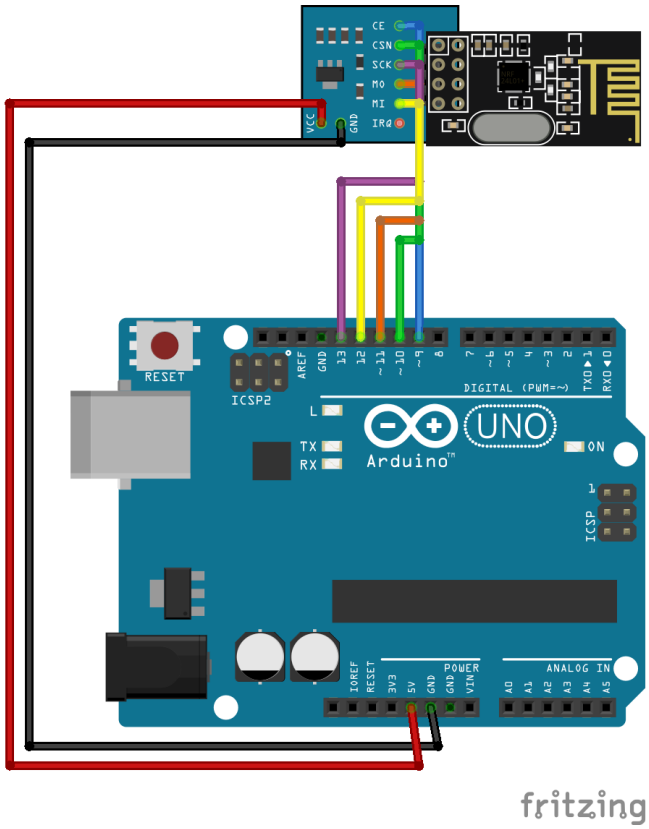
I²C interface



UART interface



RECEIVER


CE
CSN
SCK
MO
MI
IRQ
VCC
GND
RESET
ICSP2
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM=~)
L
TX
RX
UNO
Arduino™
ON
ICSP
IOREF
RESET
3V3
5V
GND
GND
VIN
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
fritzing

TEST PROCEDURES (recommended)

IMU

Detection of the IMU and test of its output

Steps:

- Connect the transmitter to a USB port on your computer.
- Open the file *test_detect_and_read.ino*, choose the good board and port in “Tools”, upload it.
- Make sure the IMU is at rest on a flat surface.
- Open the serial monitor (with the good baud rate).

Output :

You should see the address of the magnetometer 0x0C and the one of the accelerometer/gyroscope 0x48. If you work with the GPS on the I²C bus you should also see the address 0x42. The self-test gives you the percentage of deviation from the factory trim values for the accelerometer and gyroscope and should return only a few percent. The magnetometer sensitivity is calculated by a formula given in the Sparkfun MPU-9250 datasheet. If your IMU is at rest on a flat position, you should obtain values in the order of magnitude of the following values (accX, accY, accZ, gyrX, gyrY, gyrZ, magX, magY, magZ).

```
-> ---Scanning---
-> 0x0C
-> 0x42
-> 0x68
-> 3 devices found on the bus
->
-> ---Self Test---
-> Put the device on a flat surface and do not move it
-> x-axis self test: acceleration response within : -1.1% of factory value
-> y-axis self test: acceleration response within : -4.2% of factory value
-> z-axis self test: acceleration response within : -0.3% of factory value
-> x-axis self test: gyration response within : -0.2% of factory value
-> y-axis self test: gyration response within : -0.3% of factory value
-> z-axis self test: gyration response within : -0.1% of factory value
->
-> ---Accelerometer and Gyroscope Initialization ---
->
-> ---Magnetometer Initialization ---
-> Magnetometer sensitivity adjustement values (µT):
-> X = 1.23
-> Y = 1.23
-> Z = 1.18
-> GO
-> -0.03,-0.01,1.04;1.56,0.96,0.72;105.16,340.23,1.77
-> -0.03,-0.01,1.04;1.54,0.96,0.76;110.70,349.43,3.55
-> -0.03,-0.01,1.04;1.54,0.95,0.75;110.70,349.43,3.55
-> -0.03,-0.01,1.04;1.56,0.93,0.78;110.70,353.11,24.85
-> -0.03,-0.01,1.04;1.56,0.93,0.79;110.70,360.46,14.20
-> -0.03,-0.01,1.04;1.57,0.93,0.79;127.30,343.91,12.42
-> -0.02,-0.01,1.04;1.57,0.93,0.78;127.30,343.91,12.42
-> -0.02,-0.01,1.04;1.60,0.93,0.78;119.92,343.91,15.97
-> -0.02,-0.01,1.04;1.59,0.93,0.76;107.01,353.11,14.20
-> -0.02,-0.02,1.04;1.59,0.93,0.76;107.01,353.11,14.20
```

Possible issues :

Symptom	Cause and solution
Nothing is detected	Check the power supply and the wiring, check if the SCL and SDA pins have not been inverted
The Z-acceleration is around 2g and not 1g	The raw acceleration is multiplied by the wrong resolution. This means an issues occurred in the accelerometer initialization or in the function *getAres* that calculates the resolution.

Detection of the IMU (in case you do not want to execute the whole program presented above)

Steps:

- Connect the transmitter to a USB port on your computer.
- Open the file *test_i2c_scanner.ino*, choose the good board and port in “Tools”, upload it.
- Open the serial monitor (with the good baud rate).

Output :

You should see the address of the magnetometer 0x0C and the one of the accelerometer/gyroscope 0x48. If you work with the GPS on the I²C bus you should also see the address 0x42.

```
-> ---Scanning---
-> 0x0C
-> 0x42
-> 0x68
-> 3 devices found on the bus
```

Possible issues :

Symptom	Cause and solution
Nothing is detected	Check the power supply and the wiring, check if the SCL and SDA pins have not been inverted

GPS

Test of NMEA sentences

Steps:

- Connect the transmitter to a USB port on your computer.
- Open the file *test_raw_GPS_i2c.ino*, choose the good board and port in “Tools”, upload it.
- Open the serial monitor (with the good baud rate).

Output :

Note that a cold start (first start ever of first start after several hours) will make the receiver to take more time to find satellites and obtain geographic coordinates.

1 – first output :

```
-> $GNRMC,,V,,,,,,,,,,N*4D
-> $GNVTG,,,,,,,,,N*2E
-> $GNGGA,,,,,,0,00,99.99,,,,,,*56
-> $GNGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*2E
-> $GNGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*2E
-> $GPGSV,1,1,01,16,,,22*7F
-> $GLGSV,1,1,00*65
-> $GNGLL,,,,,,V,N*7A
```

2 – Information is arriving little by little :

```
-> $GNRMC,092011.00,V,,,,,,,260819,,,N*6C
-> $GNVTG,,,,,,,,,N*2E
-> $GNGGA,092011.00,,,,,0,03,3.22,,,,,,*43
-> $GNGSA,A,1,21,27,20,,,,,,,,,,3.37,3.22,1.00*1F
-> $GNGSA,A,1,,,,,,,,,,,,,3.37,3.22,1.00*1B
-> $GPGSV,2,1,05,13,,,24,15,14,060,13,20,58,114,30,21,49,067,28*43
-> $GPGSV,2,2,05,27,63,298,17*49
-> $GLGSV,1,1,02,66,37,056,36,,,,19*5D
-> $GNGLL,,,,,092011.00,V,N*5F
```

3 – More information received :

```
-> $GNRMC,092105.00,V,,,,,,260819,,,N*68
-> $GNVTG,,,,,,,,,N*2E
-> $GNGGA,092105.00,,,,,0,05,4.83,,,,,*4D
-> $GNGSA,A,1,21,20,10,,,,,,,,,,7.23,4.83,5.37*16
-> $GNGSA,A,1,66,67,,,,,,,,,,,7.23,4.83,5.37*17
-> $GPGSV,2,1,06,10,46,156,34,13,,,19,15,14,060,16,20,59,113,32*4C
-> $GPGSV,2,2,06,21,48,067,32,27,64,298,*74
-> $GLGSV,2,1,06,66,37,055,37,67,33,122,29,68,00,163,,73,37,300,*62
-> $GLGSV,2,2,06,74,14,341,,,,,25*54
-> $GNGLL,,,,,092105.00,V,N*5B
```

4 – Coordinates are finally received. They can be seen in hundredths of degrees in $GNRMC, $GNGGA and in $GNGLL. In $GNGGA are also located the information related to the number of satellites in view, the HDOP (index of trust of the received information), the altitude in meters :

```
-> $GNRMC,092130.00,A,5048.96113,N,01255.78669,E,0.129,,260819,,,A*69
-> $GNVTG,,T,,M,0.129,N,0.239,K,A*3E
-> $GNGGA,092130.00,5048.96113,N,01255.78669,E,1,06,1.92,324.2,M,44.7,M,,*47
-> $GNGSA,A,3,21,27,20,10,,,,,,,,,5.23,1.92,4.86*1D
-> $GNGSA,A,3,66,67,,,,,,,,,,,5.23,1.92,4.86*19
-> $GPGSV,2,1,07,10,46,156,33,13,,,20,15,14,060,14,20,59,113,32*42
-> $GPGSV,2,2,07,21,48,067,30,26,32,186,08,27,64,298,16*42
-> $GLGSV,2,1,06,66,37,055,36,67,34,122,26,68,00,163,,73,37,300,*6B
-> $GLGSV,2,2,06,74,14,341,,,,,28*59
-> $GNGLL,5048.96113,N,01255.78669,E,092130.00,A,A*7E
```

Possible issues :

Symptom	Picture	Cause and solution
The GPS takes too much time to receive data (5 min +)		The chip cannot receive signal from satellites constellations, move closer to a window or go outside.
Strange message with $GNTXT, irregular blinking of the red and orange LEDs on the GPS chip.	$GNTXT,01,01,02,u-blox AG - www.u-blox.com*4E $GNTXT,01,01,02,GNSS OTP=GPS;GLO*37 $GNTXT,01,01,02,LLC=FFFFFFFF-FFFFFFED-FFFFFFFF-FFFFFFFF-FFFFFF49*21 $GNTXT,01,01,02,ANTSUPERV=AC *00	A problem occurred in the power supply of the GPS, check wiring and unplug/plug the wires if necessary. If the problem persists, connect the GPS power supply wires before the ones of the IMU , or even consider an external power supply on the GND and Vin pins of the Nano.
Strange message with $GNTXT	-> $GNTXT,01,01,01,NMEA unknown msg*46 -> $GNTXT,01,01,01,NMEA unknown msg*46 -> $GNTXT,01,01,01,NMEA unknown msg*46 -> $GNTXT,01,01,01,NMEA unknown msg*46 -> $GNTXT,01,01,01,NMEA unknown msg*46	There is a short circuit between the GPS pins RX and TX, check the wiring of the soldering.

Test of the position output at rest

Steps:

- Upload the file *TR_nano_GPS_i2c.ino* on the receiver with Arduino IDE and wait for “Done uploading”.
- Upload the file *RE_uno_i2c.ino* on the transmitter with Arduino and wait for “Done uploading”.
- Execute the file *readAndPlot_GPS.m* in MATLAB.

Output :

The program collects data during 5s, plot it in real-time on a white plane and then plot it in the Web Map Brower. It will be of course less precise and will tend to deviate if you are inside a building.

Possible issues :

Symptom	Picture	Cause and solution
The GPS takes too much time to receive data (5 min +)		The chip cannot receive signal from satellites constellations, move closer to a window or go outside.

RF ANTENNAS

Transmission parameters checking and 2.4GHz scanner test (must be done for the transmitter and receiver)

Steps :

- Upload the file *test_RF_scanner.ino* on a first board with Arduino IDE.
- Wait until you see “Done uploading”, then switch to the other window and upload the file on the other board. If you upload a file on the other board before the first upload is finished, an error will occur.
- Open the serial monitor (with the good baud rate) on both windows.

Output :

First you will see an overview of the antennas’ parameters. These parameters must match between each other to guarantee that the antennas are capable to communicate. Then you will see a scanner of the 125 channels of the 2.4GHz frequency band. The antennas are not synchronized so it is normal that the 2.4Ghz scanner output is different between the two antennas.

```
-> STATUS          = 0x0e RX_DR=0 TX_DS=0 MAX_RT=0 RX_P_NO=7 TX_FULL=0
-> RX_ADDR_P0-1    = 0xe7e7e7e7e7 0xc2c2c2c2c2
-> RX_ADDR_P2-5    = 0xc3 0xc4 0xc5 0xc6
-> TX_ADDR         = 0xe7e7e7e7e7
-> RX_PW_P0-6      = 0x00 0x00 0x00 0x00 0x00 0x00
-> EN_AA           = 0x00
-> EN_RXADDR       = 0x03
-> RF_CH           = 0x4c
-> RF_SETUP        = 0x07
-> CONFIG          = 0x0e
-> DYNPD/FEATURE   = 0x00 0x00
-> Data Rate       = 1MBPS
-> Model           = nRF24L01+
-> CRC Length      = 16 bits
-> PA Power        = PA_MAX
-> 000000000000000011111111111111112222222222222223333333333333334444444444444444555555
-> 0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef012345
-> 000010020000010010010000000000000000000010011000000000000010000100100101100000000010000
-> 000100001101000110100001200200000000000000000000000000000010010001000000000000000200000
-> 010201000000000001000001100100000000000000000000000000000011000100001000000000000001000
-> 100130010200000112113000111000000000000000000000000000000011000010000022000000001010000
```

```
-> STATUS          = 0x0e RX_DR=0 TX_DS=0 MAX_RT=0 RX_P_NO=7 TX_FULL=0
-> RX_ADDR_P0-1    = 0xe7e7e7e7e7 0xc2c2c2c2c2
-> RX_ADDR_P2-5    = 0xc3 0xc4 0xc5 0xc6
-> TX_ADDR         = 0xe7e7e7e7e7
-> RX_PW_P0-6      = 0x00 0x00 0x00 0x00 0x00 0x00
-> EN_AA           = 0x00
-> EN_RXADDR       = 0x03
-> RF_CH           = 0x4c
-> RF_SETUP        = 0x07
-> CONFIG          = 0x0e
-> DYNPD/FEATURE   = 0x00 0x00
-> Data Rate       = 1MBPS
-> Model           = nRF24L01+
-> CRC Length      = 16 bits
-> PA Power        = PA_MAX
-> 000000000000000011111111111111112222222222222223333333333333334444444444444444555555
-> 0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef012345
-> 012000001020000000100000010010000000000011100000000000000000000000000000000011100000
-> 000000001010011010000001010000000000000000000000000000000000000000000000000001000000
-> 000200000001001011110000000000000000000000000000000000000000000000000000000000000000
-> 000000000101210110110000001000000000000100000000000000000000000000000000000011000000
```

Possible issues :

Symptom	Picture	Cause and solution

The parameters of the antennas are different/seem wrong	-> STATUS = 0x00 RX_DR=0 TX_DS=0 MAX_RT=0 RX_P_NO=0 TX_FULL=0 -> RX_ADDR_P0-1 = 0x0000000000 0x0000000000 -> RX_ADDR_P2-5 = 0x00 0x00 0x00 0x00 -> TX_ADDR = 0x0000000000 -> RX_PW_P0-6 = 0x00 0x00 0x00 0x00 0x00 0x00 -> EN_AA = 0x00 -> EN_RXADDR = 0x00 -> RF_CH = 0x00 -> RF_SETUP = 0x00 -> CONFIG = 0x00 -> DYNPD/FEATURE = 0x00 0x00 -> Data Rate = 1MBPS -> Model = nRF24L01 -> CRC Length = Disabled -> PA Power = PA_MIN	Check the RF communication initialization in the “setup” of the Arduino file, check the channels that are used. Check wiring (the example opposite comes from a wiring mistake).

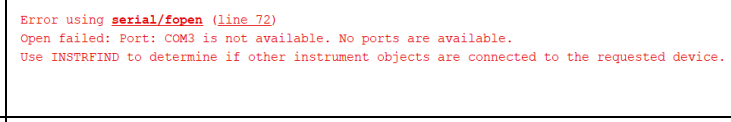
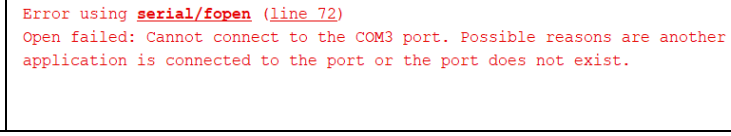
MATLAB PROGRAMS EXECUTION

MAGNETOMETER CALIBRATION

Steps :

- Upload the file *TR_nano_GPS_i2c.ino* on the receiver with Arduino IDE and wait for “Done uploading”.
- Upload the file *RE_uno_i2c.ino* on the transmitter with Arduino and wait for “Done uploading”.
- Execute the file *magCalibration.m* in MATLAB.
- Move the transmitter and try to draw a sphere while moving it as it is presented in the following videos :
  - magnetometer calibration (screen view)
  - magnetometer calibration (webcam view)
- When the calibration is done, a test window will open so that you can see how effective the calibration has been.
- If you want to begin again the calibration after you stopped the first one, make sure you pushed the reset button on the Nano before.

Possible issues :

Symptom	Picture	Cause and solution
Nothing is happening		If you have restarted the MATLAB code, you must push the reset button on the Nano. It can also come from the transmitter Arduino program that is stuck in an infinite loop if it does not detect the GPS. In that case, check the wiring and power supply of the GPS, or test the NMEA sentences.
The MATLAB program does not detect any Arduino board.		MATLAB tries to create a link to the Arduino board through the serial port. Check the port specified in the MATLAB code.
Data flow of the magnetometer is not regular or has strange peak values.		Check wiring (wires tend to disconnect easily) and/or power supply.
Serial port not available		The serial monitor in Arduino IDE or the software U-Center are probably connected to the serial port of the Uno.

COMPLEMENTARY FILTER (REQUIRES A CALIBRATED MAGNETOMETER)

Steps :

- Upload the file *TR_nano_GPS_i2c.ino* on the receiver with Arduino and wait for “Done uploading”.
- Upload the file *RE_uno_i2c.ino* on the transmitter with Arduino and wait for “Done uploading”.
- Open the file *readAndTreat_Euler_compFilter.m* in MATLAB and uncomment the section you want. You can choose the animated lines, the animated lines filtered or the animated car. In all cases, the Euler angles values will be displayed in the command window. Execute the program.

- Wait until the animation figure opens. When it is open, wait a few seconds until the values are stabilized. See two videos of the two possible outputs below :
  - animated lines
  - animated car
- If you want to begin again the animation after you stopped the first one, make sure you pushed the reset button on the Nano before.

Possible issues : see the common errors.

MADGWICK ALGORITHM (REQUIRES A CALIBRATED MAGNETOMETER)

Steps :

- Upload the file *TR_nano_GPS_i2c.ino* on the receiver with Arduino and wait for "Done uploading".
- Upload the file *RE_uno_i2c.ino* on the transmitter with Arduino and wait for "Done uploading".
- Open the file *readAndTreat_Euler_Madgwick.m* in MATLAB and uncomment the section you want. You can choose the animated lines display or the animated car. In both case, the Euler angles values will be displayed in the command window. Execute the program.
- Wait until the animation figure opens. When it is open, wait a few seconds until the values are stabilized. See two videos of the two possible outputs below :
  - animated lines
  - animated car
- If you want to begin again the animation after you stopped the first one, make sure you pushed the reset button on the Nano before.

Possible issues : see the common errors.

MADGWICK + GPS (REQUIRES A CALIBRATED MAGNETOMETER)

Steps :

- Upload the file *TR_nano_GPS_i2c.ino* on the receiver with Arduino and wait for "Done uploading".
- Upload the file *RE_uno_i2c.ino* on the transmitter with Arduino and wait for "Done uploading".
- Open the file *readAndTreat_Euler_Madgwick.m* in MATLAB and uncomment the section you want. You can choose the animated lines display or the animated car. In both case, the Euler angles values will be displayed in the command window. Execute the program.
- Wait until the animation figure opens. When it is open, wait a few seconds until the values are stabilized (stabilization of the Madgwick algorithm). The display window includes the geographic position (in red) and the current orientation from this position (blue). See two videos of the two possible outputs below :
  - test at the office
  - test in better conditions (better GPS signal, less magnetic disturbance)
- If you want to begin again the animation after you stopped the first one, make sure you pushed the reset button on the Nano before

Possible issues : see the common errors.

KALMAN FILTER

to be filled later

COMMON ERRORS

Symptom	Picture	Cause and solution
Nothing is happening	Warning: Unsuccessful read: A timeout occurred before the Terminator was reached. 'serial' unable to read any data. For more information on possible reasons, see Serial Read Warnings.	If you have restarted the MATLAB code, you must push the reset button on the Nano. It can also come from the transmitter Arduino program that is stuck in an infinite loop if it does not detect the GPS (in that case, check the wiring and power supply of the GPS).
The MATLAB program does not detect any Arduino board.	Error using **serial/fopen** (line 72) Open failed: Port: COM3 is not available. No ports are available. Use INSTRFIND to determine if other instrument objects are connected to the requested device.	MATLAB tries to create a link to the Arduino board through the serial port. Check the port specified in the MATLAB code.
Serial port not available	Error using **serial/fopen** (line 72) Open failed: Cannot connect to the COM3 port. Possible reasons are another application is connected to the port or the port does not exist.	The serial monitor in Arduino IDE or the software U-Center are probably connected to the serial port of the Uno.

RECOMMENDED DOCUMENTATION

Hardware

Arduino

- Arduino boards comparison : https://www.arduino.cc/en/products.compare

nRF24L01

- Product specification : https://lastminuteengineers.com/datasheets/nrf24L01+2.4ghz-transceiver-datasheet.pdf

MPU-9250

- Product specification : https://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf
- Register map : https://cdn.sparkfun.com/assets/learn_tutorials/5/5/0/MPU-9250-Register-Map.pdf

u-blox M8 (GPS chip receiver)

- Receiver description : https://cdn.sparkfun.com/assets/0/b/0/f/7/u-blox8-M8_ReceiverDescrProtSpec__UBX-13003221__Public.pdf

Software

Arduino

- Wire library : https://www.arduino.cc/en/reference/wire
- RF24 methods : https://tmrh20.github.io/RF24/classRF24.html#a4cd4c198a47704db20b6b5cf0731cd58
- NeoGPS methods : https://github.com/SlashDevin/NeoGPS/blob/master/extras/doc/Data%20Model.md

MATLAB

- Object animation : https://fr.mathworks.com/videos/animating-an-objects-trajectory-in-matlab-with-hgtransform-97224.html
- Kalman filtering : https://fr.mathworks.com/help/control/ug/kalman-filtering.html
- Understanding Kalman filters : https://fr.mathworks.com/videos/understanding-kalman-filters-part-1-why-use-kalman-filters--1485813028675.html

Code folder on Github

- https://github.com/alexpiot91/Tracking-Project-Chemnitz

ENSTA Bretagne

RAPPORT D'EVALUATION
ASSESSMENT REPORT

Merci de retourner ce rapport par courrier ou par voie électronique en fin du stage à :
At the end of the internship, please return this report via mail or email to:

ENSTA Bretagne – Bureau des stages - 2 rue François Verny - 29806 BREST cedex 9 – FRANCE
☎ 00.33 (0) 2.98.34.87.70 / stages@ensta-bretagne.fr

I - ORGANISME / *HOST ORGANISATION*

NOM / *Name* TU-CHEMNITZ

Adresse / *Address* REICHENHAINER STR. 70

Tél / *Phone (including country and area code)* +49(0)371 531 39362

Nom du superviseur / *Name of internship supervisor*
Fonction / *Function* VLADIMIR BUDAY R&D

Adresse e-mail / *E-mail address* VLADIMIR.BUDAY@MB.TU-CHEMNITZ.DE

Nom du stagiaire accueilli / *Name of intern* ALEXANDRE PIOT

II - EVALUATION / *ASSESSMENT*

Veuillez attribuer une note, en encerclant la lettre appropriée, pour chacune des caractéristiques suivantes. Cette note devra se situer entre **A (très bien)** et **F (très faible)**
Please attribute a mark from ***A (excellent)*** *to* ***F (very weak).***

MISSION / *TASK*

- La mission de départ a-t-elle été remplie ? A (B) C D E F
  Was the initial contract carried out to your satisfaction?
- Manquait-il au stagiaire des connaissances ? ☐ oui/*yes* ☒ non/*no*
  Was the intern lacking skills?

  Si oui, lesquelles ? / *If so, which skills?* _____

ESPRIT D'EQUIPE / *TEAM SPIRIT*

- Le stagiaire s'est-il bien intégré dans l'organisme d'accueil (disponible, sérieux, s'est adapté au travail en groupe) / *Did the intern easily integrate the host organisation? (flexible, conscientious, adapted to team work)* (A) B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* _____

COMPORTEMENT AU TRAVAIL / *BEHAVIOUR TOWARDS WORK*

Le comportement du stagiaire était-il conforme à vos attentes (Ponctuel, ordonné, respectueux, soucieux de participer et d'acquérir de nouvelles connaissances) ?
Did the intern live up to expectations? (Punctual, methodical, responsive to management instructions, attentive to quality, concerned with acquiring new skills)?

(A) B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* _____
_____

INITIATIVE – AUTONOMIE / *INITIATIVE – AUTONOMY*

Le stagiaire s'est –il rapidement adapté à de nouvelles situations ? A B C D E F
(Proposition de solutions aux problèmes rencontrés, autonomie dans le travail, etc.)

Did the intern adapt well to new situations? (A) B C D E F
(eg. suggested solutions to problems encountered, demonstrated autonomy in his/her job, etc.)

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* _____
_____

CULTUREL – COMMUNICATION / *CULTURAL – COMMUNICATION*

Le stagiaire était-il ouvert, d'une manière générale, à la communication ? (A) B C D E F
Was the intern open to listening and expressing himself /herself?

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* _____
_____

OPINION GLOBALE / *OVERALL ASSESSMENT*

❖ La valeur technique du stagiaire était : A (B) C D E F
Please evaluate the technical skills of the intern:

III - PARTENARIAT FUTUR / *FUTURE PARTNERSHIP*

❖ Etes-vous prêt à accueillir un autre stagiaire l'an prochain ?

Would you be willing to host another intern next year? ☒ oui/*yes* ☐ non/*no*

Fait à CHEMNITZ, le 29.04.2019
In _____, on _____

Technische Universität Chemnitz
Fakultät Maschinenbau
Professur Fahrzeugsystemdesign
D-09107 Chemnitz

Signature Entreprise _____ Signature stagiaire
Company stamp _____ *Intern's signature*

Technische Universität Chemnitz
Fakultät Maschinenbau
Professur Alternative Fahrzeugantriebe
D-09107 Chemnitz

Merci pour votre coopération
We thank you very much for your cooperation