UNIVERSITÀ DEGLI STUDI DI CATANIA
FACOLTÀ DI INGEGNERIA ELETTRICA ELETTRONICA E INFORMATICA

XXVII cycle Ph.D. in System Engineering

Antonino Catena

Control Architectures for Heterogeneous Fleets of Unmanned Vehicle Systems

Ph.D. Thesis

Tutor: Prof. G. Muscato
Coordinator: Prof. L. Fortuna

2014

Preface

The field of aerial robotics, from 20 years ago up to now, has had an incredible growth. The reasons are several, but surely, the key motivation is the development of MEMS sensors and microcontroller more and more cheap and reliable. The "Dipartimento di Ingegneria Elettrica, Elettronica e Informatica" (DIEEI) at the University of Catania is involved in several research projects focused on the study and development of Unmanned Aerial Systems (UASs).

The most well-known project of DIEEI in the field of Unmanned Aerial Vehicles (UAVs) is the Volcan Project, an autonomous aerial platform for volcano activities monitoring in order to analyze gases and to improve the forecast of the lava flow during an eruption.

In addition to the Volcan project, many research activities of DIEEI are focused on problematic related to UAVs that nowadays are still open, such as cooperation with different type of robotic platforms, inertial navigation, visual navigation and last but not least, the power management in order to maximize the autonomy. This Ph.D. course was related to and funded by the Ambition Power project [64], whose objective was the virtual prototyping of power devices in avionics field.

Acknowledgements

First of all, I would like deeply thank my family: had it not been for them, nothing would have been possible.

My very special thanks go to my Tutor Prof. Eng. Giovanni Muscato, who believed in me, gave me the possibility to demonstrate my capabilities, supported me with his scientific knowledge, and to the Ph.D. Coordinator Prof. Eng. Luigi Fortuna, because he worked as a real team manager, defending and motivating our group.

I would also like to thank the members of the Service Robots Group for their being helpful, patient and sapient: Eng. Donato Melita, Eng. Luciano Vito Cantelli, and Prof. Domenico Longo.

Last but not least, a particular thanks to my Ph.D. colleagues: Viviana, Davide e Marco, because they have contributed to make these three years unforgettable.

Nomenclature

ACI: AscTec Communication Interface
ACK: Acknowledgement
ADAHRS: Air Data and Attitude Heading Reference System
ADC: Analog to Digital Converter
AGATE: Advanced General Aviation Transport Experiment
AoA: Angle of Attack
ASL: Above Sea Level
AUV: Unmanned Underwater Vehicle
CAN: Controller Area Network
CSMA/CD: Carrier Sense Multiple Access with Collision Detection
DIEEI: Dipartimento di Ingegneria, Elettrica, Elettronica ed Informatica
DLL: Dynamic Link Library
DoF: Degree of Freedom
DSD: Debug Service Data
EAP: Electro-Active Polimer
EKF: Extended Kalman Filter
ENAC: Ente Nazionale Aviazione Civile
EEPROM: Electrically Erasable Programmable Read-Only Memory
FCCS: Flight Control Computer System
FW: FirmWare
GPS: Global Positioning System
GUI: Graphical User Interface
HIL: Hardware In the Loop
HL: High Level (processor)
HMI: Human Machine Interface
I²C: Inter Integrated Circuit
ICAO: International Civil Aviation Organization
IDE: Integrated development environment
IMU: Inertial Measurement Unit
LiPo: Lithium Polymer
LL: Low Level (processor)
LTA: Light then Air

MEMS: Micro Electro Mechanical Systems
NASA: National Aeronautics and Space Administration
NOD: Normal Operation Data
NSH: High-priority Node Service Data
ODR: Output Data Rate
OS: Operating System
PCB: Printed Circuit Board
PIC: Pilot in Command
ROR: Route Of Robot
RPA: Remotely piloted aircraft
RPV: Remotely Piloted Vehicle
RPY: Roll Pitch Yaw
RS-232: Recommended Standard 232
RTK: Real Time Kinematic
S&R: Search and Rescue
SACS: Servo Actuators Control System
SDK: software development kit
SLAM: Simultaneous Localization And Mapping
SNR: Signal Noise Ratio
SoA: State of Art
SPI: Serial Peripheral Interface
SWD: Serial Wire Debug
UAV: Unmanned Aerial Vehicle
UDP: User Datagram Protocol
UGV: Unmanned Ground Vehicle
USART: Universal Synchronous-Asynchronous Receiver/Transmitter
USB: Universal Serial Bus
UVS: Unmanned Vehicle System
VI: Virtual Instrument
VTOL: Vertical TakeOff and Landing
WP: WayPoint
WPL: WayPoint List

Contents

Chapter I. Introduction

I.1 The State of the Art

The adoption of UVSs (Unmanned Vehicle System) as performing tools to be used for data gathering, S&R operations, civil protection and safety issues is rapidly increasing. Generally, the unmanned vehicles are grouped in three categories:

- UAV, Unmanned Aerial Vehicles
- UGV, Unmanned Ground Vehicles
- AUV, Autonomous Underwater Vehicles

Figure 1 - Examples of UVSs

The topic of this thesis is strictly related to the first class of robots, which are the robotic platforms able to fly without pilot.

I.1.1 Normative: UAV, RPA or RPV?

The increasingly frequent use of drones in recent years [2] [3] has made it essential a legislation concerning them in order to regulate its use. There is a general legislation drafted by ICAO [15], the International Civil Aviation Organization, whereas as regards the UAVs with a weight less than 150 Kg there is a particular normative depending by the nation. In the case of Italy, this

normative is drafted by the ENAC [14]. The key aspect of the normative is that the use of a completely autonomous robotic platform in open field is illegal, except for very particular cases that, however, require permission by the relevant authorities. In fact, according to normative, there must be someone that pilots, or in general supervises, the aircraft. For this reason, the relevant authorities prefer using the terms such as Remotely Piloted Vehicle (RPV) or Remotely Piloted Aircraft (RPA) in place of UAV, in order to underline the role, and the responsibilities, of the operator that controls the plane, which, according to the normative, is to all effects a pilot. Moreover, still according to the law, anyone who pilots a drone must have a license, issued after an examination, and an insurance that covers the risks connected with the activity of the drone.

I.1.2 Classification of UAVs

To make a complete summary of all the models of UAV is a really difficult operation, because of the huge variety of applications in which they are used. On the other hand, it's possible to make a comparison between them, on the basis of particular aspects, such as dimension, airframe and scope [1].

I.1.2.1 UAV Dimension

A first comparison between UAV takes into account their dimension, and then their mass (Figure 2). As it is shown in the following table, the mass of a drone is strictly connected to its autonomy and operational range.

Category	Acronym	Mass[Kg]	Max Op. Range[Km]	Max Flight Altitude[m]	Max Duration of Flight[h]
Nano	η	<0.0250	1	100	0.5
Micro	μ	<5	10	250	1
Mini	Mini	<30	10	300	2
Close Range	CR	<150	30	3000	4
Short Range	SR	<200	70	3000	6
Medium Range	MR	<1250	200	5000	10

Table 1 - UAV classification by dimension

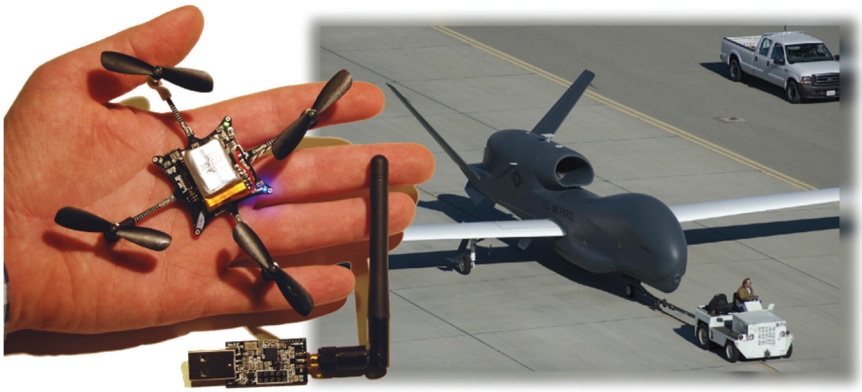

Figure 2 - Different dimensions of UAVs

I.1.2.2 UAV Airframes

Essentially, the airframe of an UAV may be of five different types. It is worth to point out that there isn't "the best one" airframe for every application, but each airframe has advantages and disadvantages, which must be evaluated, in order to choose the best drone to accomplish a given task.

Fixed wings

This is the most common airframe for big drones, especially for the maturity of technology (Figure 3). This type of airframe gives great advantages in terms of efficiency and power consumption, due to the additional lift provided from the wings. However, the wings require a minimum speed cruising to perform their tasks. This aspect brings to the main limit of this type of airframe: the incapability to hover or fly slower, in addition to the fact that they require a runway for takeoff and landing operations. For these reasons they are unsuitable for indoor applications.

Figure 3 - Fixed wings airframe

Rotary wings

This class of airframe gets the necessary lift to fly directly by the propellers. Moreover, they are able to hover and to execute vertical takeoff and landing (VTOL). This makes them perfect for indoor flight. Depending on propellers number, it is possible to split this class in two groups:

- Helicopters: this type of airframe has one or two rotors (Figure 4). In relation to this class of aircraft, this structure ensures the best performance in terms of energy consumption. In fact, the rotor has a practically constant speed and the aircraft movements are given by the variation of the angle of attack (AoA) of the blades. In a few words, the speed of a helicopter is not related to the speed of rotation of the rotor, but essentially to the AoA of the blades. Anyhow, such a rotor is a very complex system (Figure 5) and, even if its technology is consolidated, this has an impact on the cost and reliability.

Figure 4 - Helicopter airframe

Figure 5 - Helicopter rotor

- Multirotors: to this group belongs airframes that have three or more (generally, up to eight) rotors (Figure 6). With respect to helicopters, in these drones the propellers have a fixed AoA, but variable speed. This means a great simplicity in the mechanical structure, but a worst power management. It is precisely the mechanical simplicity that has made the multirotors the most common airframes for small drones, especially in the field of academic research.

Figure 6 - Multirotors airframe

Tilt rotors

This class of airframes is a hybrid between the two previously discussed configurations (Figure 7). Such a structure is capable, at the same time, to execute vertical takeoff and landing, to stay in hovering and to reach a cruise speed comparable with a fixed wings system. However, it is important to underline that these features are achieved by means the rotation of the rotors, which causes reliability problems and a very complex control during the transition phase. In

addition, another disadvantage of this airframe resides in the propellers: a propeller designed for the hovering is not optimized for flying, and viceversa. In other words, a tiltrotor is capable to hover, but it is worst with respect to a rotary wing. Moreover it is capable to fly forward for a long range, but consuming more energy respect to a fixed wing.

Figure 7 - Tiltrotor airframe

Flapping wings

This bio-inspired UAV airframe is the most recent. Its main limit, in fact, derives from the fact that the technology behind it is not mature yet. There are still open issues: ignoring for now problems related to control, probably the most interesting challenge regards the development of linear actuators, capable to reproduce the muscle motion. A possible solution could be the electro-active polymers (EAP), but this is not yet a mature technology and some years are still needed to obtain a commercial product. Depending on the animal to which they are inspired, there are two classes of flapping wings airframes:

- Ornithopters: bird-like airframes, which generate lift by flapping wings up and down with synchronized small variations of AoA (Figure 8). As in the fixed wings airframes, this type of flapping wings requires a forward flight to generate lift.
- Entomopters: inspired to the insect structure, this airframe generates a great variation of AoA between the upstroke and the downstroke phases (Figure 9). Unlike the previous one, this airframe is capable to hover and to execute vertical takeoff and landing.

Figure 8 - Flapping wings, bird-like airframe

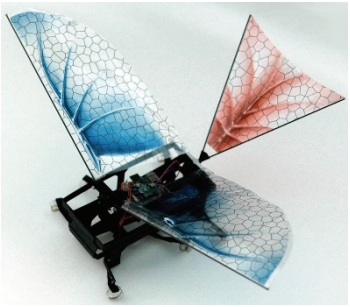

Figure 9 - Flapping wings, insect-like airframe

Blimp

To this last class of airframe belong drones called Lighter Than Air, or LTA (Figure 10). It is easy to guess how this airframe has the best efficiency in terms of energy, since no energy is needed to hover. However, generally these airframes move slower than the other types of airframes, have a bigger volume, in order to obtain enough lift force and, above all, have a limited payload. The latter peculiarity represents the biggest disadvantage of this type of airframe, because a typical mission with a drone often requires additional equipment such as cameras, sensors, robotic arms and so on.

Figure 10 - LTA airframe

To summarize, in Table 2 the main features of each airframe are compared.

	Fixed Wings	Rotary Wings	Tilt Rotors	Flapping Wings	Blimp
Power efficiency	Medium	Bad	Bad	Medium	Good
Control	Good	Good	Medium	Bad	Good
Miniaturization	Medium	Good	Bad	Medium	Bad
Payload	Good	Medium	Good	Bad	Bad
Hover	Bad	Good	Medium	Medium	Good
Low Speed Fly	Bad	Good	Medium	Medium	Good
High Speed Fly	Good	Bad	Medium	Bad	Bad
Robustness	Good	Medium	Bad	Bad	Bad
Maneuverability	Medium	Good	Medium	Bad	Good
Indoor usage	Bad	Good	Bad	Bad	Medium
Outdoor usage	Good	Medium	Medium	Medium	Good

Table 2 - UAV airframes comparison

I.1.2.3 Scope

In the last years the number and the diversity of applications regarding the use of UAVs is increased enormously. A first distinction is usually made between military and civil applications. Focusing on the second group, the non-military applications where the UAVs are commonly used are:

- Disaster management [4] [5].
- Agricultural monitoring and management [6].
- Infrastructure inspection [7] [8].
- Law enforcement [9].
- Weather monitoring.
- Environmental monitoring and exploration [10] [11] [12].
- Aerial imaging/mapping.

- Entertainment: television news coverage, sporting events, moviemaking.
- Freight transport.
- Oil and gas exploration [13].

As regards the DIEEI [52], the research activities are focused mainly on monitoring and forecasting of volcanic activity. The volcano under examination is the Mount Etna, one of the most active volcanoes in the world, which is in an almost constant state of eruption. In the next chapter the developed UAV, i.e. the Volcan, will be treated.

I.2 Architecture of an UAV

The design of an UAV control system is a very complex mission and, as it often happens in engineering topics, there is no a single way to accomplish this task. Listing all the possible architectures goes beyond the scope of the thesis, therefore in the next chapters we will focus only on the control architectures of the UAVs used. However, some milestones are always present in the development of an UAV control system.

The choice of the model

This is the first and probably the most important step. A good modeling represents the key phase in order to obtain satisfactory dynamic performances. The expression "good modeling" is not intended as a perfect modeling, where every dynamic effect is considered, but as a modeling where only the most important dynamic modes are taken into account.

The control strategies

Once the model has been developed, it is necessary to ensure the system stability and, as far as possible, the immunity to noise and to unmodeled dynamics. The most used control strategies use PID [17] controllers or digital filters such as EKF [40] [41] and complementary filters [42] [43]. Once the stability is obtained, the next step is focusing on high level control tasks such as collision avoidance, cooperation, fault detection and so on.

Sensors

Sensors are fundamental in order to obtain information about the state of the drone. As regards the stability, an Inertial Measurement Unit (IMU) is needed. This system returns roll and pitch resulting by a sensors fusion of a three axial accelerometer and of a three axial gyroscope. Often a three axial magnetometer is added in order to obtain also the yaw angle. For navigation, generally GPS and pressure sensors are used. Finally, for high level tasks as obstacle avoidance or object tracking, cameras, laser scanners and ad hoc sensors are used. For the choice of the sensors, in addition to the precision, the other parameters to take into account are the bandwidth, the power consumption, the immunity to noise and last but not least, the easy of interfacing to a microprocessor.

Motors and actuators

Motors and actuators connected to the mobile parts of the drones are necessary to transduce the commands coming from control unit. As for the sensors, precision and bandwidth are important parameters to consider in their choice. Moreover, it is extremely important to underline that this is the part of the whole architecture that consumes more energy. So, if in one hand more power means more torque, on the other hand it means more weight and less autonomy.

CPUs

The core of the control architecture computes data coming from sensors and in according to the task, sends commands to actuators. If to execute a stability control by means a set of PID controllers is adequate a commercial microprocessor that costs a few Euros, to accomplish complex tasks in real time such as SLAM or recognize a target by means of an HD camera it is necessary a dedicated PC with a real time OS. Also in this case it is mandatory the monitoring of the energy consumption, since the computational load of the control algorithm is strictly related to the energy necessary to execute it.

Communication Protocols

As discussed in the section I.1.1, for normative reason a drone always has to be connected to a remote station, where an operator can monitor and supervise its mission. Moreover, the control systems are becoming more and more complex and often they are realized as a combination of subsystems connected each other. For this reason the choice of the communication protocol is dual: to communicate to remote station and to interconnect the various subsystems of the control architecture. As regards the former, generally this link is used also to send the drone telemetry. In most cases this connection is made by a WiFi link. As concern the latter, is mandatory to choose a communication protocol that guarantees a data rate at least an order of magnitude greater than the bandwidth of the sensors and actuators used and also an SNR as small is possible. A differential communication with a good bandwidth, like the CANbus, is suitable for this purpose. However, generally also a normal serial communication could be suitable.

I.3 Limits and Challenges

From now on, this thesis will be focused exclusively on fixed wings and multirotors, i.e. those airframes mainly used for both civilian and research activities. As mentioned in the previous pages, the progress in this field has been enormous, especially in the last 5 years. However, there are still some aspects where some improvement and clarifications are needed. Probably, the key limitation resides on the current normative. If on the one hand, the scientific community tries to develop a completely autonomous UAV [2], on the other hand actual rules generally require a human supervisor responsible for the actions of the robot. Surely, this is a difficult aspect to solve and even if a normative exist, these are destined to be modified in accordance with the technological growth. As regards the technical aspect, the most evident bottleneck is the energy management. Generally, even if the energy density of the batteries has steadily increased during the last years, the autonomy of a commercial UAV with brushless motors is less than 30 minutes. This is a limit extremely incapacitating, when you consider a complex task to accomplish, such as mapping an area or search a target, which generally requires a lot of time. Moreover, a complex task requires a high computational load, which obviously consumes a lot of energy, further reducing the autonomy. At present, the most common batteries used are of LiPo type. A good alternative could be fuel cells, i.e. a sort of battery in which the fuel is transformed into electric current through an electrochemical process. However also fuel cells have an energy density lower than other sources, such as gasoline or methanol [1]. Increasing the autonomy of an UAV is without doubt the main challenge that the scientific community have to face in the next years. Concerning the control algorithms, the results so far are more than satisfactory, even if the best performance are generally obtained indoor, thanks to the feedback provided by motion detection systems, as the Vicon [59] . The ETH of Zurich [62] and the CATEC in Seville [63] are among the best European research centers in this field. However, to get a level of control comparable in outdoors conditions is a hard challenge. The main reasons are two:

- First of all, motion tracking systems are unsuitable for outdoor applications, in particular in the case of unstructured environments. Normally MEMS sensors are used, such as accelerometers, gyroscopes and magnetometers to help localization. These sensors are less precise and

noisier respect to a motion tracking system, and then the feedback of the control architecture is less reliable.

- Secondly, an unstructured environment introduces dynamics that degrade the accuracy of the mathematical model of the aircraft.

I.4 Development tools

The testing phase during the design of the control architecture of an UAV is the longest phase in terms of time. This is because, despite other robotics platform such as UGVs, a bug in the control algorithm most of the times means the destruction of the drone itself. A powerful method to test the control algorithm is the Hardware In the Loop (HIL) architecture [32] [33], where the real aircraft is substituted with a virtual one, generally within a flight simulator. In this way it is possible to test and tune the control architecture, under the assumption that the aircraft model of the flight simulator reflects satisfactorily the dynamic behavior of the real one. However, in HIL architecture it is not possible to test other fundamentals parts of the whole system, such as sensors and actuators. A step ahead in this direction is represented by the Motion Capture technique. In this scenario, the drone operates inside an arena, where a set of high speed cameras provide an extremely precisely feedback regarding pose and position. In this way, in addition to the control algorithm, it is possible to test the actuators and compare the telemetry coming from the sensors, with the other measurements one coming from the cameras. The only limitation in this case is that within the arena the environment is perfectly structured, so it is not possible to test the robustness of the control algorithm, i.e. its dynamic performance when the drone operates in noisy and not structured environments.

I.5 Objectives

In the previous paragraphs the UAV SoA was briefly presented, underlining as several research field are evolving. In particular, as regards the UAV features, it is clear as there isn't "a general purpose" drone, capable to accomplish whatever task. For this reason, a keyword in the next years for the researchers will be the cooperation between heterogeneous robotic platforms. More and more often complex tasks require features that a single drone doesn't have. For example, to patrol a huge area, it is required a drone suitable to fly forward with high speed, to be capable to hover and to have a good autonomy. In a few words, it is impossible for a single drone, but also for a homogeneous fleet.

From this consideration is born the objective of this thesis: to develop a fleet of heterogeneous UAV, composed by the following parts:

- A fixed wing aircraft, represented by the Volcan UAV [16].
- A quadrotor, represented by the Hummingbird produced by Asctec [60].
- A multiplatform HMI developed in LabView [48], in order to monitor and supervise the heterogeneous fleet.

The next chapters are organized in the following way: the second one describes the Volcan. The third chapter treats the Hummingbird by Asctec. The fourth chapter presents the HMI developed and finally, the last one discusses about the conclusions.

Chapter II. The Volcan UAV

II.1 Introduction

As mentioned before, the Volcan UAV is the fixed wing developed for the mission related to volcano monitoring. In order to accomplish missions in hard environments and conditions, the project designed is a V-tail fixed wings very similar to the famous Aerosonde [18] (Figure 11), with the following features:

- Fuselage in carbon fiber and fiberglass
- Wooden wing and V-tail
- A wing span of 3m
- A total weight of 13kg
- A 2000W brushless motor
- A maximum cruise speed of 150km/h

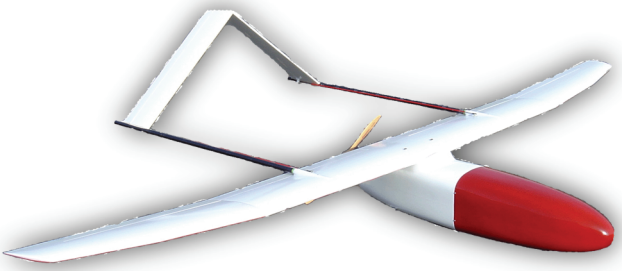

Figure 11 - Volcan UAV

The choice to develop a fixed wing is given by the fact that on the Etna the weather conditions are really adverse, not to mention the reduction of the air density, which at 3000m reduces drastically the lift of the airframe. The use of an electric engine is given by two factors:

- First of all, because the reduced air density has a negative effect on the carburetion of a stroke engine.
- Secondly, gases produced by the motor would distort the measures of the gas sensors.

The Volcan has been entirely designed in the DIEEI laboratories. In particular, during this Ph.D. activity a new control architecture has been developed [17]. In the following sections the steps that have led to the development the whole system will be discussed.

II.1.1 System architecture overview

The core of the control architecture is based on the interaction between different sub-systems developed in DIEEI laboratories (Figure 12):

- ADAHRS, the Air Data and Attitude Heading Reference System, is the sensors board and manages all the sensors in order to compute the pose and the position of the vehicle
- SACS, the Servo Actuators Control System, controls the engine and the actuators connected to the mobile parts of the drone.
- FCCS, the Flight Control Computer System, receives data from the sensors and, according with the flight plan, sends commands to the interface board.
- UDP2CAN, the data link board, connects the drone with a remote station, in order to send telemetry and to permit to the operator to supervise it.

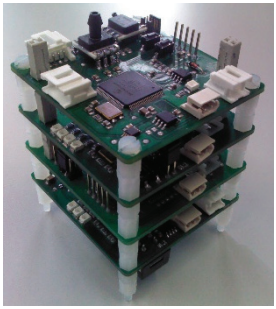

Figure 12 - Volcan control system

The whole architecture has been divided in various subsystems in order to maximize flexibility and modularity.

II.2 CAN: Control Area Network

The different subsystems forming the UAV control system need to exchange data constantly. Therefore it is necessary to use a communication protocol which gives wide guarantees of reliability and immunity to noise, moreover with a bandwidth such as to permit a real-time control of the drone. For these reasons the CANbus protocol (Controller Area Network) was chosen. CANbus is a broadcast serial bus, introduced by Bosch in the early 80s [19]. Initially designed for automotive applications, now the CANbus is used in many industrial sectors, including avionics. Its success is due to the considerable technological advantages it offers:

- Rigid Response time. This feature is fundamental for the control process.
- Simplicity and flexibility of wiring: the CAN is a serial bus which is typically implemented on a twisted pair (shielded or not, depending on the requirements).
- Multi-Master architecture, where all nodes of the network can transmit and multiple nodes of the network can request to transmit data simultaneously. They are characterized by network addresses different by the conventional sense. In fact the messages are routed on the basis on the importance of the variable to be sent and not on the basis of the address of the transmitter. Each variable has an identifier, which indicates the priority for the access to the bus. Thanks to this peculiarity, the nodes don't have an address that identifies them, so in this way they can then be added or removed to the network without reorganizing it.
- High noise immunity: the standard ISO11898 [20] imposes that the transceiver chip can continue to communicate even in extreme conditions, such as the interruption of one of the two wires or short circuit of one of them with ground or with the power supply.

The transmission rate depends on the size of the maximum length of the bus (Figure 13). In case of short distances, as in the case under exam, it is possible obtaining a bit-rate up to 1Mbit/s

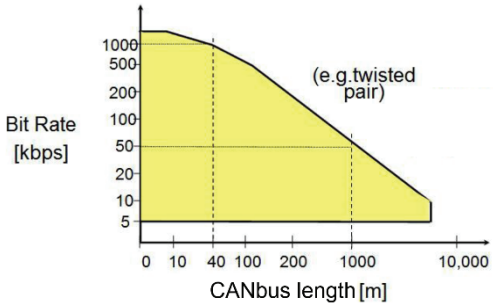

Figure 13 - CANbus data rate

II.2.1 Rules for bus access

In order to manage a multi-master architecture, the CANbus uses a modified CSMA/CD protocol, which uses the concept of dominant and recessive bits. When two or more nodes are transmitting simultaneously, the conflict is resolved with an arbitration mechanism that avoids both loss of information and time. During each transmission, the transmitting node monitors the channel and compares the level of the bit transmitted with the level on the monitored channel. If the two bits coincide, the node continues transmitting. If the level associated with the bit is recessive and in the channel there is a dominant level, the node immediately stops the transmission. Through this mechanism it is possible to assign to each CAN frame a priority level, through the *Arbitration Field*. For example, in Figure 14 three nodes try to transmit simultaneously. During the transmission of the fourth bit, node A notifies an inconsistency between what transmits and what is present on the bus, and hangs up. This is because the node A is transmitting a recessive bit, while nodes B and C a bit dominant. The same considerations applies during the transmission of the eighth bit, in which the node C hangs up and leaves the channel to node B, having an ID with a higher priority.

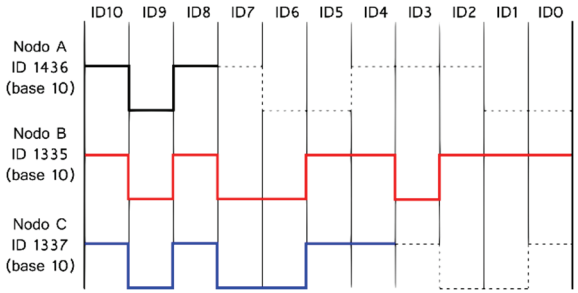

Figure 14 - Example of bus access

II.2.2 CANbus Frames

In the CANbus protocol there are five different message structures:

- Data Frame (DF): it allows the transmission of data from one transmitter node (TX) to all the others (RX). Each node decides if consider relevant or discard the received data.
- Remote Frame (RF): it has a structure similar to the Data Frame, but is devoid of the data field; it is used to request the sending of a determined Data Frame by the interrogated node.
- Error Frame: it is sent from a node that reveals an error and causes the retransmission of the message from the transmitter node.
- Overload Frame: it is sent from a node that is busy in order to delay the transmission of the next packet.
- Interframe Space: it precedes any Data and Remote Frame and has a separating function.

The last three frames are automatically generated, owing to special conditions. The implementation of the CAN protocol to communicate between the various UAV subsystems doesn't require the use of remote frames. In a few words, only the DFs will be used, which are explained in the next paragraph.

II.2.2.1 Data Frame (DF) structure

A CANbus DF consists of seven fields:

- Start of Frame (SoF): it consists of a single dominant bit and signals the start of the message. It also provides a sync function for all other nodes that detect the start of transmission.
- Arbitration Field: it contains the identifier of the content of the message. The identifier has 11 bits in the CAN protocol 2.0A (Standard CAN) or 29 bits in the CAN 2.0B (Extended CAN).
- Control Field: it consists of 6 bits, 4 are used to specify the number of bytes of the Data Field (DLC) and 2 are reserved for future expansion of the protocol.
- Data Field: it contains the data, ranging from a maximum of 8 bytes to a minimum of 0. The bytes are sent from the most significant to the least significant.
- CRC Field: it consists of 16 bits, the first 15 contain the control sequence (cyclic redundancy check) and the last bit is a recessive delimiter. If the cyclic redundancy code does not reveal the presence of errors, the node puts a recessive bit in the ACK field of the current Data Frame.
- ACK Field: it is constituted by an ACK bit and another delimiter bit. They are both sent as recessive, but ACK Slot is overwritten as a dominant by every node that receives the message correctly. In this way the TX node knows that at least one node has received the message correctly.
- End of Frame (EoF): it is made up of 7 recessive bits that indicate the end of the Frame.

A graphical representation of the DF structure is shown in Figure 15

Figure 15 - DF structure

II.2.3 CANAerospace

The CANbus covers only the first two levels of the ISO/OSI protocol, the physical layer and the data link layer. CANaerospace [21] is a specification that defines the application level, specifically for use in avionics and aerospace field. It was introduced in 1997 by Stock Flight Systems, a German company founded in 1993, now partner of many leading international aerospace companies. A subset of the specification has been standardized by NASA in 2001 as AGATE (Advanced General Aviation Transport Experiment) Avionics Databus. The specifications dictated by CANAerospace are used to arrange the Arbitration Field and Data Field of the data frame previously treated. CANAerospace supports both the CAN 2.0A (11bit) and CAN 2.0B (29bit) identification, with whatever bit-rate. For the Volcan control system a standard identifiers to 11bit and a bit-rate equal to 1 Mbit/sec have been chosen.

II.2.3.1 Arbitration Field

In according with CANAerospace directives, the ID of a given DF has the priority summarized in Table 3:

Message Type	ID range	Description
Emergency Event Data (EED)	0 - 127 0x000-0x07F	Transmitted asynchronously whenever a situation requiring immediate action occurs.
High-priority Node Service Data (NSH)	128 – 199 0x080-0x0C7	Transmitted asynchronously or cyclic with defined transmission intervals for operational commands
High-priority User-defined Data (UDH)	200 - 299 0x0C8 - 0x12B	Message/data format and transmission intervals entirely user-defined
Normal Operation Data (NOD)	300 - 1799 0x12C – 0x707	Transmitted asynchronously or cyclic with defined transmission intervals for operational and status data.
Low-priority User-defined Data (UDL)	1800 - 1899 0x708 – 0x76B	Message/data format and transmission intervals entirely user-defined
Debug Service Data (DSD)	1900 - 1999 0x76C - 0x7CF	Transmitted asynchronously or cyclic for debug communication &software download actions.
Low-priority Node Service Data (NSL)	2000 - 2031 0x7D0 -0x7EF	Transmitted asynchronously or cyclic for test & maintenance actions

Table 3 – Arbitration field CANAerospace

II.2.3.2 Data Field

The data field of a DF conforms to the specifications of the CANAerospace consisting of 8 bytes, 4 bytes form a header and the remaining 4 form the real data field (Figure 16). The header includes the following fields:

- Node ID, indicates the address of the node transmitter in the case of messages EED / NOD or the receiver address for messages NSL / NSH (the identifier 0x00 is reserved for broadcast transmissions).
- Data Type, specifies the data type of the last four byte. Are supported both standard data types and types specified by the user depending on the application.
- Service Code, reserved for specific purposes in the case of messages EED/NOD, or used to define the type of service in case of messages NSL/NSH.
- Message Code, is essentially a counter message used for debugging purposes in case of message EED/NOD.

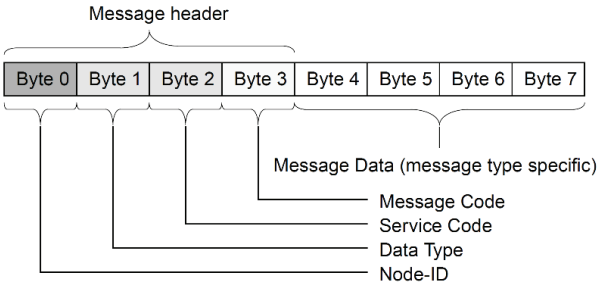

Figure 16 - Data field of a CANAerospace DF

A complete description of the CANAerospace frames used in the Volcan UAV is available in [22].

II.3 Flight Control Computer System

The FCCS subsystem is the core of the control architecture. It is responsible of the stability of the aircraft, in addition to the management of a given flight plan. In other word, the FCCS is the controller of the whole system, since it provides two level of control:

- A low level control, in order to ensure the stability of the aircraft. In this case, the FCCS needs to receive roll and pitch data from an IMU, and it acts only on the ailerons and the elevator.
- A high level control, to accomplish a given flight plan, generally formed by a set of waypoints. In this scenario, in addition to the IMU, magnetometers, GPS and pressure sensors are needed.

II.3.1 Control strategy

In the scientific literature there are various control methods reported [24] [36] [42] [43] and the choice of which type to adopt is of crucial importance for the performance of the system. A first step in this evaluation is to list all the variables that are supposed to be controlled, i.e. Euler angles, GPS and air pressure sensors. Secondly, it must consider whether and how these variables are related to each other: in this case it is necessary that their control is in some way correlated. For instance, taking into consideration the roll and yaw, is evident as this two variables are strictly related. To point along a given direction (yaw), a plane must first turn activating the ailerons on the wings, thereby setting a certain roll angle. From the above it is clear another key observation concerning the dynamics of the two variables: the roll has a dynamic faster than the yaw, then to control these two variables a cascade control represents a suitable solution [23]. As regards pitch and altitude, the relation is the same. For this reason, two cascaded PID control loops, one for the heading-by-roll and one for the altitude-by-pitch regulations, and a simple feedback PID control loop for the speed regulation are implemented in the control architecture of the Volcan UAV [17]. The developed controllers are a similar implementation of the Altitude-Hold and Heading-Hold schemes presented in [24].

The "Mission Management" block of the FCCS supervises and controls the mission execution: the desired trajectory is assigned to the FCCS by means of a set of waypoints coordinates [32] and the "Mission Management" block

computes the reference signals to be assigned to the control loops with the aim of performing the planned mission. As it can be observed in Figure 17, the heading-by-roll regulator acts on the ailerons and the rudder, the altitude-by-pitch computes the signals for the elevator, while the throttle command is computed by the airspeed controller.

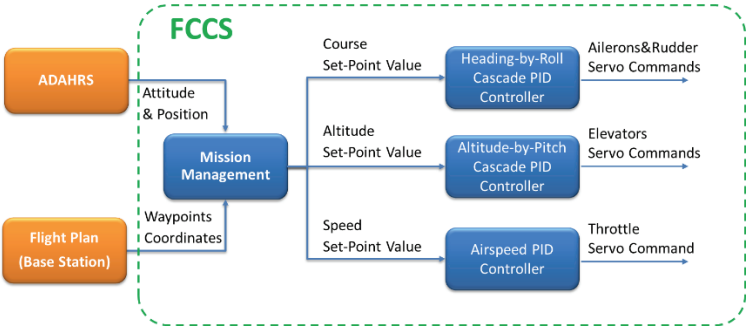

Figure 17 - The interaction of the different sub-systems implemented on the FCCS.

In Figure 18 the block scheme of the altitude-by-pitch regulator is shown. The reference altitude is the height of the next waypoint to be reached. The current altitude is obtained by means of an EKF-based sensor fusion between the altitudes given by the on-board GPS and by the absolute pressure sensor of the ADAHRS board; the computed altitude is sent via CANAerospace protocol to the FCCS. The resulting error is processed by PID_{Alt} which provides the reference signal to the inner loop, regulating the pitch angle.

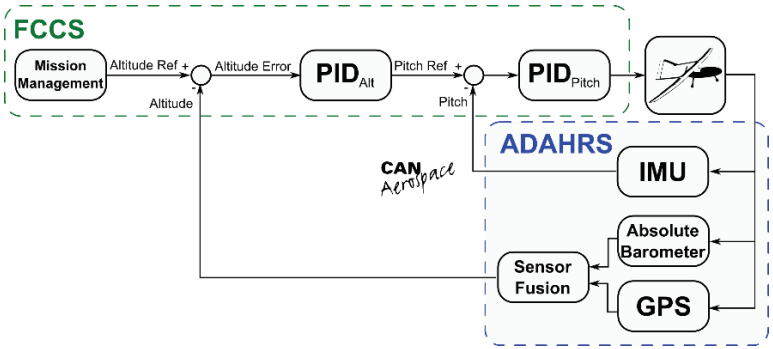

Figure 18 - Altitude-by-pitch control loop.

In Figure 19 the block scheme of the heading-by-roll control loop is shown. The course error is determined as it can be seen in Figure 20. The value of the desired course essentially depends on the coordinates of the next waypoint and the current coordinates of the aircraft (given by GPS); the measure of the heading is obtained from the ADAHRS. Obviously, this represents a simplification, because we are considering the heading as the direction in which the plane is pointing, without taking into account the effects of wind drift [26]. However, this problem is compensated by the FCCS navigation algorithm that continuously updates the Course Error. The resulting error is processed by the PID_{Course} regulator, which provides the reference signal to the inner loop, regulating the roll angle.

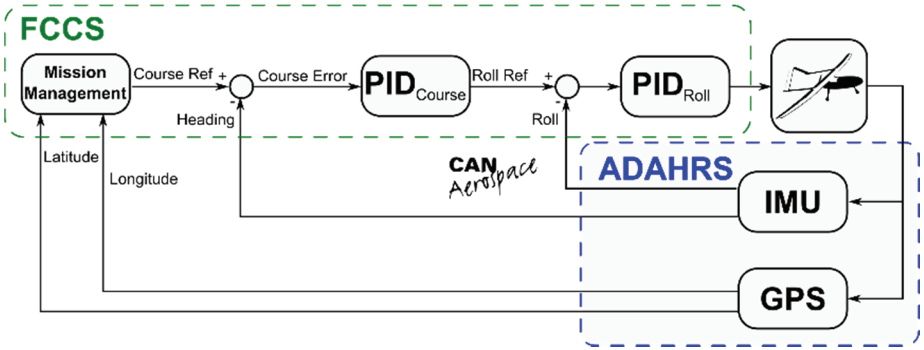

Figure 19 - Heading-by-roll control loop.

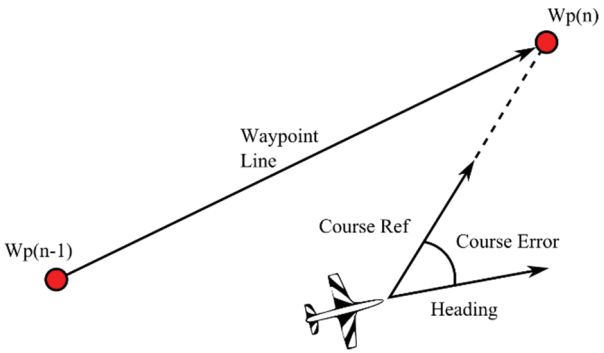

Figure 20 - Course error computation.

Finally, in Figure 21 the speed control block scheme is shown. The desired speed is related to the next waypoint, whereas the current speed is obtained

through a sensor fusion between the speed coming from GPS and the airspeed obtained by the differential pressure sensor connected to the Pitot tube.

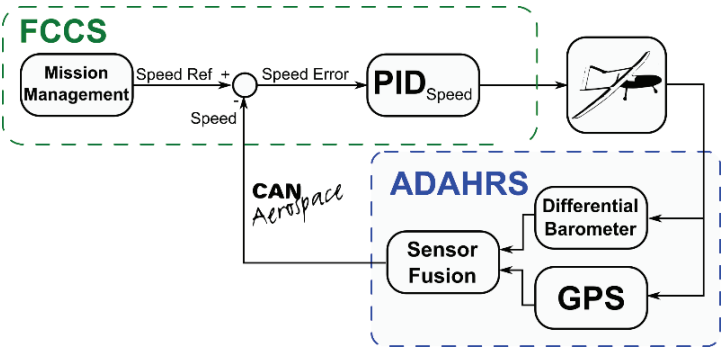

Figure 21 – Speed control loop.

II.3.2 Hardware Design

The complete schematic of the FCCS subsystem is shown Appendix A. The main component is the dsPIC33FJ256GP710A, a microcontroller produced by Microchip [65]. Moreover, an EEPROM memory and a transceiver CAN are present. In Figure 22 the developed PCB is shown.

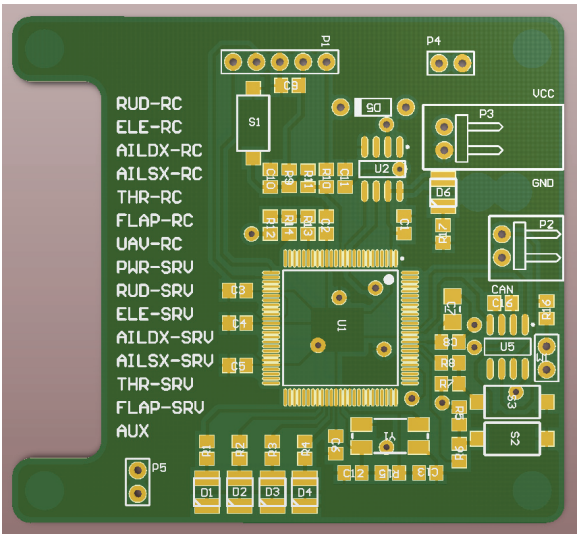

Figure 22 - FCCS board

II.3.3 Operating modes

The operating modes of the FCCS determine the particular conditions in which the UAV operates, which results in a different iteration with the environment. Essentially, the operating modes necessary are two: the first is necessary for the tuning of the PIDs, and the second one is used to accomplish a given flight plan.

II.3.3.1 Assisted Mode

In assisted mode the aircraft does not follow any flight plan, but the reference values of the controlled variables (Roll, Pitch, Heading, Altitude and Speed) set by the user. This mode is particularly useful during calibration of the control loops or to verify the correct operations of them. Using the complete cascade control, it is possible to assign the reference to the variable on the outer loop (Heading and Altitude) Only. To assign the reference variables in the inner loop (Roll and Pitch) the outer loop should be opened, transforming the cascade control in a simple feedback control (in this configuration, yaw and altitude are not checked). For this reason two flags are inserted in the heading-by-roll and in the altitude-by-pitch control loops, *TrackHold* and *AltitudeHold*. These flags make it possible to set the reference of the outer loop, or inhibit it and give a reference value directly to the inner loop. As regards the speed control, being controlled by a simple feedback control, it always follows the reference set by the operator. This is summarized by Table 4 and Figure 23.

Flag →	**TrackHold**		**AltitudeHold**	
Setpoints ↓	True	False	True	False
RollAssisted	Notused	Used	X	X
TrackAssisted	Used	NotUsed	X	X
PitchAssisted	X	X	Notused	Used
AltitudeAssisted	X	X	Used	NotUsed
AirspeedAssisted	X	X	X	X

Table 4 - Assisted Mode Flags

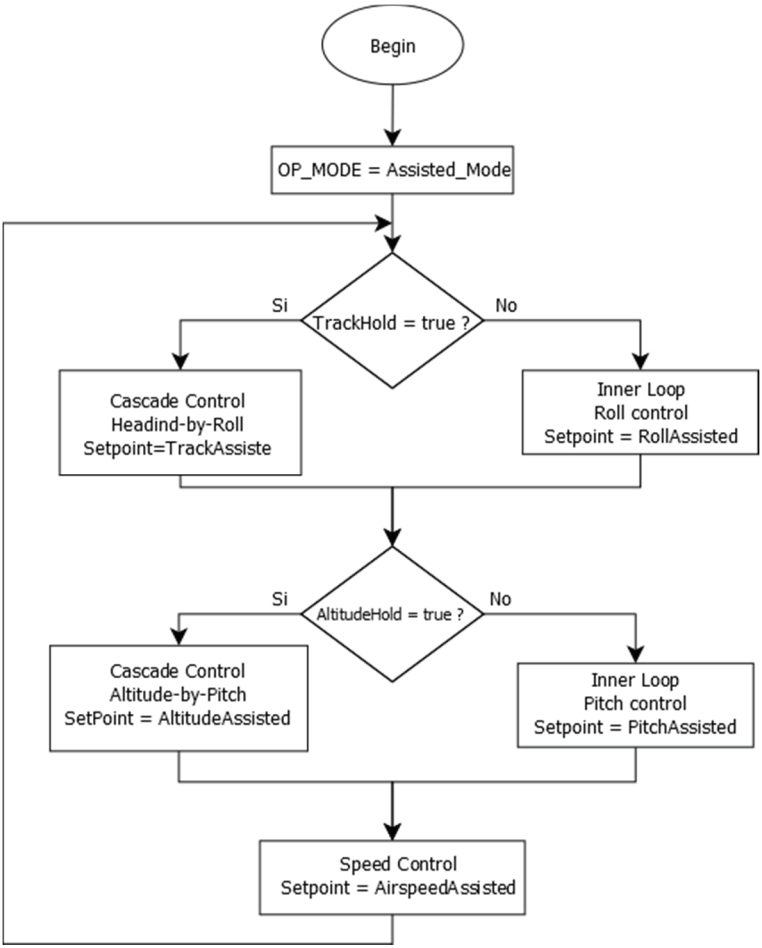

Figure 23 - Assisted Mode Flow chart

II.3.3.2 Navigation

This is the typical operating mode of the UAV, i.e. the navigation between waypoints. In this modality a waypoint is considered reached if the UAV is located within a given radius from it (typically chosen as 50m in this work). In Figure 24 the flow chart of this modality is shown.

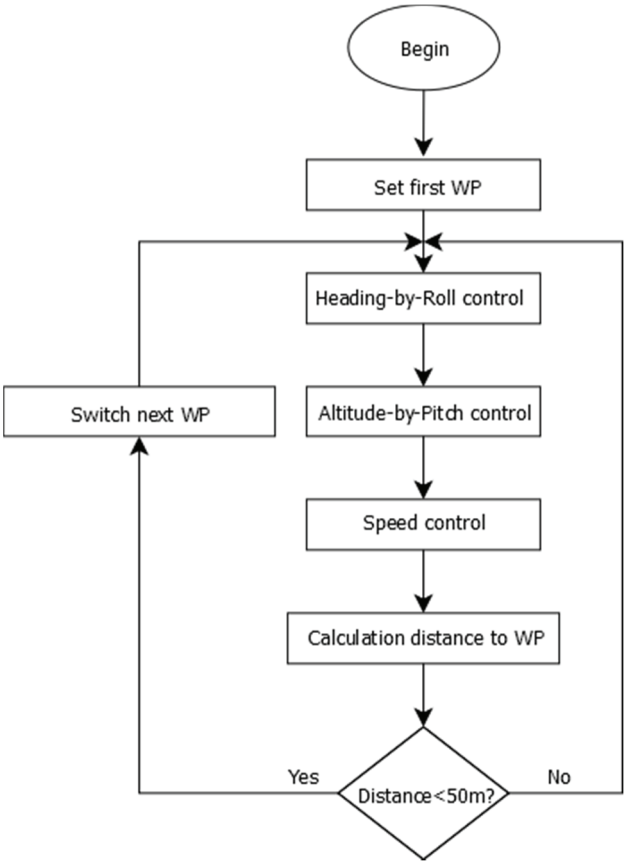

Figure 24 - Navigation mode flow chart

II.3.4 Autotuning algorithm

The tuning procedures of the control algorithms of unmanned platforms represent one of the most time consuming phases, especially in the case of flying robots adopted in strongly not structured environments, like in volcanoes [11]. Usually a new mission needs to be preceded by a tuning procedure of the control loops, depending on weather and environmental conditions (pressure, temperature and so on). Propellers efficiency, wings lift and the power of a stroke engine depend on air density, which is strongly related to the weather conditions.

The implementation of automatic procedures represents an advantage that makes the development phase of an UAV easier and faster. Several papers deal with the automatic or self-tuning of control systems [34]. However, only a few

attempts are related to UAVs [28] [29] [30] [31]: this is mainly due to the risks connected to the tuning procedures of this kind of robotic platforms.

The relay feedback technique is widely adopted for the automatic tuning of PID controllers [27]. However, such algorithm is unsuitable for the automatic tuning of the control loops of an aircraft. Indeed, the first phase of this algorithm leads the system in a steady-state oscillation. It is clear how dangerous could be this condition for an aircraft. A different approach is based on Fuzzy rules, suggested by experience. The papers [28] and [29] are examples of this approach, where a self-adaptive fuzzy control is used to tune the PIDs of an UAV.

However, up to now the literature presents only works focused on the different methodologies to be adopted for the self or automatic tuning of UAVs. The step ahead is represented by the development of an hardware and software suite that makes the tuning phase safe, reliable and fast, independently of the control loops implemented on the on-board avionics [25].

The tuning algorithm analyses the step response of the control loop to be tuned and automatically implements a method based on Åström and Hägglund [27], where the PID parameters are chosen as a compromise between stability and speed, as summarized in Table 5.

	Speed	**Stability**
Proportional Action increases	Increases	Reduces
Integral Action increases	Reduces	Increases
Derivative Action increases	Increases	Increases

Table 5 - Effect of the controller on speed and stability

Going into detail, a constant set-point is assigned to the control loop to be tuned and the response of the aircraft is analyzed to verify if the desired behavior is achieved. A set of constraints, assigned before the tuning procedure, must be satisfied: rise time, overshoot, steady state error and settling time (Figure 25). The values assigned to the constraints should be based on the specific aircraft model and, in general, on experience.

The algorithm, developed in Matlab-Simulink [68], communicates with the onboard avionics via CANAerospace protocol. A screenshot of the GUI implemented for executing and monitoring the tuning procedure is shown in Figure 27: in particular, the form dedicated to the automatic tuning is represented. This form allows the operator to assign the desired constraints ("Specifications" section), to start the tuning procedure ("Auto Tuning" section),

to analyze in real-time the response of the control loops ("Step Response" section) and to view the computed parameters ("Dynamic Features" section).

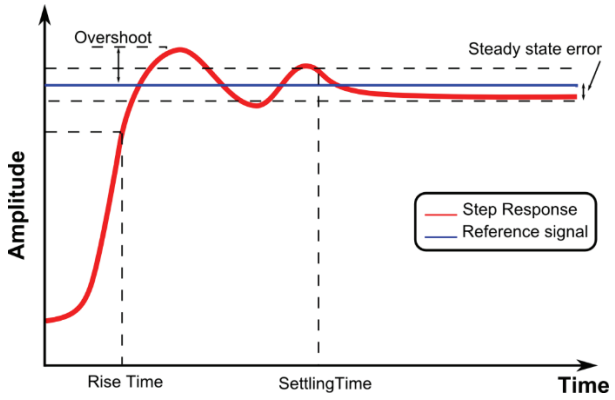

Figure 25 - Generic step response

The tuning procedure, automatically executed by the implemented algorithm and started by acting on the "Auto Tuning" section of the GUI, is shown in Figure 26 and summarized below:

- The constraints must be introduced in the "Specifications" section of the developed HMI.
- An initial set of the PID parameters is assigned and sent via CANAerospace to the control loops of the FCCS.
- A step signal is assigned as input reference for the autopilot.
- The dynamic response of the system is analyzed and is compared with the assigned constraints.

If the response is not satisfactory, the PIDs gains are modified according to the following classical rules [27]:

- The proportional action is correlated to the speed and, then, to the rise time;
- The derivative action is correlated with the overshoot;
- The integral action acts on the steady state error.

This procedure is iterated until the constraints are not satisfied or a specified number of iterations is reached.

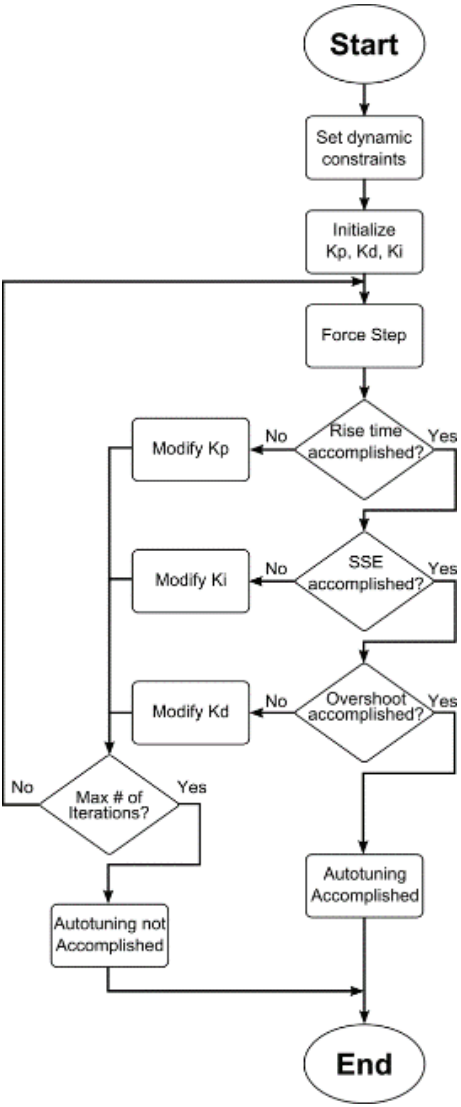

Figure 26 - Flow chart of the developed automatic tuning algorithm

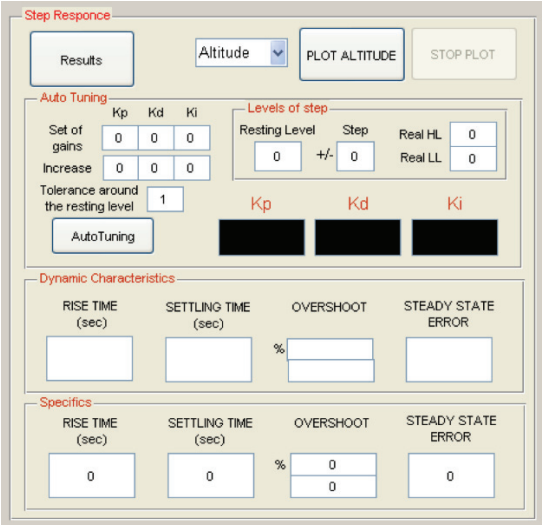

Figure 27 - The developed GUI for the autotuning procedure

II.3.5 HIL Architecture

Hardware in the Loop architectures have been adopted by several projects to develop and test different aeronautical components [33] [35] [36].

To reduce time, costs and risks related to the trials on a real aircraft, an HIL architecture has been used to test and verify the developed architecture: the real aircraft has been substituted with a simulated virtual model closed in a HIL architecture with the real controller.

The used HIL architecture is similar to the one adopted in [32] to develop the VOLCAN project. In this architecture the VOLCAN has been replaced by the X-Plane Simulator by Laminar Research [53], connected both to the FCCS and the GUI via CANAerospace. In Figure 28 the adopted architecture is shown; the block named "FCCS" represents the real electronic board. Telemetry data, concerning plane pose, are sent from the simulator through CANbus to the FCCS board by using the CANaerospace protocol. Once attitude and position of the aircraft are known, the control algorithms implemented on the FCCS board compute the signals for the servo commands that are sent back to the simulator to actuate the mobile parts of the plane.

The GUI, presented in II.3.4, is closed in the loop to execute the automatic tuning procedure, sending the reference signals to the FCCS and capturing and recording the telemetry data sent by the simulator.

In the next paragraphs, several experimental results are presented. In particular, in the first part the procedure and the results related to the automating tuning of the "heading-by-roll" control loops (Figure 19) are discussed. The first step regards the tuning of the inner (and faster) variable, the roll (II.3.5.1). Once the roll loop has been tuned, the procedure to tune the outer loop (heading, with a slower dynamic) is automatically executed (II.3.5.2).

To validate the results of the automatic tuning algorithm, the computed parameters have been used during the execution of a complete flight mission in absence of wind (II.3.5.3) and in windy conditions (II.3.5.4).

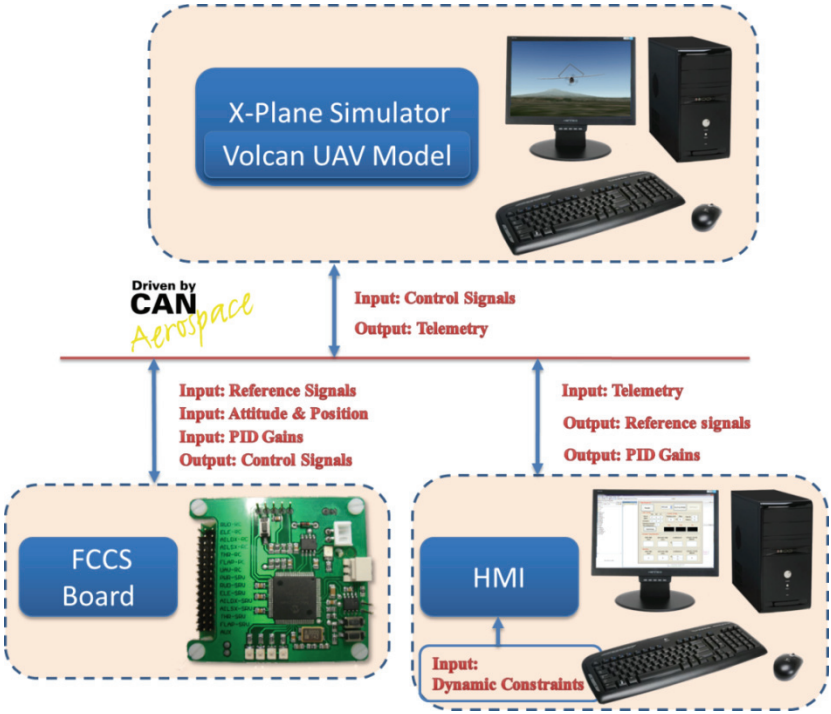

Figure 28 - The HIL architecture adopted to test the algorithm

II.3.5.1 Automatic tuning of the roll control loop

To tune the roll control loop, the outer loop (heading control) is deactivated and the reference signal is directly assigned to the inner control loop.

To execute this procedure, the aircraft has been placed at an altitude of 200m ASL, with a ground speed of 90 Km/h. The reference roll signal used to evaluate the step response has been set to ±30°: the procedure is considered as successfully completed when the constraints are satisfied both on the positive (+30°) and the negative (-30°) roll steps.

The constraints to be satisfied, assigned at the beginning of the procedure, are reported in the first row of Table 6; the second row reports the values reached at the end of the tuning procedure.

The initial values of the PID gains are shown in the first row of Table 7, while the computed parameters are reported in the last row.

Parameters	Rise time [s]	Settling time [s]	Overshoot [DEG]	Overshoot [%]	SSE [DEG]
Imposed	1	1	0.3	1	0.3
Achieved	0.82	0.82	0.17	0.58	0.21

Table 6- Constraints and results of roll loop tuning.

Gains	Kp	Kd	Ki
Starting values	1.5	0	0
Final values	2.4	0	0

Table 7 - Roll PID gains.

Figure 29 shows the time evolution of the roll angle, measured during the tuning procedure. As it can be observed, the responses of the first eighth steps are incomplete (A), because the constraint related to the rise time on the positive edge is not satisfied. In fact, when the reference signal is not reached in the assigned rise time, the algorithm stops the step under execution; then, the K_P gain is modified and another step is executed.

At the ninth step (B), the constraints are satisfied on the rising edge of the +30° imposed roll; but they are not satisfied on the negative step.

Finally, at the tenth attempt (C), the algorithm executes positive and negative steps and verifies that the imposed constraints are achieved (Table 6), by acting only on proportional action (Table 7).

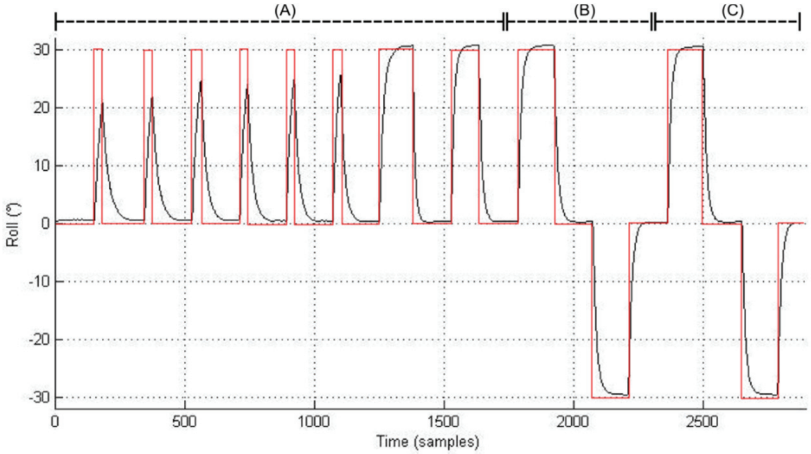

Figure 29 - Roll control loop auto-tuning procedure

II.3.5.2 Automatic tuning of the heading control loop

Once the roll loop has been tuned, the outer loop (heading) is reactivated while the PID gains of the inner loop (roll) are those achieved in the previous roll loop tuning. In this case, the reference amplitude, used to evaluate the step response, has been imposed to ±45°. Constraints and results are summarized in Table 8, while the achieved PID gains are shown in Table 9.

Parameters	Rise time [s]	Settling time [s]	Overshoot [DEG]	Overshoot [%]	SSE [DEG]
Imposed	6	6	2	4.44	2
Achieved	4.2	4.26	0.44	1	1.25

Table 8 - Constraints and results of heading loop tuning

Gains	Kp	Kd	Ki
Startingvalues	0.8	1	0
Increments	0.1	0.1	0.01
Finalvalues	1	1.2	0.01

Table 9 - Heading PID gains

In Figure 30 the result of the heading control loop automatic tuning procedure is shown. Likewise to the roll tuning procedure, the first two steps (A) are partial, because the rise time constraint is not satisfied on the positive edge; then, the constraints are satisfied for the positive edge but not on the negative edge (B). Finally, the assigned specifications are satisfied both on the positive and the negative edges (C).

In order to tune the remaining control loops (pitch, altitude and speed) automatic tuning procedure have been executed and the algorithm has always found suitable PID gains satisfying the imposed constraints.

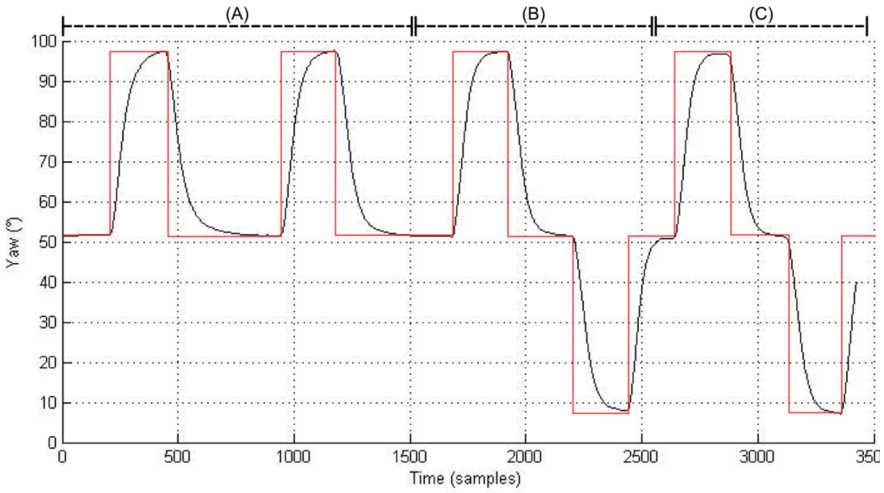

Figure 30- Heading control loop auto-tuning

II.3.5.3 Parameters validation: mission execution without wind

The same procedure used for the "heading-by-roll" control loops has been adopted for the automatic tuning of the "altitude-by-pitch" and "speed" control loops.

Then, a mission has been executed in the HIL architecture to validate the obtained parameters. In Figure 31 the mission is represented: the red circles are the assigned WPs while the purple line represents the executed trajectory. Table 10 summarizes the assigned WPs and the altitude and speed assigned for each one.

Waypoint	Latitude [Deg]	Longitude [Deg]	Altitude [m]	Speed [km/h]
WP1	37.4728737	15.0714064	200	80
WP2	37.4591484	15.0772877	500	110
WP3	37.4603195	15.0517006	300	90

Table 10 - Waypoint assigned for parameters validation

Figure 31- The mission executed after the automatic tuning procedure

Figure 32 allows to observe the behaviors of the two control loops involved in the "heading-by-roll" regulation during the mission. The picture shows the time evolution during the take-off, navigation and landing phases.

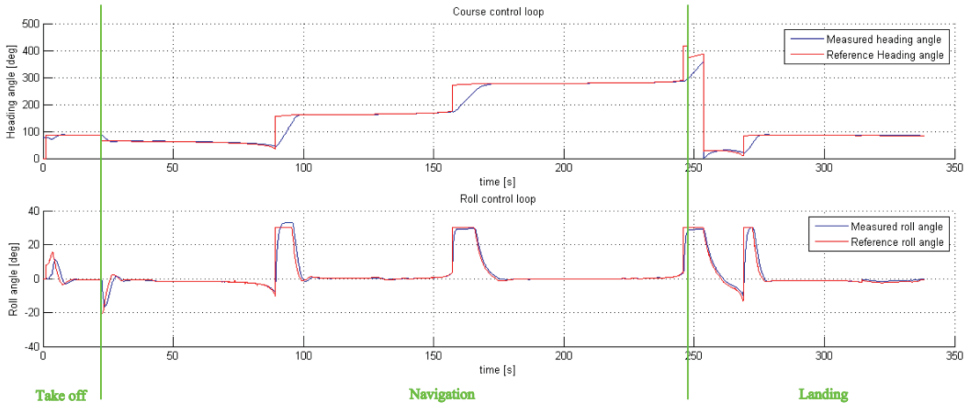

Figure 32 - The time evolution of the heading-by-roll control loops

In Figure 33 the time evolutions of the control loops involved in the "altitude-by-pitch" regulation is shown.

As it can be observed by analyzing Figure 31, Figure 32 and Figure 33, the behavior of the tuned loops is fine and allows to execute the flight plan in a satisfactory way.

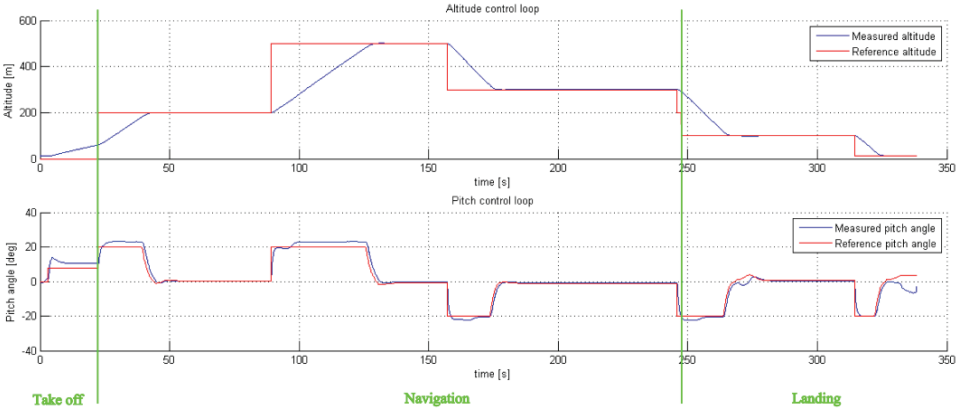

Figure 33 - The time evolution of the altitude-by-pitch control loops

II.3.5.4 Parameters validation: mission execution in windy conditions

To validate the robustness of the control architecture with the parameters computed by the automatic tuning algorithm, the same mission assigned in the previous section has been executed in windy conditions, introducing in the simulator a wind of 20 km/h and with the direction shown in Figure 34. Such a wind condition is remarkable taking into account the aircraft under test.

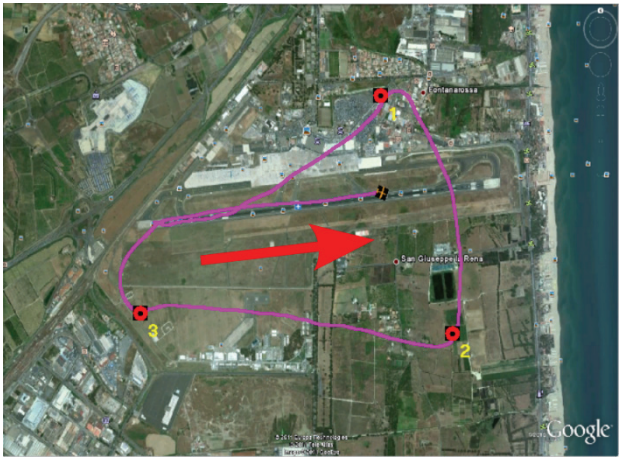

Figure 34- The mission executed in windy conditions. The red arrow indicates the wind direction.

Figure 35 allows to observe the behaviors of the two control loops involved in the "heading-by-roll" regulation during the mission. The picture shows the time evolution during the take-off, navigation and landing phases.

In Figure 36 the time evolutions of the control loops involved in the "altitude-by-pitch" regulation is reported.

As it can be observed by analyzing Figure 35 and Figure 36, the response of the inner loops (roll and pitch) allows to obtain a good behavior of the outer loops (heading and altitude). This permits the aircraft to reach successfully the assigned waypoints, even if the trajectory is disturbed.

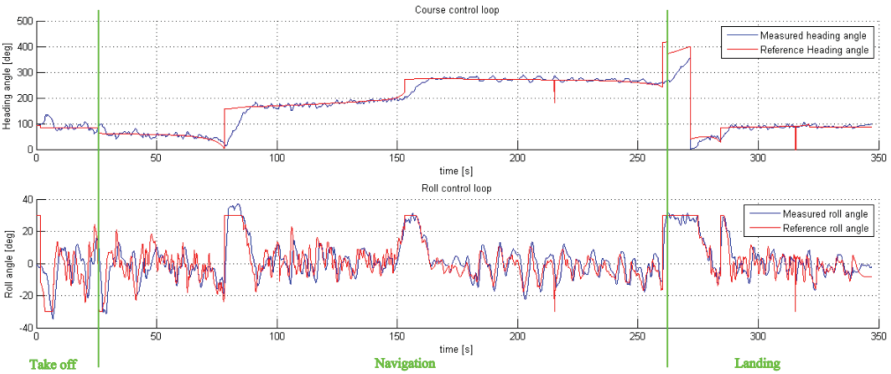

Figure 35 - The time evolution of the heading-by-roll control loops in windy conditions.

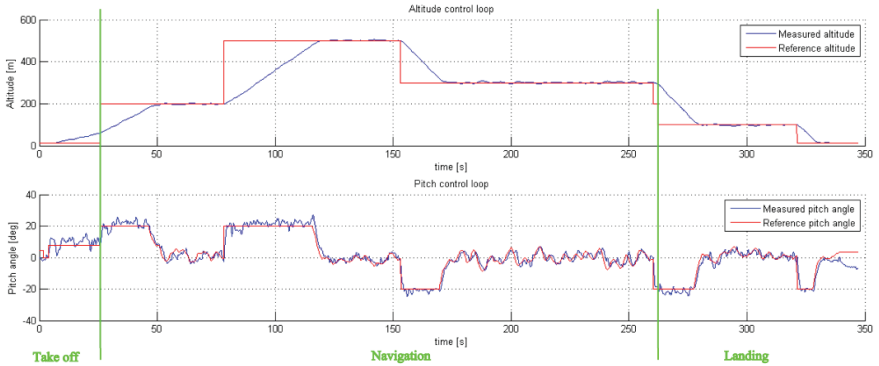

Figure 36 - The time evolution of the altitude-by-pitch control loops in windy conditions

II.4 Servo Actuators Control System

This subsystem manages the engine and the actuators (up to seven) connected to the mobile parts of the Volcan UAV. It also allows switching from the UAV mode to the Pilot In Command (PIC) mode, in order to execute takeoff and landing operations or also to bypass instantaneously the FCCS in case of failures. Therefore, in UAV mode the reference signals come from the FCCS, whereas in PIC mode they come from the RC receiver. The core of the SACS is the microcontroller PIC18F4580 by Microchip [65]. Moreover, a transceiver CAN and an array of digital isolators, in order to isolate the ground of the servos (generally noisy) with the ground of the microcontroller, are mounted. PCB of the SACS is shown in the following figure, whereas the schematic is reported in Appendix A.

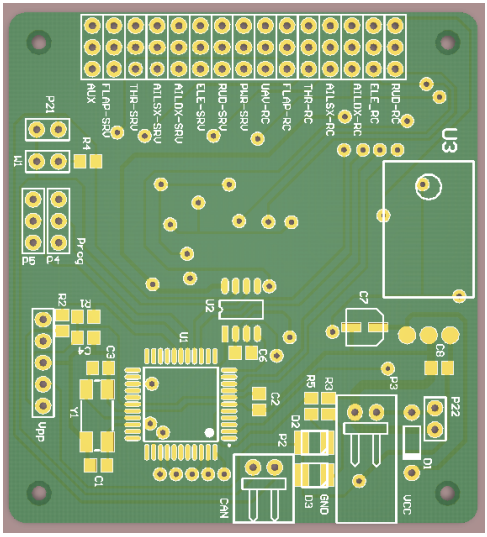

Figure 37 - SACS board

II.5 UDP2CAN

This subsystem acts as a bridge between the CANAerospace bus and a wireless link. This is because it is necessary for the operator to supervise and monitor the drone by means of a ground station. Obviously, a wireless link does not have the same robustness of a wired link. However, considering that the ground station receives only telemetry data and it sends asynchronous commands with ACKs (the NSH frames), it's clear as this type of connection represents an appropriate solution.

Going into details, an UDP2CAN bridge has been realized, i.e. a device that converts CAN frames coming from the drone control system in UDP frames suitable for a commercial WiFi link.

As regards the hardware, this subsystem has the same microcontroller of the FCCS, the dsPIC33FJ256GP710A, in addition to a transceiver CAN (MAX3051) and the ENC28J60, an Ethernet controller produced by Microchip [65]. PCB of the UDP2CAN is shown in the following figure, whereas its schematic is reported in Appendix A.

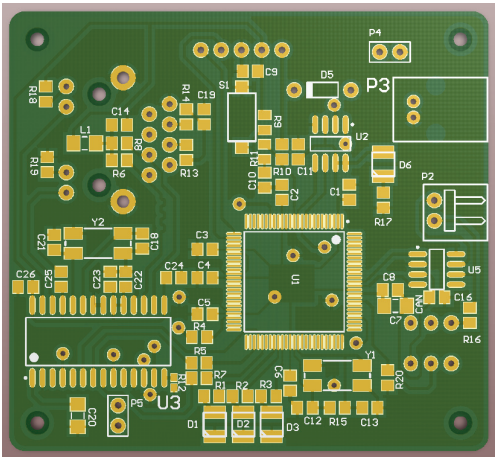

Figure 38 – UDP2CAN board

II.6 Air Data and Attitude Heading Reference System

The last subsystem of the whole control architecture is represented by the ADAHRS. In reality, for reasons of convenience, this subsystem is split into two boards:

- A sensors board, which manages GPS, pressure sensors and temperature sensor.
- An Inertial measurement unit, that gives information about attitude and heading. To this board, considering its complexity, the section II.7 is dedicated.

II.6.1 Sensors Board

The dsPIC33FJ256GP710A onboard manages the following sensors:

- A classical GPS with RS232 interface.
- The STTS75 Digital temperature sensor.
- The absolute pressure sensor HSCMAND015PA2A3 by Honeywell [66], used as barometer in order to obtain information about altitude.
- The differential pressure sensor HSCMRRN100MD2A3 by Honeywell [66], used as Pitot tube in order to obtain information about airspeed.

PCB of the sensors board are shown in the following figures. Its schematic is shown in Appendix A

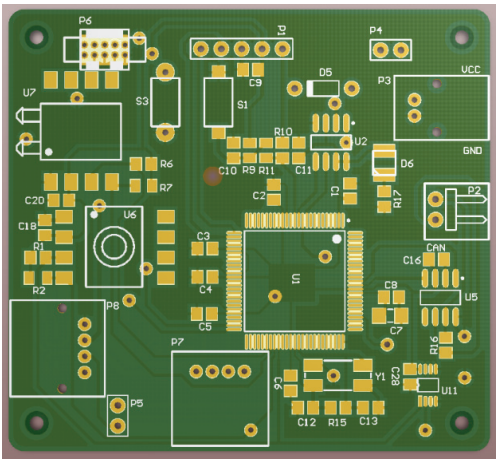

Figure 39 – Sensors board

II.7 IMU board

The aim of this board is to provide the Euler angles of the aircraft. To make this possible, a set of inertial sensors are needed. Such a set of sensors, together with a 32bit ARM microcontroller, are present in the INEMO®-M1, produced by ST-Microelectronics [37]. Going into details, the key features of this device are the following:

- STM32F103REY6: WLCSP package, high-density performance line ARM®-based 32-bit MCU
- LSM303DLHC: 6-axis digital e-compass module, ±2g, ±4g, ±8g, ±16g linear acceleration programmable full scale, from ±1.3 gauss to ±8.1 gauss, I²C digital output
- L3GD20: 3-axis digital gyroscope (roll, pitch, yaw), 16-bit data output, ±250°/s, ±500°/s, ±2000°/s selectable full scale
- LDS3985M33R: ultra-low drop, low-noise BiCMOS 300 mA onboard voltage regulator.
- Flexible interfaces: CAN, USART, SPI and I²C serial interfaces; full-speed USB 2.0
- Up to 8 ADC channels for external analog inputs
- Compact design: 13 x 13 x 2 mm

Figure 40 - INEMO®-M1

II.7.1 Hardware development

In order to interface the INEMO®-M1 with the other subsystems, some improvements have been added. In particular, a transceiver CAN and an SWD connector (to program the microcontroller) were inserted. In the following figure, the board is shown. The schematic is in Appendix A.

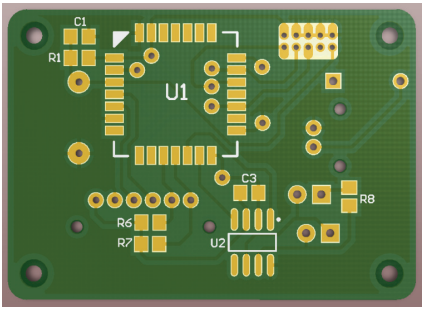

Figure 41 - INEMO®-M1 Board

II.7.2 Firmware development

The IMU firmware development has represented one of the most complex phases in the design of the Volcan control system. The reason is that to develop a reliable system that calculates the RPY angles from a set of inertial sensors requires a very long phase of test and optimization. In the next paragraphs the various development and optimization steps are described.

II.7.2.1 CANAerospace implementation

The first step in the IMU development is represented by the implementation of the CANAerospace protocol. The CANAerospace directives impose only the most important frame structures, such as the frames for the RPY angles. There is therefore the possibility to insert several user defined frames, in order to make possible the management and the personalization of such a system. A complete documentation of the CANAerospace protocol implemented is reported in Appendix B.

II.7.2.2 Extended Kalman Filter implementation

The classical Kalman filter [38] [39] is an optimal observer under the hypotheses that the system is linear and that both the process and the measurement noises are Gaussian and additive. The Kalman Filter equations are summarized in Table 11

Initial estimates for $\hat{x}_{k-1}$ and P_{k-1}	
Time Update ("Predict")	
Project the state ahead	$\hat{x}_k^- = A\,\hat{x}_{k-1} + B\,u_k$
Project the error covariance ahead	$P_k^- = AP_{k-1}A^T + Q$
Measurement Update ("Correct")	
Compute the Kalman gain	$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$
Update estimate with measurement zk	$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\,\hat{x}_k^-)$
Update the error covariance	$P_k = (I - K_k H)\,P_k^-$

Table 11 - Kalman Filter Equations

Unfortunately, in this case of study the hypothesis of linearity is not satisfied. However, under the assumption that the process and the measurement noises are Gaussian and additive, it is possible to implement an Extented Kalman Filter (EKF) algorithm [40] [41]. With respect to a conventional Kalman filter, the EKF is its linearization around the current estimate. Considering a given process with the following equations:

$$\begin{cases} \boldsymbol{x_k = f(\,x_{k-1},\,u_k, w_{k-1})} \\ \boldsymbol{z_k = h(\,x_k, v_k)} \end{cases}$$

Where x_k represents the state variables, z_k the measurements and f, h non-linear functions. The following Jacobians represent the linearization of the system around the current estimate:

$$W_{[i,j]} = \frac{\partial f_{[i]}}{\partial w_{[j]}}(\hat{x}_k, u_k, 0)$$

$$A_{[i,j]} = \frac{\partial f_{[i]}}{\partial x_{[j]}}(\hat{x}_k, u_k, 0)$$

$$H_{[i,j]} = \frac{\partial h_{[i]}}{\partial x_{[j]}}(\hat{x}_k, 0)$$

$$V_{[i,j]} = \frac{\partial h_{[i]}}{\partial v_{[j]}}(\hat{x}_k, 0)$$

The resulting equations are shown in Table 12

Initial estimates for $\hat{x}_{k-1}$ and P_{k-1}	
Time Update ("Predict")	
Project the state ahead	$\hat{x}_k^- = f(\hat{x}_{k-1}, u_k, 0)$
Project the error covariance ahead	$P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T$
Measurement Update ("Correct")	
Compute the Kalman gain	$K_k = P_k^- H^T (H P_k^- H^T + V_k R_{k-1} V_k^T)^{-1}$
Update estimate with measurement zk	$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-, 0))$
Update the error covariance	$P_k = (I - K_k H) P_k^-$

Table 12 - EKF equations

State Vector

As regards the state vector, initially the best choice would seem to be the Euler Angles RPY. However, the adoption of Euler Angles causes two drawbacks:

- Gimbal lock: the loss of one DoF in a 3D space that occurs when 2 axes are parallel.
- Mathematical singularities, caused by the trigonometric function atan2.

In order to overcome these problems, another representation of spatial orientation of a rigid body has been used, the unit quaternion [40] [41]: an efficient and non-singular description of spatial orientation used in particular for calculations involving three-dimensional rotations, such as in three-dimensional computer graphics and computer vision. Obviously, it is possible to convert the unit quaternion in Euler Angles and viceversa [44]. Another aspect to take into consideration is the gyroscopes drift. For this reason also the gyroscope biases are chosen as state variables. In summary the state vector of the EKF is the following:

$$x = [q_0 \quad q_1 \quad q_2 \quad q_3 \quad b\omega_x \quad b\omega_y \quad b\omega_z]^T$$

Prediction phase

In the discrete time, using the Euler integration method, it is possible to write:

$$x_k = x_{k-1} + \dot{x}_k \Delta t \qquad (1)$$

Considering the gyroscope measurements ω as inputs u (see Table 12) and using the relationship that exists between the derivative of the unit quaternion $\dot{q}$ and the angular velocity ω [44], it is possible to write:

$$\dot{x}_{k-1} = \frac{1}{2} Q(q_{k-1}) \begin{bmatrix} 0 \\ \omega_{k-1} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} q_{0_{k-1}} & -q_{1_{k-1}} - q_{2_{k-1}} & -q_{3_{k-1}} \\ q_{1_{k-1}} & q_{0_{k-1}} & -q_{3_{k-1}} & q_{2_{k-1}} \\ q_{2_{k-1}} & q_{3_{k-1}} & q_{0_{k-1}} & -q_{1_{k-1}} \\ q_{3_{k-1}} & -q_{2_{k-1}} & q_{1_{k-1}} & q_{0_{k-1}} \end{bmatrix} \begin{bmatrix} 0 \\ \omega_{x_{k-1}} - b\omega_{x_{k-1}} \\ \omega_{y_{k-1}} - b\omega_{y_{k-1}} \\ \omega_{z_{k-1}} - b\omega_{z_{k-1}} \end{bmatrix} \quad (2)$$

Substituting (2) in (1) it obtains:

$$x_k = \begin{bmatrix} q_{0_k} \\ q_{1_k} \\ q_{2_k} \\ q_{3_k} \\ b\omega_{x_k} \\ b\omega_{y_k} \\ b\omega_{z_k} \end{bmatrix} = \begin{bmatrix} q_{0_{k-1}} \\ q_{1_{k-1}} \\ q_{2_{k-1}} \\ q_{3_{k-1}} \\ b\omega_{x_{k-1}} \\ b\omega_{y_{k-1}} \\ b\omega_{z_{k-1}} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} q_{0_{k-1}} & -q_{1_{k-1}} - q_{2_{k-1}} & -q_{3_{k-1}} \\ q_{1_{k-1}} & q_{0_{k-1}} & -q_{3_{k-1}} & q_{2_{k-1}} \\ q_{2_{k-1}} & q_{3_{k-1}} & q_{0_{k-1}} & -q_{1_{k-1}} \\ q_{3_{k-1}} & -q_{2_{k-1}} & q_{1_{k-1}} & q_{0_{k-1}} \end{bmatrix} \begin{bmatrix} 0 \\ \omega_{x_{k-1}} - b\omega_{x_{k-1}} \\ \omega_{y_{k-1}} - b\omega_{y_{k-1}} \\ \omega_{z_{k-1}} - b\omega_{z_{k-1}} \end{bmatrix} \Delta t \quad (3)$$

The (3) in extended form becomes:

$$\begin{cases} q_{0_k} = q_{0_{k-1}} - \frac{1}{2}(\omega_{x_{k-1}} - b\omega_{x_{k-1}}) q_{1_{k-1}} \Delta t - \frac{1}{2}\left(\omega_{y_{k-1}} - b\omega_{y_{k-1}}\right) q_{2_{k-1}} \Delta t - \frac{1}{2}(\omega_{z_{k-1}} - b\omega_{z_{k-1}}) q_{3_{k-1}} \Delta t; \\ q_{1_k} = q_{1_{k-1}} + \frac{1}{2}(\omega_{x_{k-1}} - b\omega_{x_{k-1}}) q_{0_{k-1}} \Delta t - \frac{1}{2}\left(\omega_{y_{k-1}} - b\omega_{y_{k-1}}\right) q_{3_{k-1}} \Delta t + \frac{1}{2}(\omega_{z_{k-1}} - b\omega_{z_{k-1}}) q_{2_{k-1}} \Delta t; \\ q_{2_k} = q_{2_{k-1}} + \frac{1}{2}(\omega_{x_{k-1}} - b\omega_{x_{k-1}}) q_{3_{k-1}} \Delta t + \frac{1}{2}\left(\omega_{y_{k-1}} - b\omega_{y_{k-1}}\right) q_{0_{k-1}} \Delta t - \frac{1}{2}(\omega_{z_{k-1}} - b\omega_{z_{k-1}}) q_{1_{k-1}} \Delta t; \\ q_{3_k} = q_{3_{k-1}} - \frac{1}{2}(\omega_{x_{k-1}} - b\omega_{x_{k-1}}) q_{2_{k-1}} \Delta t + \frac{1}{2}\left(\omega_{y_{k-1}} - b\omega_{y_{k-1}}\right) q_{1_{k-1}} \Delta t + \frac{1}{2}(\omega_{z_{k-1}} - b\omega_{z_{k-1}}) q_{0_{k-1}} \Delta t; \\ b\omega_{x_k} = b\omega_{x_{k-1}}; \\ b\omega_{y_k} = b\omega_{y_{k-1}}; \\ b\omega_{z_k} = b\omega_{z_{k-1}}; \end{cases} \quad (4)$$

The equations (4) represent the prediction phase of the EKF algorithm (see Table 12).

In order to calculate the error covariance P_k^- (Table 12) it is necessary to compute:

- The Jacobian matrix A of partial derivatives of the state transition function with respect to x (state):

$$A_{[i,j]} = \frac{\partial f_{[i]}}{\partial x_{[j]}} (\hat{x}_k, u_k, 0) \quad \epsilon \, \mathbb{R}^{7x7}$$

- The Jacobian matrix W of partial derivatives of the state transition function with respect to w (noise) :

$$W_{[i,j]} = \frac{\partial f_{[i]}}{\partial w_{[j]}} (\hat{x}_k, u_k, 0) \, \epsilon \, \mathbb{R}^{7x3}$$

- The matrix Q of the variance of the gyroscope:

$$Q = diag(Variance\ GyrX, Variance\ GyrY, Variance\ GyrZ)\ \ \epsilon\ \mathbb{R}^{3x3}$$

Correction Phase

Once the prediction phase is calculated, it is necessary using the accelerometer and the magnetometer measures, to implement the correction phase. On the basis of the tests executed, the best performance is obtained splitting the correction phase in two independent steps. One step using only the accelerometers measurements and another one in which the correction is performed using only the magnetometers measurements. The measurement function that links the accelerometers measurements with the state vector is:

$$h_{acc} = R(q) * \overline{Acc} = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_2q_1 + q_0q_3) & q_0^1 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_3q_1 - q_0q_2) & 2(q_3q_2 + q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \begin{bmatrix} Acc_x \\ Acc_y \\ Acc_z \end{bmatrix}$$

In the same way, H_{mag} is obtained by the following matrix product:

$$h_{mag} = R(q) * \overline{Mag} = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_2q_1 + q_0q_3) & q_0^1 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_3q_1 - q_0q_2) & 2(q_3q_2 + q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \begin{bmatrix} Mag_x \\ Mag_y \\ Mag_z \end{bmatrix}$$

$\overline{Acc}$ and $\overline{Mag}$ represent the measures.

In order to calculate the Kalman gain K_k (Table 12), it is necessary to evaluate:

- the Jacobian matrix H of partial derivatives of the measurement function with respect to x (state):

$$H_{[i,j]} = \frac{\partial h_{[i]}}{\partial x_{[j]}}(\hat{x}_k, 0)\ \epsilon\ \mathbb{R}^{3x7}$$

- The Jacobian matrix V of partial derivatives of the measurement function with respect to v (noise measures):

$$V_{[i,j]} = \frac{\partial h_{[i]}}{\partial v_{[j]}}(\hat{x}_k, 0)\ \epsilon\ \mathbb{R}^{3x3}$$

- The matrix R of the variance of the sensor:

$$R = diag(VarianceX, VarianceY, VarianceZ) \quad \epsilon \mathbb{R}^{3x3}$$

Considering that the V is an identity matrix, the Kalman gain is the following:

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1}$$

II.7.2.3 EKF improvements

The EKF algorithm discussed in the previous chapters in theory should work perfectly. Unfortunately, in the real world some improvements are needed in order to obtain an IMU with a satisfactory performance.

Sensors Calibration

In order to convert the sensors raw data in calibrated data, the sensors datasheets give the conversion formulas. However, these equations suppose ideal conditions and don't take into account the following aspects:

- Process tolerances.
- Different sensitivity and bias for each axis.
- Misalignment between different sensors.
- Misalignment between the INEMO® and the IMU board.
- Non linearity.

Supposing the last problematic negligible, it is possible to overcome the other problems by means of the least square method.
As regards the gyroscope, the raw data and the angular velocity are related by the following relation:

$$\begin{bmatrix} R_x \\ R_y \\ R_z \end{bmatrix} = G\_m_{3x3} \begin{bmatrix} SC_x & 0 & 0 \\ 0 & SC_y & 0 \\ 0 & 0 & SC_z \end{bmatrix} * \begin{bmatrix} R'_x - R_{x0} \\ R'_y - R_{y0} \\ R'_z - R_{z0} \end{bmatrix}$$

$$= \begin{bmatrix} G_{11} & G_{12} & G_{13} \\ G_{21} & G_{22} & G_{23} \\ G_{31} & G_{32} & G_{33} \end{bmatrix} * \begin{bmatrix} R'_x - G_{10} \\ R'_y - G_{20} \\ R'_z - G_{30} \end{bmatrix}$$

Where *Rx, Ry, Rz* are the final angular velocities of each axis, *G_m* is the misalignment matrix between the gyro sensing axes and the device body axes, *SCx, SCy, SCz* are the scale factors caused by the mismatch of the sensitivity of each axis, *Rx', Ry', Rz'* are the raw measurements of the gyroscopes and *Rx0, Ry0, Rz0* are the biases for each axis. The least square method can calculate the twelve parameters (G_{11}…G_{33}, G_{10}, G_{20} and G_{30}) for a complete calibration procedure. To apply this method, the IMU board was mounted over a KUKA robotic manipulator [67]. First of all, the bias of each axis is measured (G_{10}, G_{20}, G_{30}). Then, the KUKA manipulator rotates the board around each axis at different known angular velocity and collects the measurements (Y). The matrix of the known angular velocities is related to the gyroscope raw measurements matrix (w) by the unknown matrix X:

$$Y = w \cdot X$$

Finally, to determine the other nine parameters (G_{11}…G_{33}) the least square method is applied:

$$X = [w^T w]^{-1} w^T Y$$

For the accelerometer, the procedure is the same, using also in this case the KUKA manipulator [67] in static poses. On the contrary, the magnetometer calibration, even if use the same equation, cannot be executed by means the KUKA manipulator because the metal parts of the robot affect the magnetometer measurements.

The accelerometer and gyroscope calibration is executed only once, because their parameters are practically time independent and don't depend by the environment. On the contrary, the magnetometer calibration has to be executed every time the environment changes.

Chebyshev Filtering

During the test phase, the greatest problem was represented by the vibration introduced by the brushless engine to the whole airframe. In particular, the accelerometer measurements were completely distorted, causing a malfunction of the EKF algorithm. In Figure 42 a comparison between the raw data of the Z component, when the motor is off (red) and when the motor is running (blue) is

shown. It is worth to point out that, in both cases the airframe is locked to the workbench.

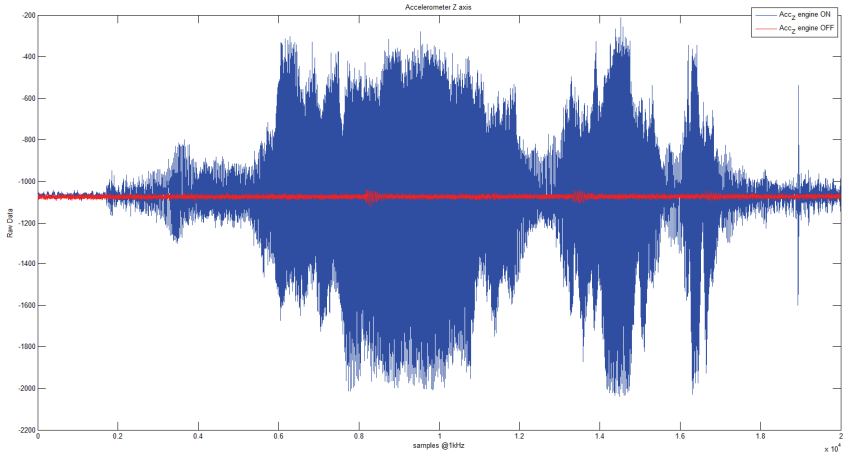

Figure 42 - Noise introduced by brushless engine – Time domain

To solve this problem, the first step is an analysis of the vibration in the frequency domain. Once the noise region has been identified, a filtering is performed. In order to catch the widest range of frequencies, the Output Data Rate (ODR) of the accelerometer has been set to 1344Hz, whereas the sample rate is 1kHz. In Figure 43 the same comparison is shown into the frequency domain. It is clear as the main harmonic corresponds to approximately 110Hz. Therefore, to filter out the vibration effect, accelerometer data were filtered with a low pass fourth-order Chebyshev filter, with a cutoff frequency of 25Hz. The results are shown in Figure 44. In contrast to the previous test, in this case the airframe is moving in order to ensure that the Chebyshev filtering does not filter part of the dynamics of the aircraft. Moreover, in addition to the comparison between the filtered (black) and unfiltered data (green), a comparison is executed with a commercial IMU (red), in particular an MTi by Xsens [54].

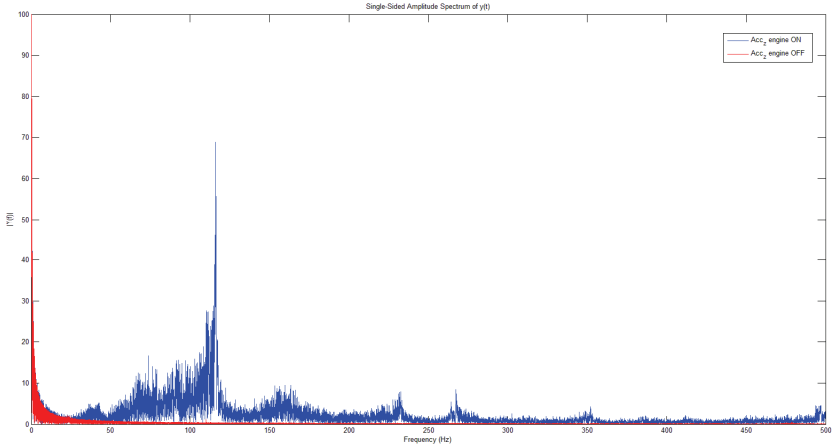

Figure 43 - Noise introduced by brushless engine – Frequency domain

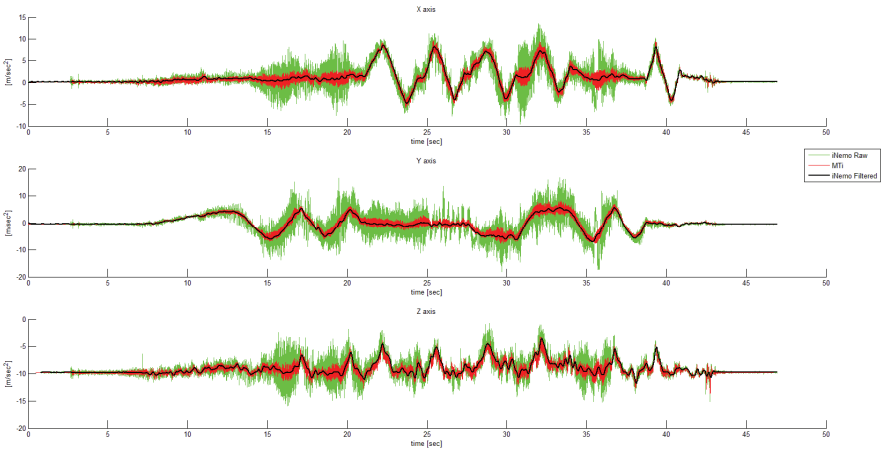

Figure 44 - Results of accelerometer filtering

Dynamic compensation

Another aspect to be taken into account that leads to a deterioration of the EKF performance, is represented by the dynamic component of the accelerometer measures. Going into detail, the EKF algorithm implemented in the IMU works properly only when the system rotates and does not translate, or at least it translates slowly. Obviously, this is not the case of a drone. During a mission, the accelerometer measures both the gravity vector and the dynamic

accelerations of the aircraft, but only the former is useful for the EKF algorithm. Considering that to identify separately the static and dynamic component is not possible without other devices, the only way to compensate the dynamic acceleration is represented by decreasing the reliability of the accelerometer data, i.e. to act in the correction phase of the EKF algorithm. More precisely, the computation of the Kalman gain in the accelerometer correction phase is done in the following way:

$$K_k = P_K H^T (H P_K H^T + R)^{-1}$$

The *R* matrix, is a diagonal matrix whose elements are the variances of the *X,Y* and *Z* axis of the accelerometer.

$$R = \begin{bmatrix} var\_AccX & 0 & 0 \\ 0 & var\_AccY & 0 \\ 0 & 0 & var\_AccZ \end{bmatrix}$$

The lower is the value of such variance, the more reliable are the measurements. In order to relate the *R* matrix to the dynamic acceleration, the following modification is made to the matrix:

$$R_{dyn} = \begin{bmatrix} var\_AccX & 0 & 0 \\ 0 & var\_AccY & 0 \\ 0 & 0 & var\_AccZ \end{bmatrix} \cdot Dyn\_comp$$

Where *Dyn_comp* is a variable related to the dynamic acceleration, as it is shown in the following formula:

$$Dyn_{comp} = sat_{0.1}\left\{\left|\frac{norm(AccX, AccY, AccZ)}{9.81} - 1\right|\right\}$$

In absence of dynamic accelerations, *Dyn_comp* is zero, converesely the value of *Dyn_comp* (saturated to 0.1) increases the unreliability of the accelerometer.

Firmware Development and Optimization

A big challenge in the IMU firmware development resides in maximizing the frequency at which the Extended Kalman Filter works. In fact, the IMU could be designed to provide the feedback of the low level control, in order to ensure the stability of the aircraft (see II.3.1). An IMU that works at higher frequency implies a system faster to compensate the disturbances, and therefore more suitable to operate in unstructured environments. However, the EKF is a very complex algorithm, and implementing it at high frequency inside a 32bit microcontroller, represents a hard challenge. A solution to this problem consists in optimizing the code, replacing the libraries for the matrix calculation with normal sums of products. These libraries, even if make the code more readable, on the other hand are not optimized. To better clarify the concept, consider the following matrix product:

$$A_{7x7} \cdot P_{7x7} \cdot A_{7x7}^{T}$$

To solve this product 686 multiplications and 588 sums are needed. However, if one considers the terms of the A matrix:

$$A = \begin{bmatrix} 1 & -\frac{1}{2}Gxdt & -\frac{1}{2}Gydt & -\frac{1}{2}Gzdt & q1dt & q2dt & q3dt \\ \frac{1}{2}Gxdt & 1 & \frac{1}{2}Gzdt & -\frac{1}{2}Gydt & -q0dt & q3dt & -q2dt \\ \frac{1}{2}Gydt & -\frac{1}{2}Gzdt & 1 & \frac{1}{2}Gxdt & -q3dt & -q0dt & q1dt \\ \frac{1}{2}Gzdt & \frac{1}{2}Gydt & \frac{1}{2}Gydt & 1 & q2dt & -q1dt & -q0dt \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$where\ Gidt = Gyroscope_i - Bias_i$$

It can be seen that there are only seven different elements, and there are seven ones and eighteen zeros. Then, in the matrix product $A \cdot P \cdot A^T$,whatever is the P matrix, there are many null terms, whereas the others are composed in part by the

seven elements of the *A* matrix. By using the symbolic calculation toolbox of Matlab, it is possible to quantify the level of optimization obtained:

```
symsp11p12p13p14p15p16p17p21p22p23p24p25p26p27real
symsp31p32p33p34p35p36p37p41p42p43p44p45p46p47real
symsp51p52p53p54p55p56p57p61p62p63p64p65p66p67real
symsp71p72p73p74p75p76p77real

P=[p11 p12 p13 p14 p15 p16 p17; p21 p22 p23 p24 p25 p26 p27;
   p31 p32 p33 p34 p35 p36 p37; p41 p42 p43 p44 p45 p46 p47;
   p51 p52 p53 p54 p55 p56 p57; p61 p62 p63 p64 p65 p66 p67;
   p71 p72 p73 p74 p75 p76 p77];

syms Gxdt Gydt Gzdt S0dt S1dt S2dt S3dt real

A=[1 -Gxdt -Gydt -Gzdt S1dt S2dt S3dt;
Gxdt 1 Gzdt -Gydt -S0dt S3dt -S2dt;
Gydt -Gzdt 1 Gxdt -S3dt -S0dt S1dt;
GzdtGydt -Gxdt 1 S2dt -S1dt -S0dt;
0 0 0 0 1 0 0;
0 0 0 0 0 1 0;
0 0 0 0 0 0 1];

Pnew=A*P*A';
```

Exploiting the symmetries and the occurrences of the terms present in the *Pnew* matrices, it turns out that it is possible to calculate it with about 300 multiplication and 700 sums. Considering that the products require more time compared to the sums to be computed and that in this way no *for* cycle is needed, it is possible to reduce the computation time to about 4-5 times. Summarizing, using this strategy the maximum frequency to the EKF algorithm increases from 50Hz (using the matrix libraries) to 250Hz (Figure 46).

II.7.2.4 Firmware Block Scheme

The EKF timing is managed by an interrupt connected to a timer whose frequency is 500Hz. At each interrupt the accelerometer data are acquired, calibrated and filtered. The acquisition and calibration of the gyroscope and the magnetometer is executed every two interrupts, i.e. every 4ms (Figure 45). The yellow wave in Figure 46 indicates the time necessary to execute the interrupt. the frequency is in fact 500Hz and it can be seen that there are positive half-waves longer (where overall sensors are acquired), alternating with positive half-waves shorter (where only accelerometer is acquired). The reason to acquire the accelerometer at 500Hz is because in this way a better filtering can be

accomplished, with respect to a sampling frequency of 250Hz. Once all sensors data are acquired and calibrated, a flag is set and the EKF algorithm is executed in the main loop. According to the tests, once executed the prediction phase (green wave of Figure 46), the best performance is obtained executing the correction phase (blue wave of Figure 46) one time with the accelerometer measurements and one time with the magnetometer measurements. The last step is represented by a rotation of the resulting quaternion, if a ROS or a RHS request is received (see Appendix B), and their conversion in RPY angles.

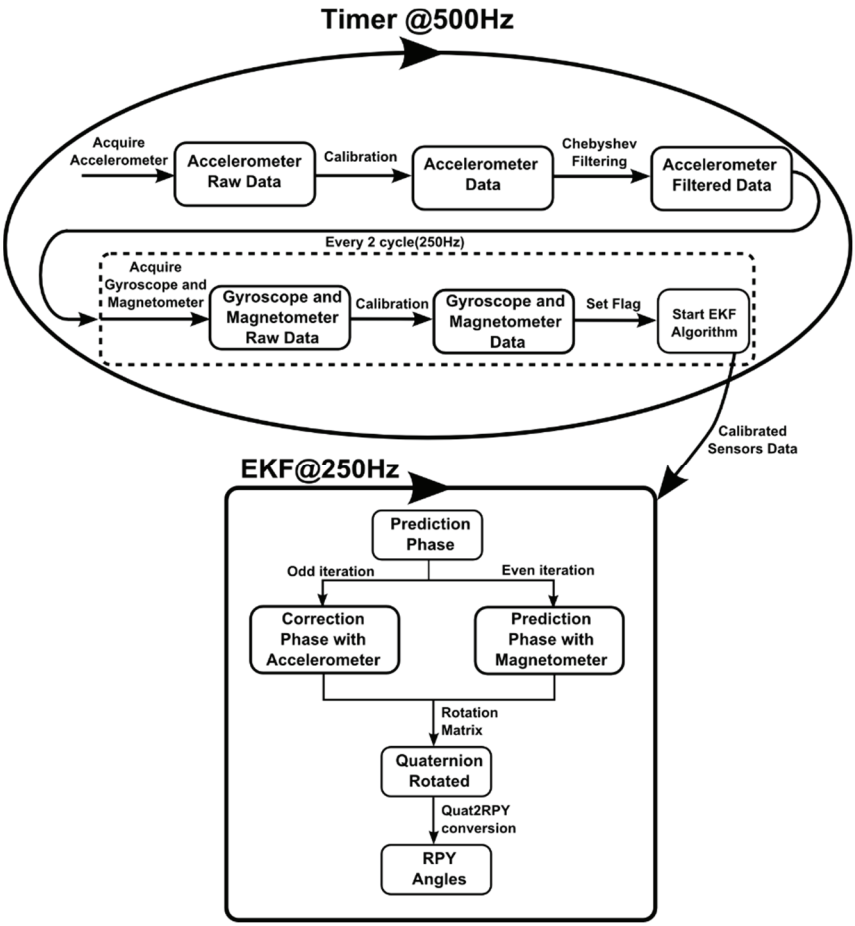

Figure 45 - EKF block scheme

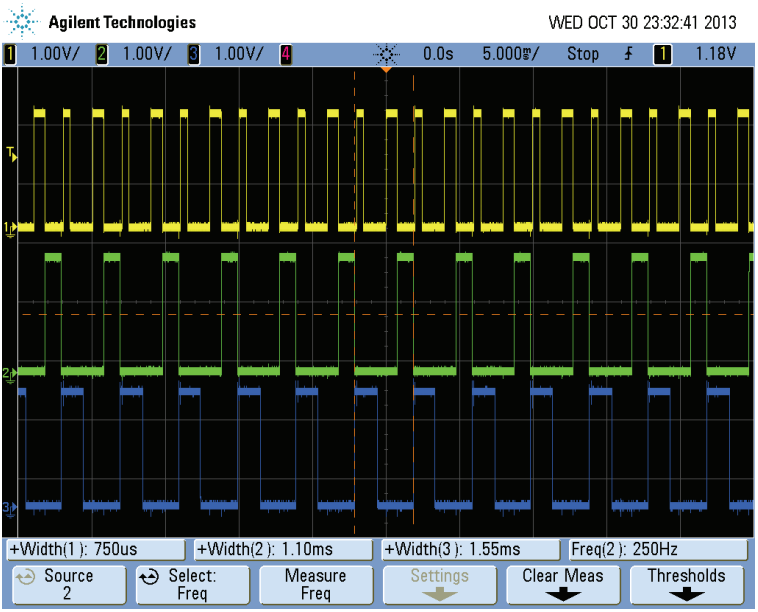

Figure 46 - EKF timestamp

II.7.3 HMI development

In the development and test phases the HMIs have a key role, because they make these processes faster than the classic debug firmware of a microcontroller. In particular, during the IMU design, two HMIs have been developed, both of them in LabView [48].

II.7.3.1 Kalman HMI

This HMI has been developed in order to verify and optimize the EKF algorithm. As discussed in the section II.7.2.3, the EKF algorithm and especially its optimization process, requires a lot of time and attention. Even a single wrong sign can lead the algorithm to diverge. It is really important in this phase, in order to recognize a possible error, to monitor each intermediate calculation and each term of the various matrices. It is obvious that if the EKF algorithm is running in the microcontroller, it is practically impossible to control every matrix terms by a classical debug. Moreover, every FW change requires a compiling and a programming operation. Conversely, with a LabView HMI it is possible monitoring easily each term and executing changes on the fly in the EKF

algorithm. Moreover, in this way a comparison with the MTi by Xsens can be done. In the Figure 47 a block scheme of the Kalman HMI is shown, whereas the Figure 48 shows a screenshot.

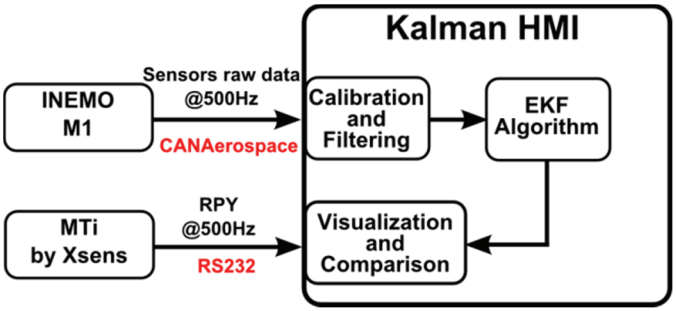

Figure 47 - Kalman HMI block scheme

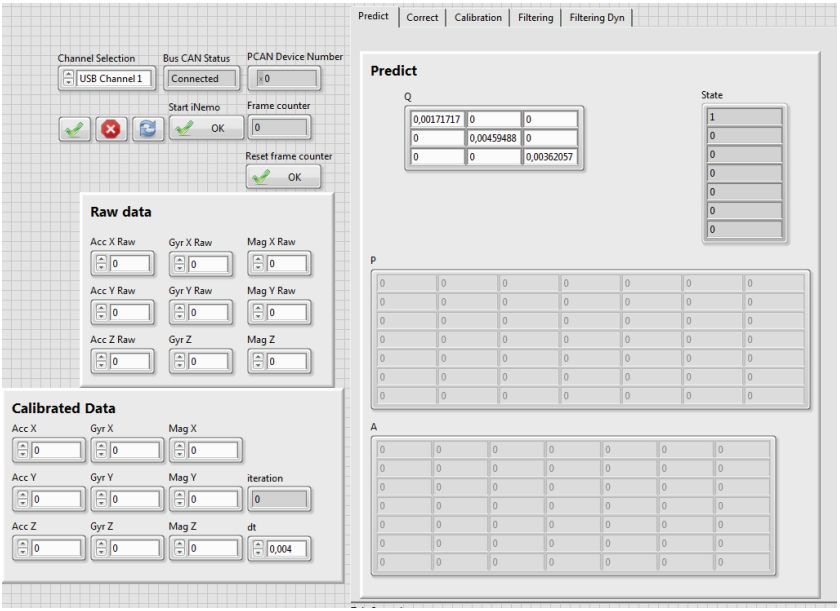

Figure 48 - HMI Kalman, EKF development

II.7.3.2 INEMO® M1 HMI

Once the testing and the development phases were completed, a new HMI has been designed in order to exploit the potentialities of the IMU (Figure 49).

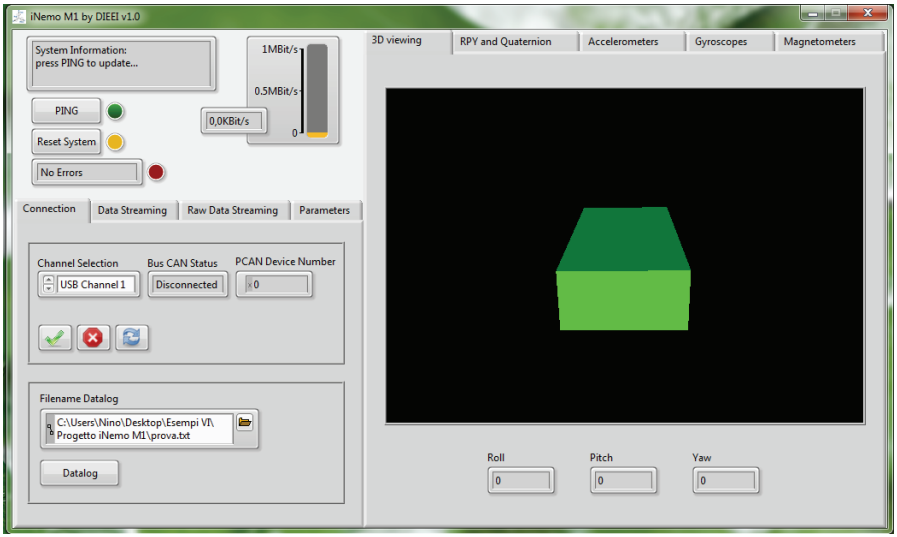

Figure 49 - INEMO® M1 HMI

The key features of the INEMO® M1 HMI are the following:

- complete management of the CANAerospace protocol (Appendix B)
- Selection of the data to send (Figure 50).
- Possibility to change on the fly whatever parameter or reset the attitude and the heading reference (Figure 51).
- Online plotting of any sensors data (Figure 52).
- Online magnetometer calibration (Figure 53). In contrast with the gyroscope and the accelerometer, whose calibration can be made only once, the magnetometer measurements are affected by many factors, such as batteries, metal parts and so on. For this reason, a magnetometer calibration is needed every time the environment in which the IMU works is modified.

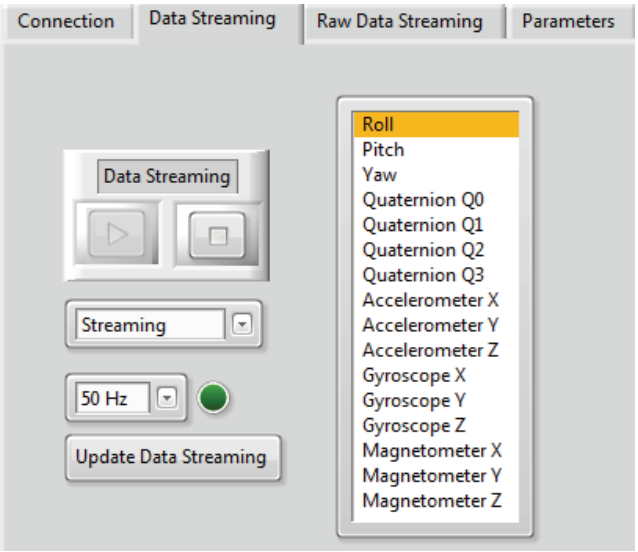

Figure 50 - INEMO® M1 HMI, data selection

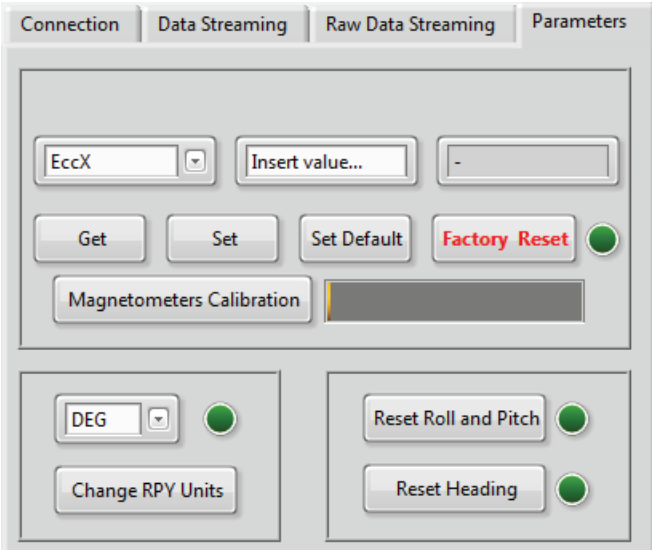

Figure 51 - INEMO® M1 HMI, parameters management

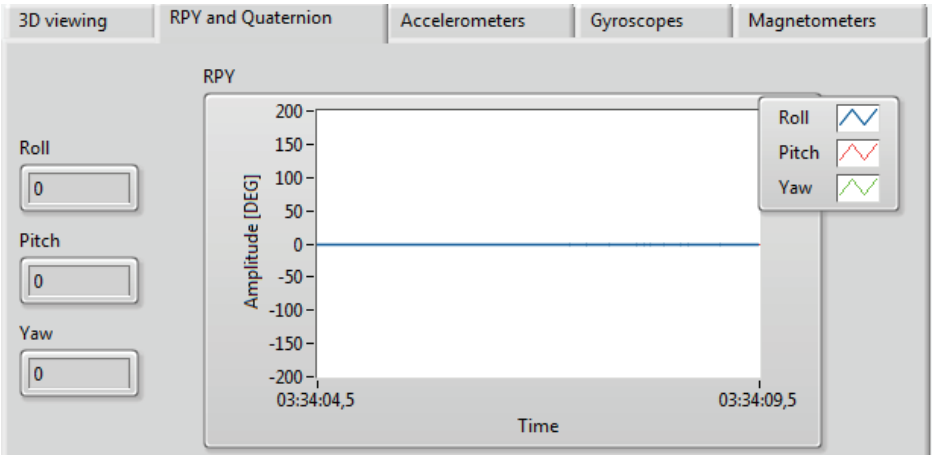

Figure 52 - INEMO® M1 HMI, sensorsplotting

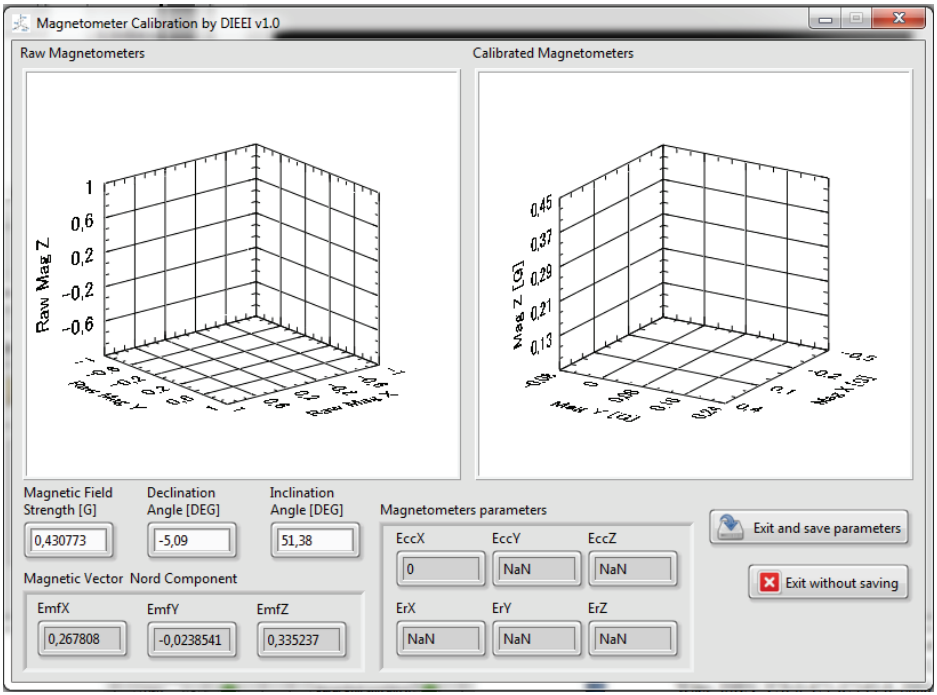

Figure 53 - INEMO® M1 HMI, online magnetometer calibration

II.7.4 Results

In this paragraph a comparison between the developed IMU board and a commercial one is treated. The IMU taken as reference is the MTi produced by Xsens [54]. The features of both devices are summarized in Table 13:

Sensors	IMU boardwith INEMO M1	Xsens MTi
Gyroscopes	3-axis digital gyroscope, ±250°/s, ±500°/s, ±2000°/s full scales, 0.03 deg/s/√Hz Noise, 100 Hz Bandwidth 760 Hz max update rate	3-axis gyroscope, ±300°/s Full Scale, 0.05 deg/s/√Hz Noise, 40 Hz Bandwidth, 512 Hz max update rate
Accelerometers	3-axis digital accelerometer, ±2g, ±4g, ±8g, ±16g full scales, 220 ug/√Hz, 149,3 Hz Bandwidth, 1.344 kHz max update rate	3-axis accelerometer, ±50 m/s^2 Full Scale, 0.002 m/s^2/√Hz, 30 Hz Bandwidth, 512 Hz max update rate
Magnetometers	3-axis digital magnetometer, from ±1.3 gauss to ±8.1 gauss, 0.05 mGauss, 220 Hz max update rate	3-axis accelerometer, ±750 mGauss Full Scale, 0.05 mGauss, 10 Hz Bandwidth, 512 Hz max update rate
Maximum update rate processing	250 Hz	512 Hz

Table 13 - INEMO® M1 board vs Xsens MTi

In order to execute the comparison, the two inertial platform boards were aligned and fixed in a rigid support, as shown in the Figure 54. Both slow and fast dynamics were performed. The Figure 55, Figure 56 and Figure 57 show as the two IMU boards have a comparable behaviour.

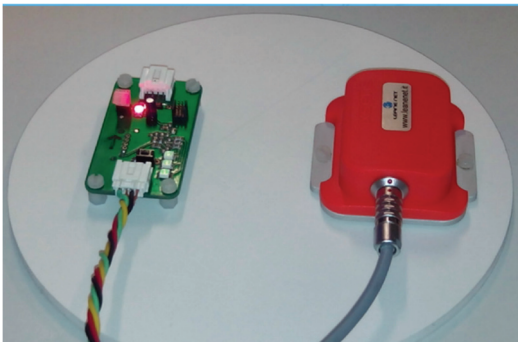

Figure 54 - Rigid support to compare the two devices

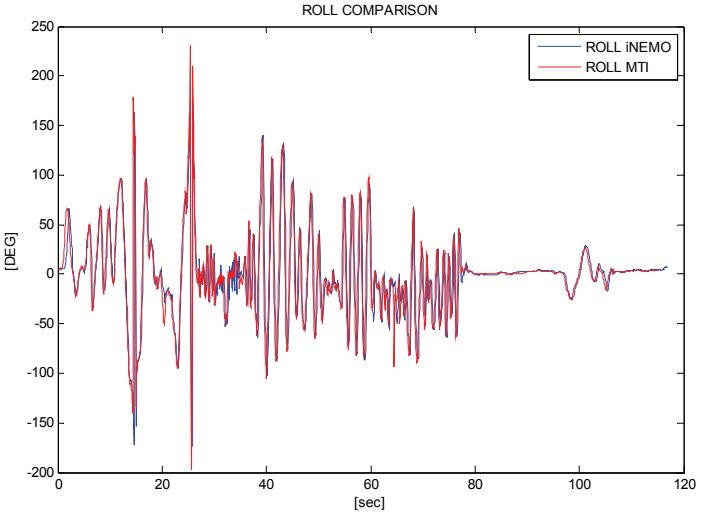

Figure 55 - Roll comparison

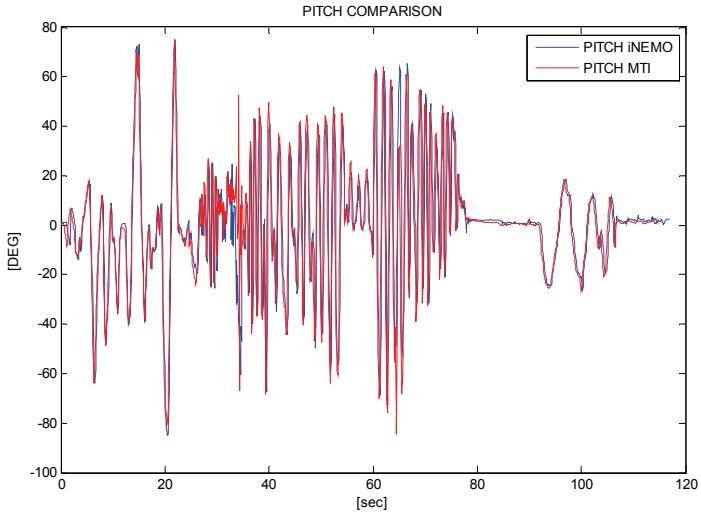

Figure 56 - Pitch comparison

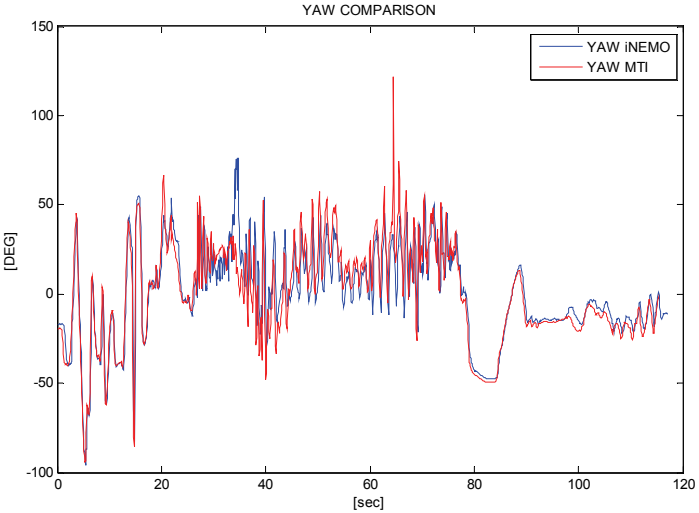

Figure 57 - Yaw comparison

Chapter III. The Asctec Hummingbird

III.1 Introduction

In this section the other aerial platform is treated, the Hummingbird quadrotor produced by Asctec [60]. As discussed in the I.1.2.2 paragraph, quadrotors and in general the multirotors have the advantage that they can be controlled only by varying the speed of the propellers and thus fixed-pitch blades, in contrast to helicopters, can be used. This aspect implies a simplification in the design and in the control of the drones. Moreover, the use of four rotors allows each individual rotor to have a smaller diameter, compared to a helicopter with the same size, producing them to store less kinetic energy during the flight. In order to compensate gyroscopic effect and aerodynamic torques, the front and the rear propellers rotate counter-clockwise, while the others rotate clockwise (Figure 58).

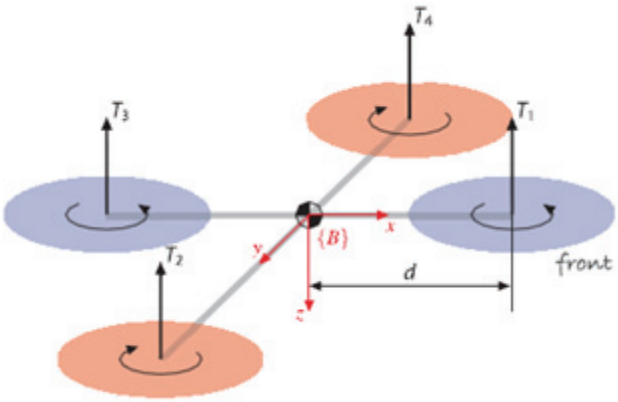

Figure 58 - Rotation of the quadrotor propellers

III.1.1 Quadrotor Movements

The total thrust generated by each motor is given by [45]:

$$T_i = C_T \rho A_{ri} r_i^2 \omega_i^2$$

Where, for any rotor i, A_{ri} is the rotor disk area, r_i is the radius of the propeller, ω_i is the angular velocity, C_T is a thrust coefficient depending of propeller geometry and ρ is the air density. In practice, a simple lumped parameter model like the following is used:

$$T_i = \widetilde{C_T} \omega_i^2$$

In this equation, $\widetilde{C_T} > 0$ represents a constant determined from static thrust test, in order to include also the effects of the drag on the airframe induced by the rotor flow. Consequently, the total thrust at hover applied to the airframe can be easily calculated as the sum of the thrusts from each individual rotor:

$$T = \sum_{i=1}^{n} |T_i| = \widetilde{C_T} \sum_{i=1}^{n} \omega_i^2$$

The system is underactuated, and the remaining degrees of freedom (DoF) corresponding to the translational velocity in the x-y plane must be controlled through the system dynamics. In particular, in order to accomplish a given movement, each rotor has to modify its thrust in the following way:

- **Ascend**: each rotor increases its angular velocity.
- **Descend**: each rotor decreases its angular velocity.

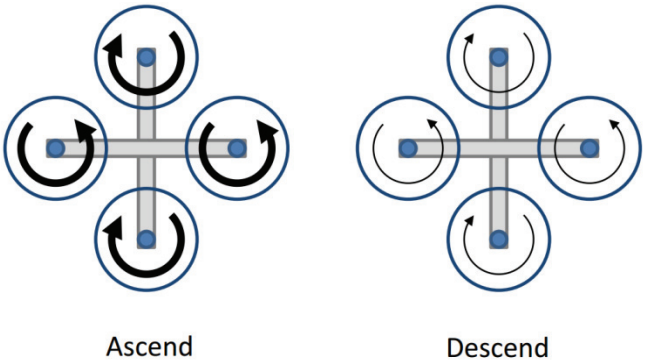

Figure 59 - Representation of angular velocities during ascending and descending phases

- **Turn Left**: front and rear rotors increase their angular velocity while the others two maintain unchanged their angular velocity.
- **Turn Right**: left and right rotors increase their speed while the others two maintain unchanged their speed.

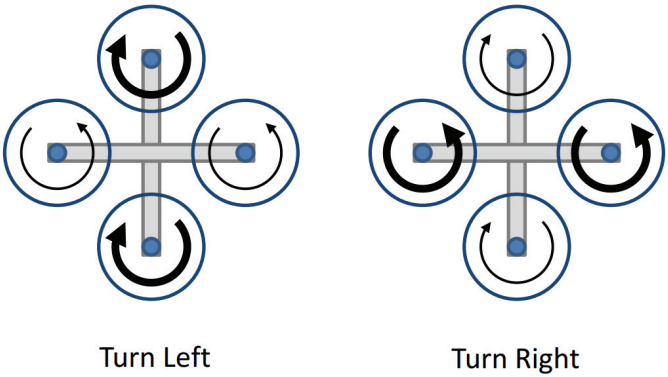

Figure 60 - Representation of angular velocities during the yaw motion

- **Move forward**: front rotor decreases its angular speed and rear rotor increases it. The others two don't change their angular speed.
- **Move backward**: front rotor increases its angular speed and rear rotor decreases it. The others two don't change their angular speed.

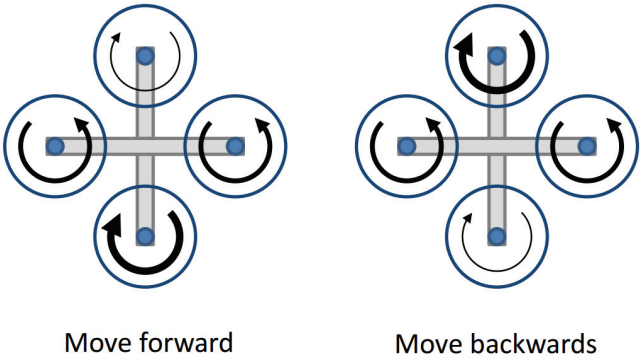

Figure 61 - Angular speeds during the move forward and backward phases

- **Move right**: right rotor decreases its angular speed and left rotor increases it. The other two rotors maintain a constant angular speed.

- **Move left**: right rotor increases its angular speed and left rotor decreases it. The other two rotors maintain a constant angular speed.

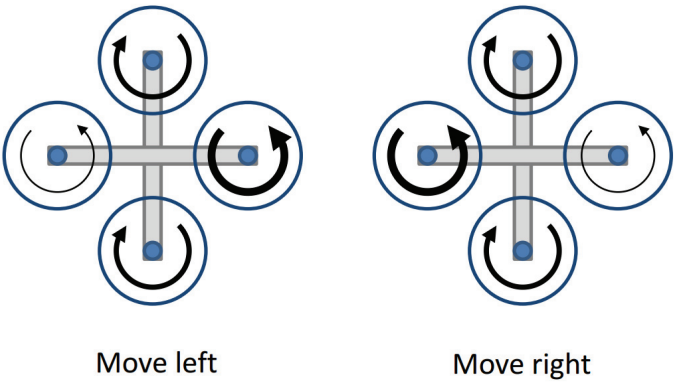

Figure 62 - Angular speeds during the move right and left phases

III.2 The Hardware

The Hummingird used is equipped with:

- A three-axial IMU for attitude control.
- A GPS module for outdoor navigation.
- A barometric sensor for altitude measure.

In Table14 its main features are summarized, whereas in Figure 63 a sketch of the quadrotor is shown.

Size	540 x 540 x 85,5 mm
Max. take off weight	0,71 kg
Max. payload	200 g
Flight time incl. payload	20 min.
Range	4,500 m ASL, 1,000 m AGL
Max. airspeed	15 m/s
Max. climb rate	5 m/s
Max. thrust	20 N
Wireless communication	2,4 GHz XBee link, 10–63 mW
Intertial guidance system	AscTecAutoPilot with 1,000 Hz update rate
Flight modes	GPS Mode, Height Mode, Manual Mode

Table14 – Asctec Hummingbird features

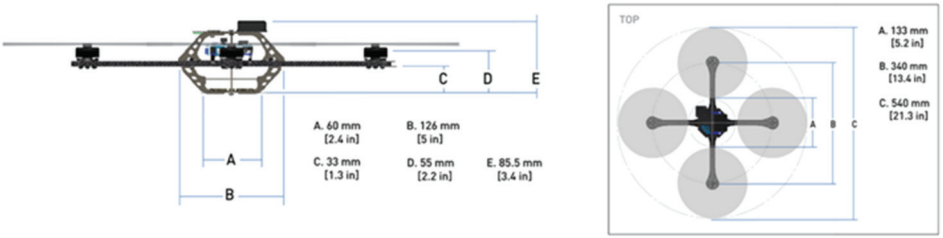

Figure 63 - Sketch of Hummingbird Asctec

As regards the autopilot, the Hummingbird adopts a different approach with respect to the Volcan UAV (II.3). Indeed, the Hummingbird autopilot is constituted by two ARM7 microcontrollers, a low level processor (LL) and a high level processor (HL), in addition to several communication interfaces such as UART, SPI and I2C. The tasks of the LL processor are the management of the sensor data processing, data fusion, sending commands to the motor controller, and above all the implementation of the basic attitude control in order to ensure the stability of the system. The HL processor manages the GPS as well as it is

responsible for the high level control algorithms, such as navigation through waypoints. A key difference between the two microcontrollers is that the code of the LL processor is not accessible and not editable, whereas in the HL processor there is the possibility to implement user-defined code, in order to add sensors, to accomplish a custom task and so on. To confirm this, the attitude control cannot be disabled or bypassed, and it is always running in the three flight modes in which the Hummingbird operates:

- GPS Mode (attitude, height and position control activated)
- Height Mode (attitude and height control activated)
- Manual Mode (attitude control activated)

As it is shown in Figure 64, the two processors communicate with a data rate of 1kHz.

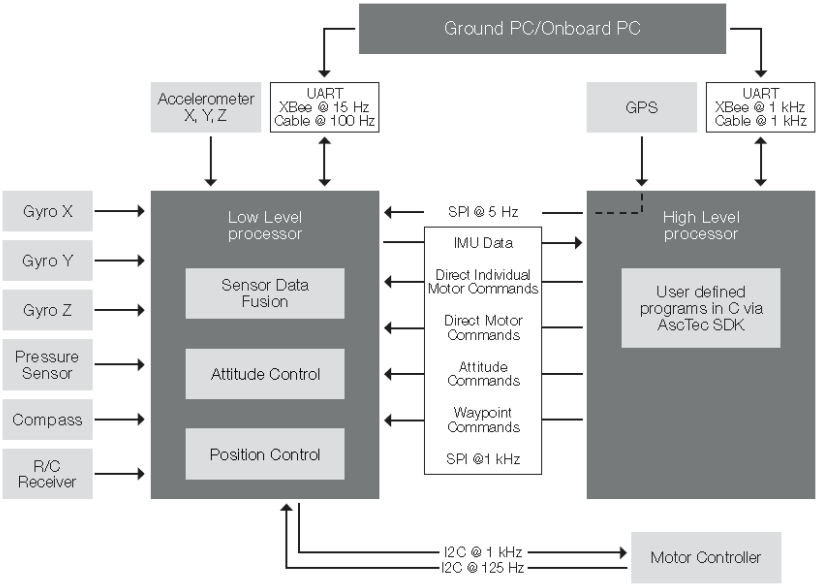

Figure 64 - Block scheme Asctec autopilot

III.3 The Software

The software platform is composed by three different blocks:

- A development environment to build the code inside the HL processor.
- A graphical interface connected to the LL processor, in order to receive and display the telemetry.
- A communication protocol, necessary to interface the quadrotor with a PC or another device.

III.3.1 AscTec SDK

The AscTec HL SDK is a C-code framework in an Eclipse environment with cross-compiler and debugger. It can be used as a starting point to program different algorithms, sensor interfaces and communication protocols in C-Code. It contains the basic configuration and the predefined control algorithms to use immediately the quadrotor, furthermore it has a particular .c file (sdk.c) which is triggered at 1 kHz, where new control strategies can be implemented.

III.3.2 AscTec AutoPilot Control

The HMI provided by AscTec (Figure 65) is connected to the LL processor via an Xbee wireless link and it is used to execute basic operations, such as the navigation through waypoints using a static map, motor setup, parameters setting, sensors calibration and telemetry. However, with this HMI it is not possible adding new features, such as to manage other sensors. Moreover, with this HMI the HL processor is not used. This means that is not possible, among other things, to use the HL processor to store a flight plan. In fact, to execute a navigation task it is necessary that the drone and the remote PC are connected. This represents a heavy limit to the drone potential, since the flight plan area have to be smaller than the wireless module range.

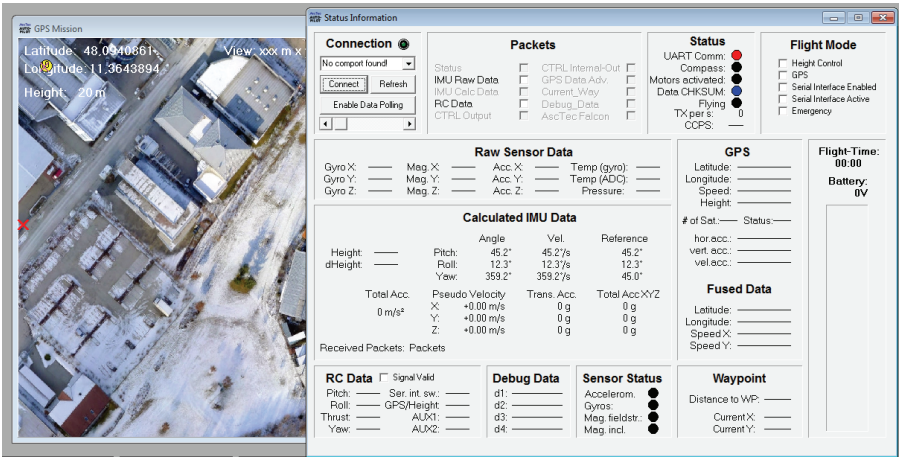

Figure 65 - Screenshoot of AscTecAutoPilot

III.3.3 ACI Protocol

The AscTec Communication Interface (ACI) [46] is a communication protocol developed by AscTec, in order to connect their UAVs and a user local machine (remote software). It is designed for requesting variables, sending commands and setting parameters. It is possible to create own packages to send or receive data. The advantage of this method is that it is possible to choose which variables shall be received, at which transmitting rate from the device, and which commands and parameters is possible to send to or to set on the device. There are two different modules for the AscTec Communication Interface (Figure 66):

• ACI Device in C: a simple module for the UAV, where it's possible to set easily, which variables, commands and parameters are available on the device and which of them the local machine can choose.

• ACI Remote in C: a small module written in C to get all variables, commands and parameters of the device. It only uses standard libraries and works on every operating system.

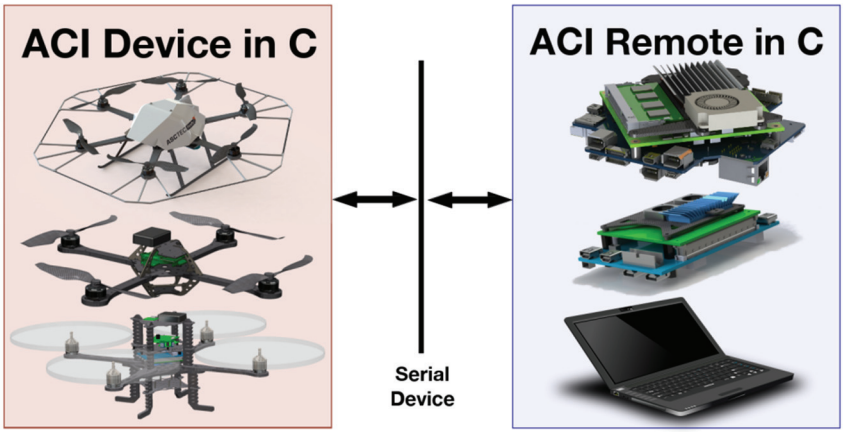

Figure 66 - ACI protocol modules

III.4 Library development for ACI remote

In the section III.3.2 it's clear to understand that the HMI provided to Asctec is not suitable to develop complex tasks, not to mention the possibility to manage third part devices connected, for example, to an I2C port of the HL processor. In a few words, a more powerful HMI is needed. A first step in this direction is represented by the ACI protocol, discussed in the previous part, constituted by a set of C-language files. However, if the ACI device packet (onboard the drone) it is well suited to be used in the HL processor, which is generally programmed with a C language IDE (Eclipse), on the other hand the C files of the ACI remote packet are not suitable to be used in those IDEs with a high level graphical language. For this reason a DLL library file has been developed, in order to make the interface between the drone and a graphical IDE, such as LabView, simpler. Moreover, in addition to ACI remote standard functions [46], several functions have been added in order to simplify the connections and the data exchange between the drone and the HMI.

III.4.1 Connection initialization

init_ACI

The first function in this category is used to initialize the ACI protocol in the HMI and to initialize all the internal variables and flags. The prototype of this function is the following:

```
void init_ACI(void);
```

getSystemInfo

This function is called when the operator needs information such as software version and max number of variable packets. The prototype is:

```
void getSystemInfo(SystemInfo *info);
```

Where *SystemInfo* is the following struct:

```
typedefstruct
   {
   unsigned char verMajor;
       unsigned char verMinor;
       unsigned char maxNameLength;
       unsigned char maxDescLength;
       unsigned char maxUnitLength;
       unsigned char maxVarPackets;
       unsigned char memPacketMaxVars;
       unsigned short flags;
       unsigned short dummy[8];
}SystemInfo;
```

III.4.2 Variables management

In the ACI protocol the variables are the read only data. The whole list is present in [47]. The variables transmitted are managed by means of packets. Every packet can include a max number of variables and it is possible to assign a different data rate to each packet. Even if theoretically there is no a max number of packets, the online documentation recommends to use a max number of three packets, in order to avoid overloading of the transmission channel. Every variable is identified by an ID, a short description and a data type. The following struct shows the variable modeling.

```
   typedef struct{
    unsigned short id;
    unsigned char varType;
      char value_int8;
      unsigned char value_u_int8;
      short value_int16;
      unsigned short value_u_int16;
      int value_int32;
       unsigned int value_u_int32;
      float value_float;
   }Packet;
```

This struct is used also to model the commands and the parameters.

varListUpdateFinished

Before using the variable packets, it is necessary to synchronize the HMI receiving their complete list. After the calling of the function *aciGetDeviceVariablesList* (see [46]), the ACI protocol executes the following function, which sets a flag indicating that the variables list has been updated:

```
void varListUpdateFinished(void);
```

var_packetX_management with X=1,2,3

At the beginning of the paragraph it was explained that a variables packet can include a different number of variables (up to twenty). Moreover, there are different types of variables, such as 8bit, 16bit, 32bit integer or float. In order to maximize the flexibility of the code, the generic variables packet is modeled as it follows:

- A transmission rate.
- A packet size, i.e. the number of variables included in the packet.
- An array of variable types, which identifies the type of each variable.
- An array of ID, which identify each variable.
- An array of *packet* struct, in which the variables are stored. This array is a global variable, so does not appear in the function prototype.

Then, the prototype of the function that generates the packet is the following:

```
void var_packetX_management(unsigned short transmission_rate,
unsigned char packet_size,unsigned char vartype_array[],
unsigned short ID_array[]);
```

For the sake of clarity, consider a variables packet example with the following items:

- *motor_rpm[1]*, ID 0x0100, type UINT8
- *GPS_latitude*, ID 0x0106, type INT32
- *battery_voltage*, ID 0x0003, type INT16

Graphically, the DLL creates the packet as it is shown in Figure 67:

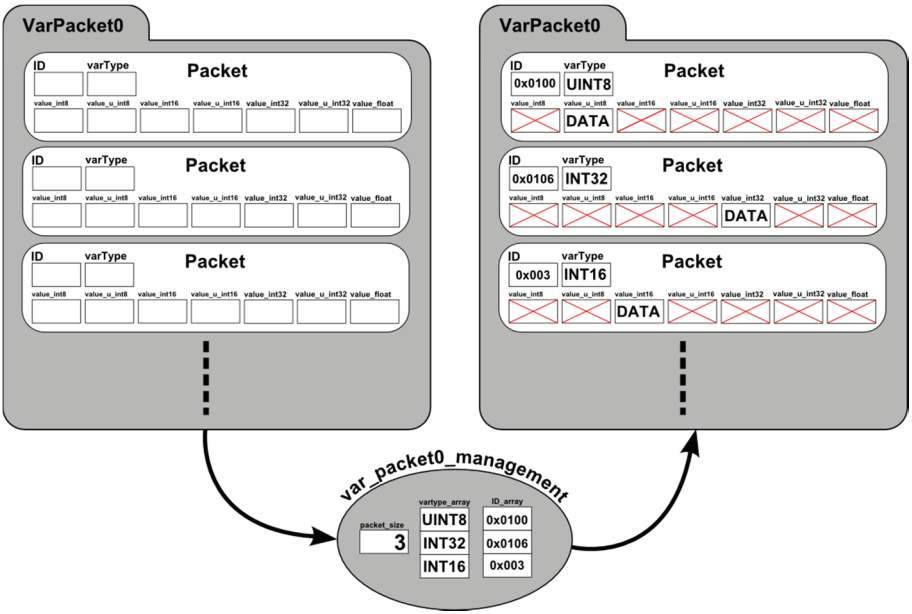

Figure 67 - var packet management

get_variables_packedX with X=1,2,3

This function is used to send sensors data to the HMI. Also in this case there is the problem that a packet can contain a different number of variables, of different types. A solution to this problem is to use a 32bit array (passed as pointer) to send the variables, and to use the function *memcpy* to convert the 32bit data in the correct format. The information about the correct datatype of the variable can be taken by the *Vartype_array* used in the previous function. The prototype of the function is the following:

```
void get_variables_packedX(unsigned int *variables);
```

Continuing the previous example, the DLL creates the array, as shown in Figure 68:

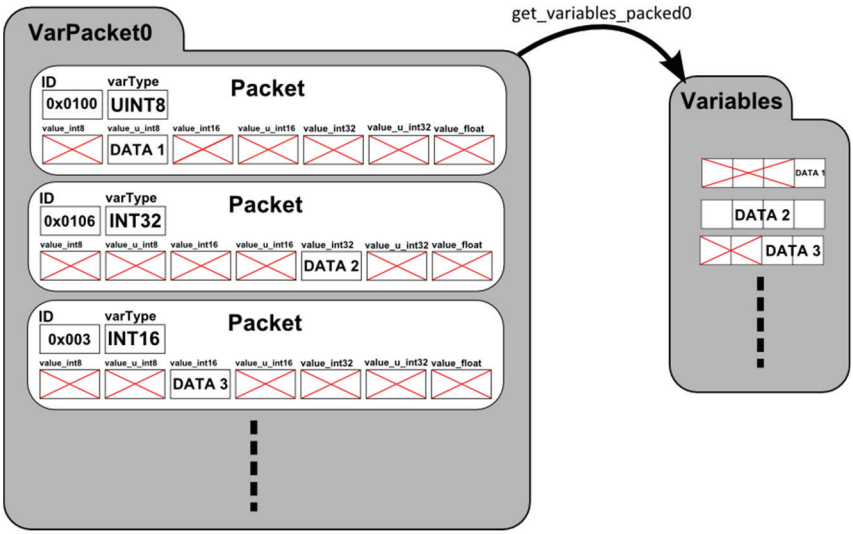

Figure 68 get variable packet

Now the variables, contained in the packed0, can be used to the HMI.

reset_var_packedX with X=1,2,3

This function simply flushes the variables packet. Its prototype is the following:

```
void reset_var_packedX(void);
```

III.4.3 Commands management

In the ACI protocol the commands are the write only data. The whole list is presented in [47]. The organization of the commands packets is the same of the variables, so similar functions have been implemented. The only difference is that instead of the function *GETvariable*, there is a *SETcommand* function. They are:

```
void cmdListUpdateFinished(void);

voidcom_packetX_management(unsigned short transmission_rate,
unsigned char packet_size,unsigned char cmdtype_array[],
unsigned short ID_array[]);
```

```
voidset_commands_packedX(unsigned int *variables);

void reset_com_packedX(void);
```

III.4.4 Parameters management

In the ACI protocol the parameters are both read and write data. The whole list is presented in [47]. The parameters packet functions are the same of the commands packet functions. They are:

```
voidparamListUpdateFinished(void);

voidpar_packetX_management(unsigned short transmission_rate,
unsigned char packet_size,unsigned charpartype_array[],
unsigned short ID_array[]);

voidset_parameters_packedX(unsigned int *variables);

void reset_par_packedX(void);
```

Chapter IV.
The Multiplatform Drone HMI

IV.1 Introduction

A disadvantage of using a heterogeneous fleet of UAVs is represented by the different HMIs necessary to supervise them. In order to implement complex cooperation tasks, a data exchange between the different HMIs is mandatory. A smart solution to overcome this drawback is represented by the development of a multiplatform HMI capable to manage different type of UAV; in this case of study, this HMI, developed in LabView, has the following features:

- Possibility to manage, monitor and supervise different robotic platforms by a single remote station.
- Compliant with CANAerospace and ACI protocol.
- Online Telemetry and datalog for post processing.
- On line mapping by means principal providers such as GoogleMap, BingMap and OpenStreetMap.
- Complete managing of waypoint lists and routes.

LabView [48] is a graphical programming language that uses icons instead of text to create applications. In contrast with the conventional textual programming languages, in which the instructions determine program execution, LabView uses the programming based on the data flow, i.e. the flow of data determines the program execution. Any LabView application is made of two parts (Figure 69):

- The Front Panel: in this form are present buttons, knobs, text boxes and so on. It is the effective interface with which the operator interacts.
- The Block Diagram: very similar to a block scheme, in this window there are the relations and the links between the various elements of the LabView Front Panel.

Summarizing, generally a block in LabView has a double representation: one for the operator (in the front panel) and one for the application developer (in the diagram panel).

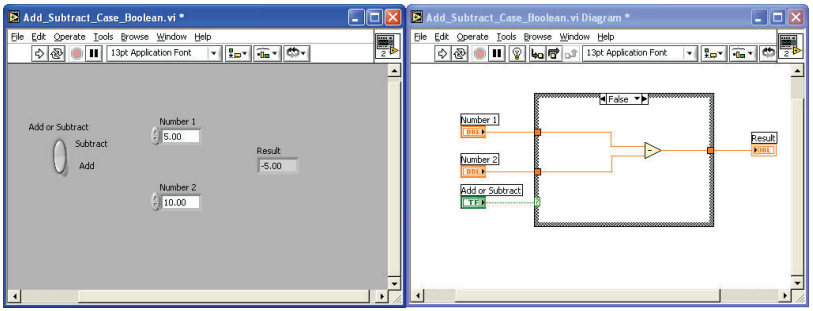

Figure 69–LabView: Front Panel and Block Diagram

An application developed in LabView is named *VI*, i.e. Virtual Instrument. A given *VI* has a set of input and output parameters, like a conventional C function. Moreover, it is possible to use a *VI* within another LabView application. In this case, often the name *subVI* is used.

The following two sections are so organized: the first one discusses about the various classes of *subVI* developed, the second one explains the HMI developed for the Volcan and the Asctec Hummingbird.

IV.2 LabView subVIs

The *subVIs* treated in this chapter, have been implementes in order to develop LabView applications with specific features, such as CANbus connection, ACI connection and online mapping.

IV.2.1 CANbus subVIs

The CANbus connectivity between the bus and the PC is realized by means of the PCAN-USB converter by Peak-system [55] (Figure 70). Even if the Peak-system releases a set of LabView *subVIs* [56], further *subVIs* based on this suite have been developed in order to simplify the CANbus management.

Figure 70 - PCAN-USB

IV.2.1.1 PCAN connection.vi

This *subVI* manages the connection and the disconnection to the CANbus. The prototype is shown in Figure 71, whereas the front panel and the block diagram of a typical application is shown in Figure 72.

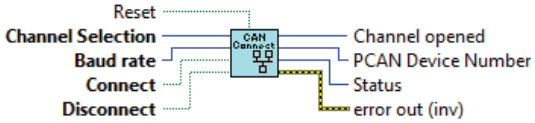

Figure 71 - PCAN connection.vi

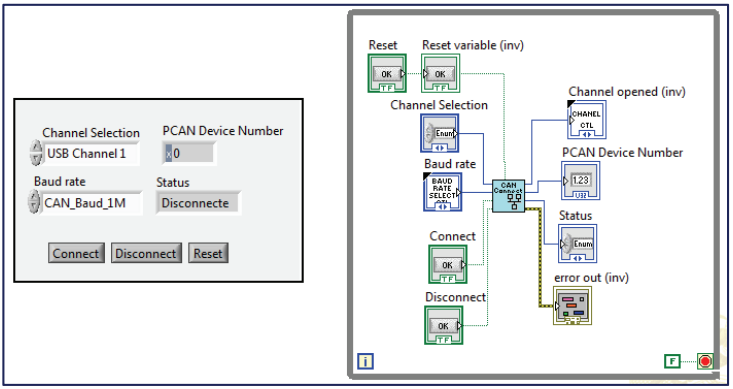

Figure 72 - PCAN connection example

IV.2.1.2 PCAN receive.vi

Once the PC is connected to the CANbus, the use of this *VI* makes it possible the reception of the CANbus frames. The prototype is shown in Figure 73, whereas the front panel and the block diagram of a typical application are shown in Figure 74.

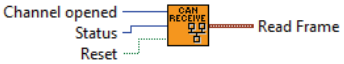

Figure 73 - PCAN receive.vi

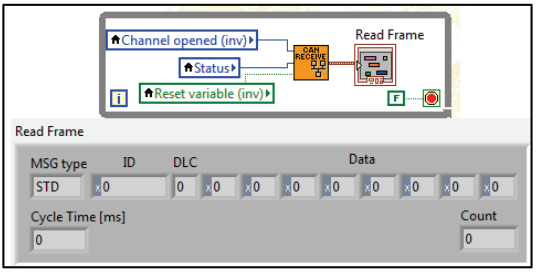

Figure 74 - PCAN receive example

The output *Read Frame* is a *cluster*, i.e. the equivalent of a *struct* in C-language, that models a frame received from a CANbus. As it is shown in Figure 74, it has the following fields:

- The enum *MSG type*, that identifies the different types of frames, such as standard, extended, remote request and so on.
- The ID field.
- The DLC field.
- Eight byte used as data field.
- The *Cycle time* field, expressed in *ms*, that indicates the time elapsed between the current frame and the previous one.
- The *Count* field, that counts the total number of frames received.

IV.2.1.3 PCAN Send.vi

This *subVI* is the dual of the previous one. It manages the transmission of CANbus frames. The prototype and a typical application are shown in the following figures. The input *cluster Send Frame* has the same fields of the *Read Frame* explained previously.

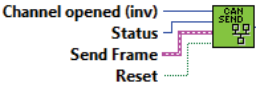

Figure 75 - PCAN send.vi

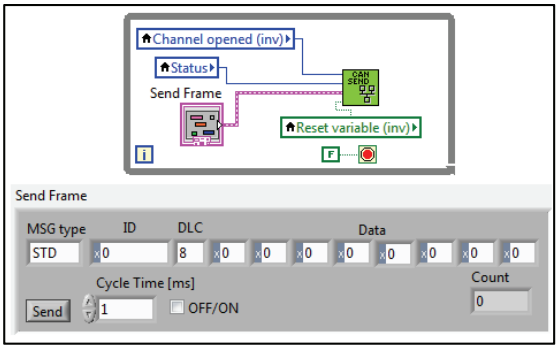

Figure 76 - PCAN Send example

IV.2.2 FTDI subVIs

To connect the Asctec Hummingbird to the PC, an USB-RS232 converter or an Xbee device are needed. Both of them are based on chip produced by FTDI [57], which converts USB protocol in RS232 protocol. This company releases a full set of LabView *subVI* (available at [58]), that allows a simple interface with its devices, without any further *subVI*.

IV.2.3 ACI protocol subVIs

In LabView there is the possibility to import a DLL file in order to create a *VI* from each function within it. This tool is the "Import Shared Library". In this way, the functions managing the ACI protocol, treated in the section III.4 can be used in LabView by means of a set of *subVIs*. The prototypes of the *subVI* created in this way, are exactly the same of their C-language counterpart (Figure 77), therefore to explain them again would be an unnecessary duplication.

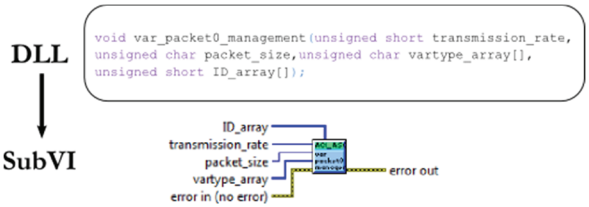

Figure 77 - Example of matching between C-function and *subVI*

In addition to the DLL-imported *subVI*, a couple of polymorphic *subVI* have been developed, *ALL_2_UINT32.vi* and *UINT32_2_ALL.vi* (Figure 78), in order to make easier the use of the functions such as *get_variables_packedX* (III.4.2) or *set_commands_packedX* (III.4.3). These *subVIs* fit, respectively, any numeric type variable in a memory location suitable to store a 32bit unsigned integer variable and viceversa. In a few words, is similar to a *memcpy* instruction in C-code.

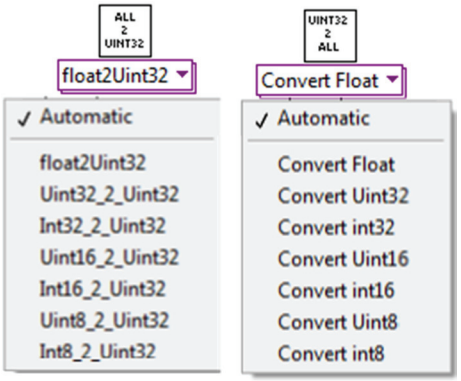

Figure 78 - Polymorphic subVIs

IV.2.4 Datalog subVIs

A key feature of the HMI developed, regards the possibility to execute a log of the telemetry, choosing the variables to be logged, with a given sample time. In order to maximize the flexibility and the exportation simplicity, the following *subVIs* have been developed.

IV.2.4.1 Create Header Datalog.vi

This *subVI* modifies dynamically the header of the datalog, according to the input parameter "*Select Variable*". In Figure 79 an example regarding a VOLCAN datalog is shown.

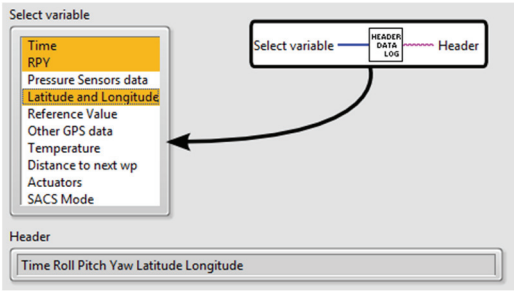

Figure 79 - Create Header Datalog Example

IV.2.4.2 Record Data.vi

In a similar way to the previous *subVI*, this one updates the string variable *Datalog OUT*. To make this, at each execution this *subVI* adds a row to the *Datalog IN* variable, inserting those variables selected in the *Select variable* input parameter.

In Figure 80 the prototype for the Volcan datalog is shown. The clusters *Sensor IN*, *Actuators IN* and *Assisted mode ref. value* are shown in the Figure 102 in the next chapter.

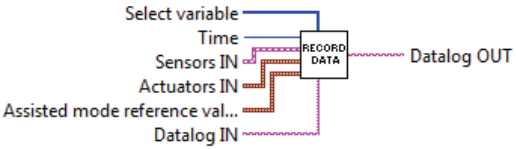

Figure 80 - Record Data.vi

IV.2.5 Instruments subVIs

Even if LabView owns a lot of graphic indicators, a customized set of flight instruments has been developed, exploiting the connectivity between LabView and .NET code. The basic idea is the design of a custom flight instrument, composed by the following parts:

- A background image.
- An image which represents the needle
- A numeric input which indicates the needle rotation over the background image.

To create a background image in LabView, simply create a *picturebox* and set its property *background image*, indicating the path of the chosen image (Figure 81).

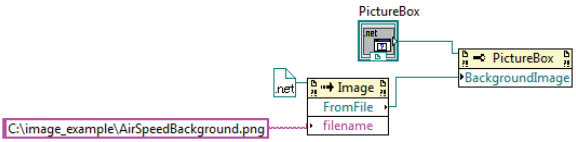

Figure 81 - Picturebox background image.

As concern the needle, the procedure is the same, but in this case the picturebox property is simply *image*. At last, the rotation of the needle requires a specific DLL file, whose source files in C# code is available at [49]. From this DLL a *subVI* has been created, *Rotate image.vi*. In Figure 82 its prototype is shown, whereas the Figure 83 shows an example of a flight instrument.

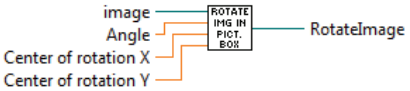

Figure 82 - Prototype of "Rotate image.vi"

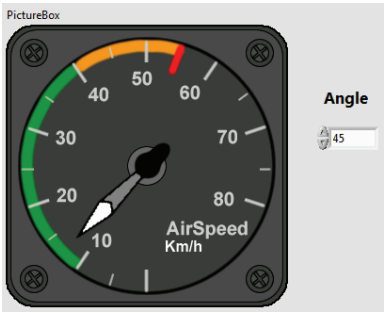

Figure 83 - Example of a flight instruments

IV.2.6 Mapping subVIs

The biggest limit of the Asctec HMI, discussed in the section III.3.2, lies in the fact that a static map is used. This means that, before to begin a new mission a georeferenced image of the location has to be loaded. To overcome this limitation, the LabView HMI developed use a set of *subVIs* based on the DLL files of the GMap.NET open source platform [50].

IV.2.6.1 Map provider.vi

This *subVI* binds a given map provider, selectable by means of an ENUM control, to an instance of the .NET class `GMap.NET.MapProviders`. The options are summarized in Table 15

Provider	.NET class
BingMap Hibrid	GMap.NET.MapProviders.BingHybridMapProvider
BingMap Satellite	GMap.NET.MapProviders.BingSatelliteMapProvider
BingMap	GMap.NET.MapProviders.BingMapProvider
GoogleMap	GMap.NET.MapProviders.GoogleMapProvider
OpenStreetMap	GMap.NET.MapProviders.OpenStreetMapProvider
YahooMap Hybrid	GMap.NET.MapProviders.YahooHybridMapProvider
YahooMap Satellite	GMap.NET.MapProviders.YahooSatelliteMapProvider
YahooMap	GMap.NET.MapProviders.YahooMapProvider

Table 15 - Map provider options

Figure 84 - Map provider.vi prototype

IV.2.6.2 Init GmapControl.vi

The aim of this *subVI* is to initialize the *GMap control*, i.e. the portion of the screen in the LabView application where the dynamic map will be shown. The editable properties are:

- The map provider
- The GMap mode: only server, server and cache, only cache.

The other properties, fixed by default, are summarized in Table 16:

GMap property	value
Dimension	600x1100 pixels
Drag button	With mouse left button
Min Zoom Value	1
Max Zoom Value	22
Cache Capacity	250MB

Table 16 - GMap Control default properties

As shown in Figure 85, this *subVI* returns the Gmap *Overlays*, a set of layers in which graphic elements can be inserted.

Figure 85– InitGmapControl.vi prototype

IV.2.6.3 Init Gmap Overlays.vi

The overlays initialization is accomplished by means of this *subVI*. Generally, for any robotic platform, three layers are used: a layer to draw the waypoints, a second layer to draw the route and a last one to draw polygons.

Figure 86 - Init Gmap Overlays.vi prototype

IV.2.6.4 Current coordinates.vi

This *subVI* is used to obtain the geographic coordinates pointed to the mouse cursor, when it is over the map. In addition to the *GMap control* and the mouse position, the *subVI* needs the position of the *GMap control* respect to the LabView application (Left and Top input in Figure 87). The outputs are the mouse geographic coordinates, together with the limit coordinates of the displayed map.

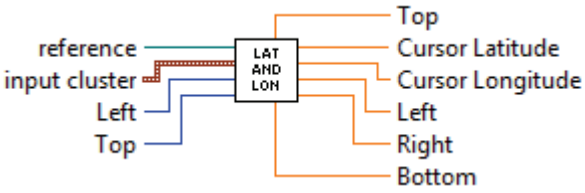

Figure 87 - Current coordinates.vi prototype

IV.2.6.5 LAT LON 2 pixel.vi

This *subVI* is the dual of the previous one. For a given geographic coordinates, the *subVI* returns the corresponding pixel coordinates (top and left) in the *GMap control*. As will be discussed in the next paragraph, such *subVI* is used to place an image in the map.

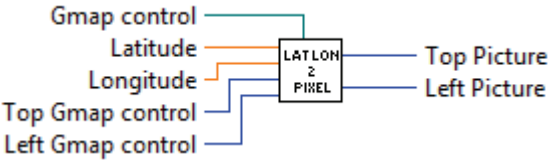

Figure 88 - LAT LON 2 pixel.vi prototype

IV.2.6.6 Show picture on map.vi

The following *subVI* is used to show an image over a *GMap control*, in a given position. In Figure 89 the prototype is shown.

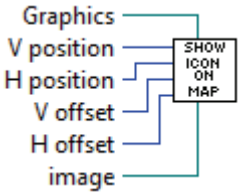

Figure 89 - Show picture on map.vi prototype

To make this, the following steps have to be accomplished:

- Create a .NET reference of the class `System.Drawing.Bitmap`, using the chosen image.

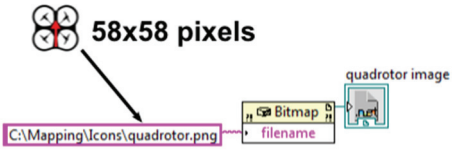

Figure 90 - create the image reference

- Create a .NET reference of the class `System.Drawing.Graphics`, used as input in the *subVI* (Figure 89).

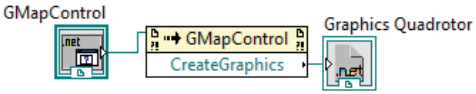

Figure 91 - create the Graphics reference

- Since the center of the image is out of phase for 29 pixels along x and y, to show it over the map correctly, it is necessary to draw the block diagram in Figure 92. The resulting front panel is shown in Figure 93.

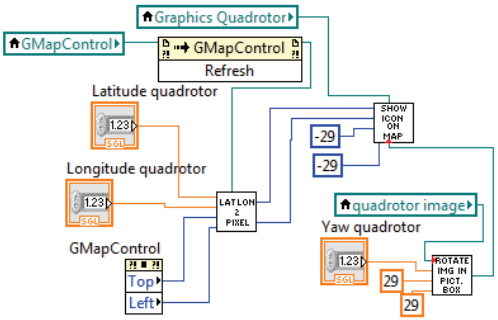

Figure 92 - example of block diagram with *Show picture on map.vi*

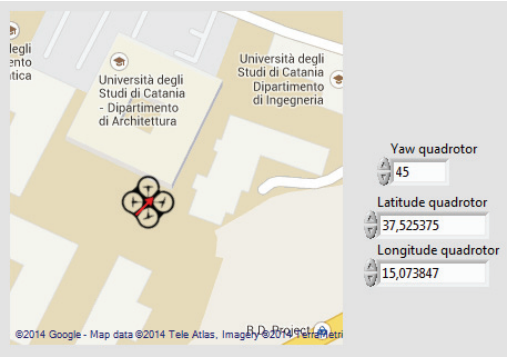

Figure 93 - Example of Front panel with *Show picture on map.vi*

IV.2.6.7 Show Waypoint.vi

The task of the *subVI* developed is to display a given set of waypoint over the *GMap control*. Its prototype is shown in Figure 94. The *Route IN* input is an array of cluster which models a waypoint (Figure 95).

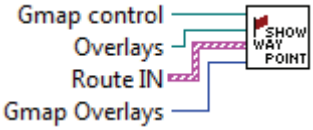

Figure 94 - Show Waypoint.vi prototype

Figure 95 - waypoints example

IV.2.6.8 Save and Load Waypoint List.vi

This pair of files has been developed to save and load a set of waypoints. *Save Waypoint List* creates a WPL file (WayPoint List), where the waypoints are stored. Viceversa, *Load Waypoint List* displays on the map the waypoints previously saved in a WPL file, with the possibility to append them in a preexisting list.

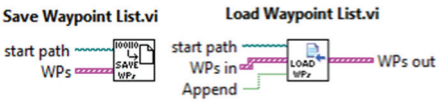

Figure 96 - subVI to save and load WPL files

IV.2.6.9 Update Route.vi

The route of the drone is modeled by a simple matrix, in which in its columns latitude, longitude and time are stored. The matrix updating is executed only when the drone covers a distance greater than the chosen threshold.

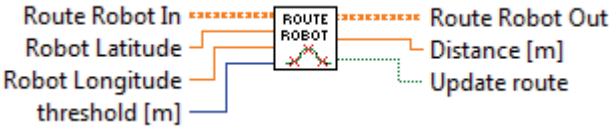

Figure 97 - Update route.vi prototype

IV.2.6.10 Show Route.vi

This *subVI* has been developed in order to make the drawing of a route over a *GMap control* simpler. In Figure 98 the prototype is shown.

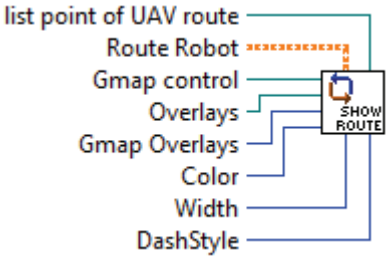

Figure 98 - Update Route prototype

The input *list point of UAV route* is a reference of the .NET class `System.Collections.Generic.List'1`.

IV.2.6.11 Save and Load Route.vi

This pair of files has been developed to save and load an UAV route. *Save route* creates a ROR file (Route Of Robot), where the route is stored. Viceversa, *Load Route* displays on the map the route previously saved in a ROR file.

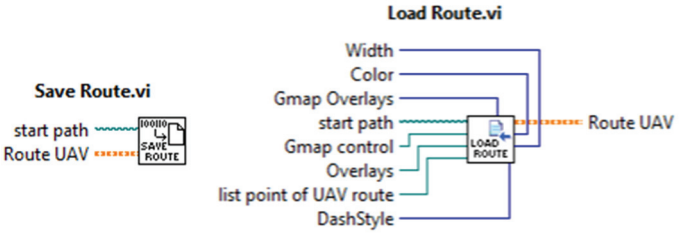

Figure 99 - subVI to save and load ROR files

IV.3 LabView HMI

The libraries introduced in the previous section represent the bricks necessary to build the final HMI treated in this one.

IV.3.1 Different drones, one HMI

Exploiting the *show icon on map.vi* (IV.2.6.6), it is possible to display different robotic platforms: just choose an icon and assign it a couple of geographic coordinates. In Figure 100, the latitude and longitude from the CANAerospace are assigned to the icon of the Volcan, whereas the coordinates from ACI protocol are assigned to the icon of the Hummingbird.

Figure 100 - Example of icons on map

IV.3.2 CANbus and ACI connection

As concern the connection to the drones, the Figure 101 shows the forms dedicated to the connection with the CANbus (left side) and the ACI protocol (right side).

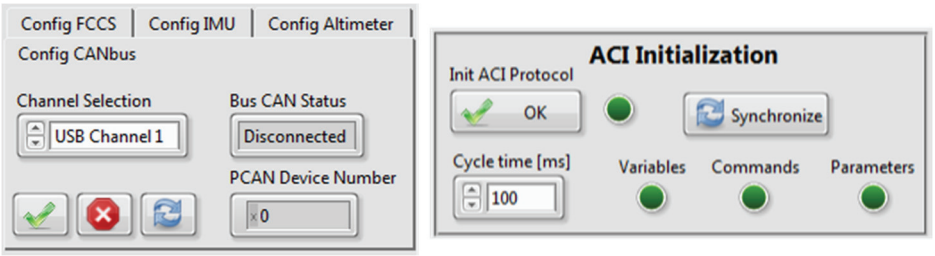

Figure 101 - Drones connection

IV.3.3 Telemetry and Datalog

Once the ACI connection is established, it is possible to choose which, variables, commands and parameters manage. In Figure 103 the forms for variables, commands and parameters management are shown. As regards the Volcan UAV, an important part coincides with the management of the control loop, in order to execute an optimal tuning (II.3.4). In Figure 102 the form dedicated for the telemetry and the tuning of the PID gains in shown.

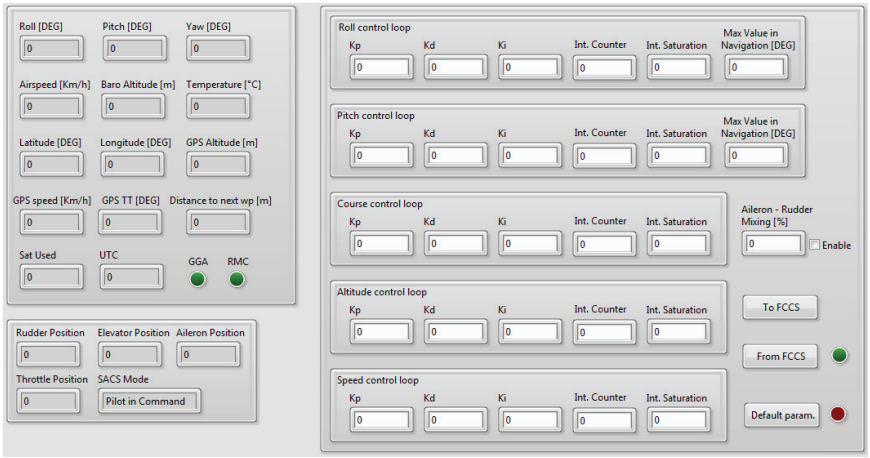

Figure 102 - Telemetry and PID tuning on the Volcan

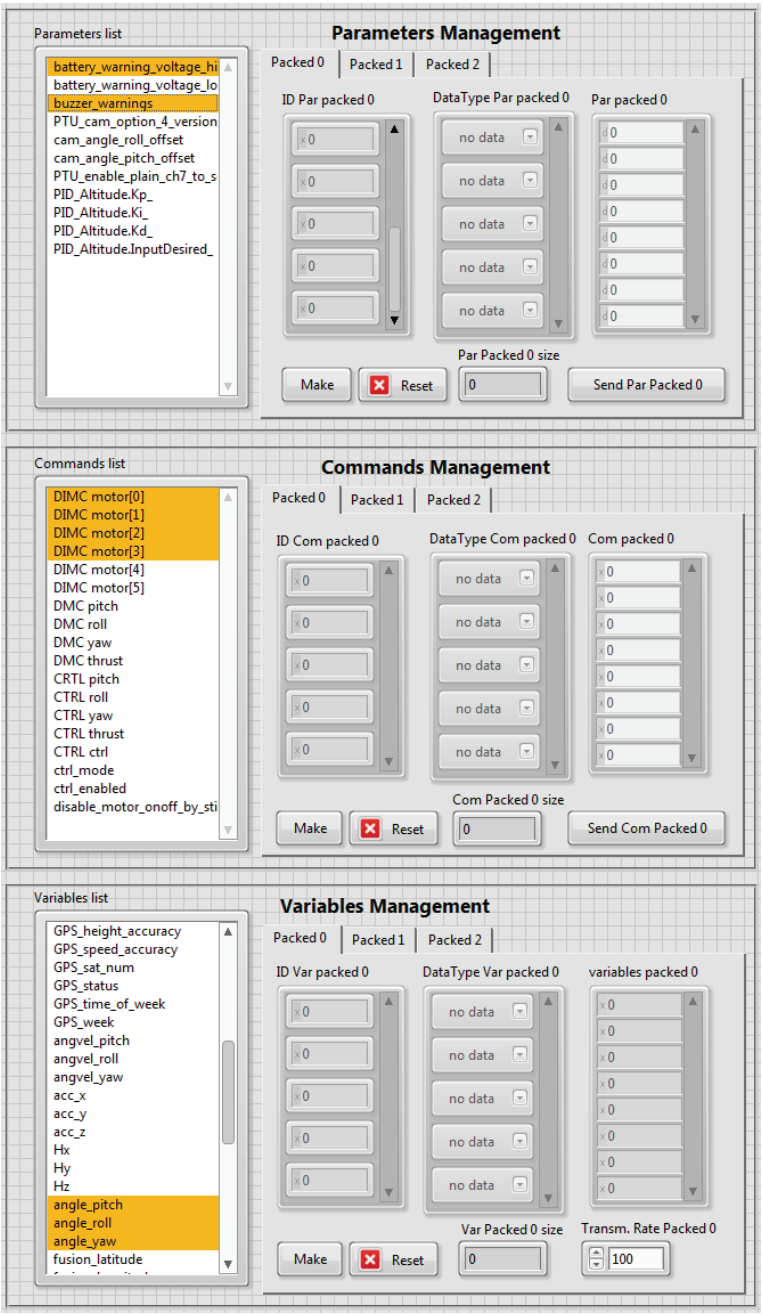

Figure 103 - Variables, Commands and Parameters management

In order to log the data relating to the Volcan and the Hummingbird, the form for the datalog setup, shown in Figure 104, has been developed.

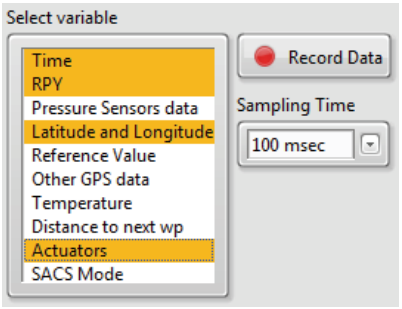

Figure 104 - Datalog form

IV.3.4 Map providers

The adoption of the GMap.NET framework permits to use different map provider.

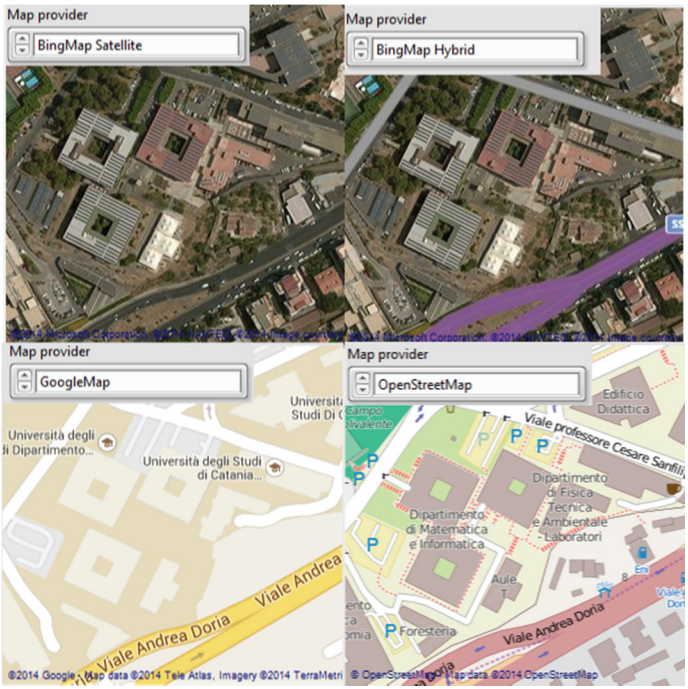

Figure 105 - Examples of map providers

This is useful because, for example, a satellite image provides a lot of detail of the environment, but generally the maximum level of zoom is lower than a conventional map. The Figure 105 shows the same place, the DIEEI laboratories, from different map providers.

IV.3.5 Waypoints and routes

The *subVI* treated in the sections IV.2.6.7 - IV.2.6.11 have been used to develop the waypoint and route management. In Figure 106 an example of waypoint and route management is shown.

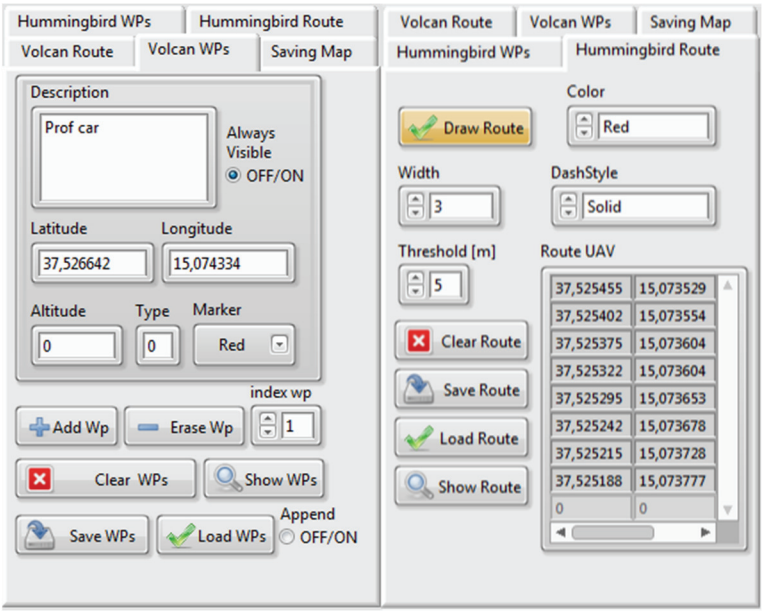

Figure 106 - WPs and route management

Finally, in Figure 107 an example is shown.

Figure 107 - Mapping example

Conclusions

Potentialities

The effectiveness of the developed Volcan control system has been confirmed by several flight tests, one of them is treated in [17]. Other satisfactory tests have been executed also with the Maya model by Bormatec [61] (Figure 108).

As concerns the IMU board, the comparison with other commercial inertial platforms has confirmed its potentialities. However, some improvements could be introduced, such as the compensation of the magnetometer disturbs caused by external magnetic field close to the IMU.

As regards the Hummingbird quadrotor, the developed ACI library has made possible to control and supervise the robotic platform, not to mention the possibility to integrate and manage different payloads.

Finally the Multiplatform Drone HMI represents a powerful tool capable to manage different UVSs, in order to accomplish complex tasks that require coordination and cooperation between the different robots.

Figure 108 - Maya by Bormatec

Limits

As regards the autotuning of the Volcan control loops, the algorithm requires the insertion of initial values of the PID gains, based on the designer experience. However, in case of a new type of aircraft with its own dynamic model, this approach could be really complex and risky, because the initial values could carry the system to an instable condition. Moreover, there is no guarantee that the PID gains achieved by our tuning algorithm are optimal and that they guarantee a robust stability. The adoption of genetic algorithms to set up a multi-objective optimization problem could improve this point.
Another limit to take into consideration resides in the precision of the GPS used in both aerial platform [51]. The accuracy of a couple of meters could represent a really disabling limit for tasks that require high precision positioning, such as collision avoidance and robots cooperation.

Future works

The next step of this work will be the development of a sensor board with RTK GPS, capable to ensure a centimetric accuracy.

Furthermore, several enhancements of the LabView HMI, such as the managements of UGVs, is currently under study.

Appendix A.
Volcan control system schematics

FCCS Schematic

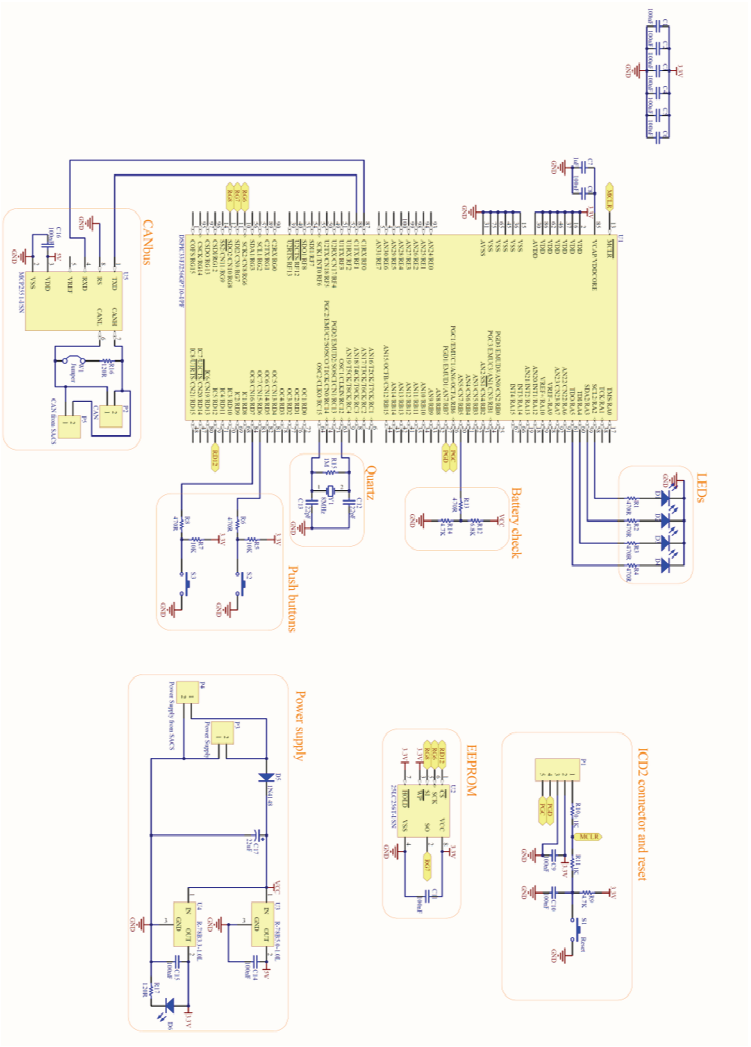

Figure 109 - FCCS Schematic

SACS Schematic

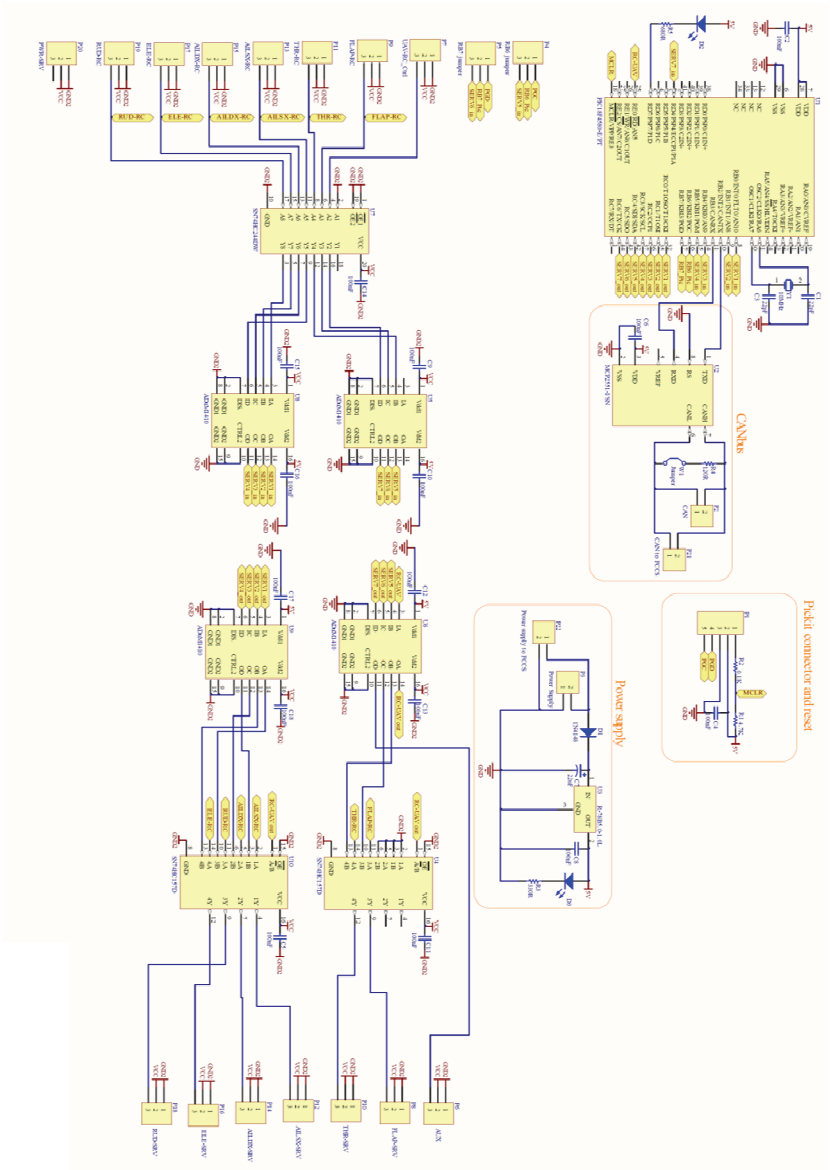

Figure 110 - SACS Schematic

UDP2CAN Schematic

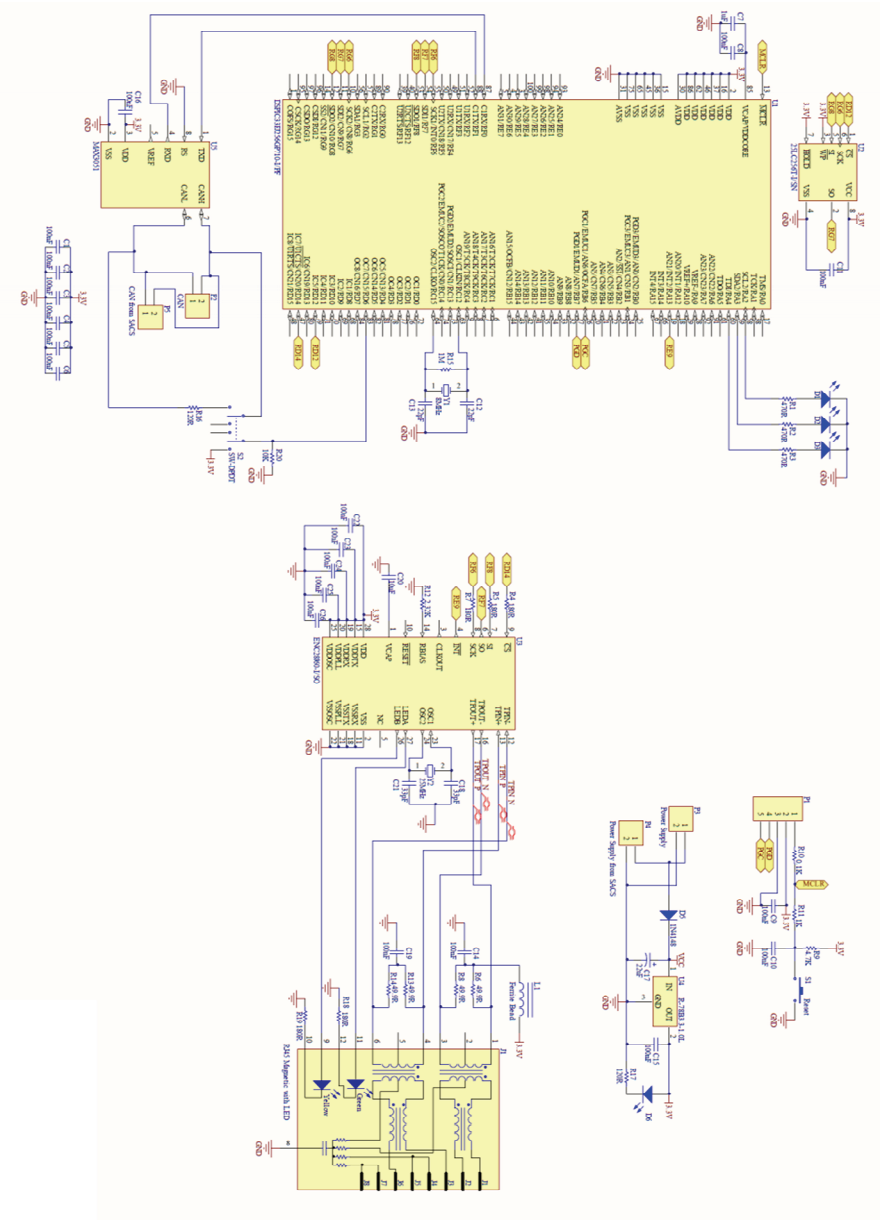

Figure 111 - UDP2CAN Schematic

Sensor Board Schematic

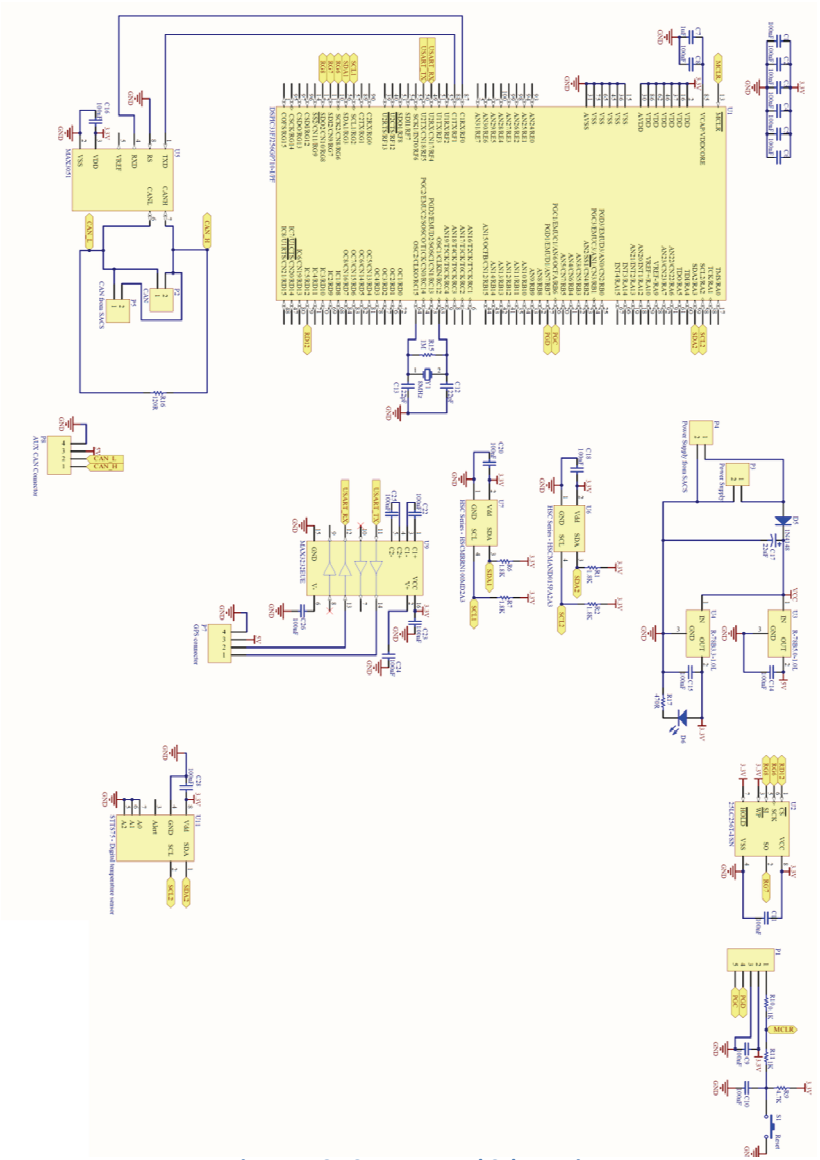

Figure 112 - Sensor Board Schematic

IMU Board Schematic

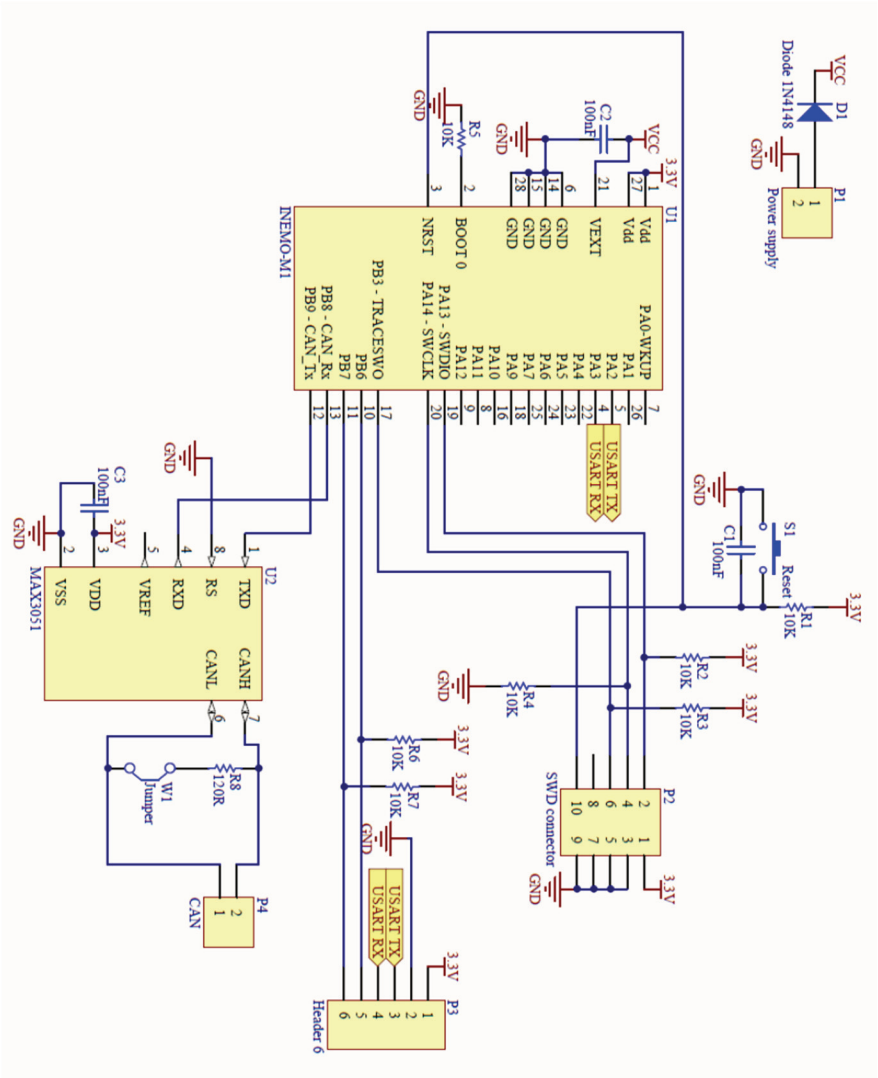

Figure 113 - IMU Board Schematic

Appendix B. IMU Board CANAerospace frames

NSH Frames

As mentioned in the section II.2.3.1, these frames classes are used to configure the system or to send and receive state information.

IDS (Identification service)

This frame is generally sent by the HMI to the IMU in order to get information regarding HW and FW version. Moreover it is also used to ping the system.

HMI ->IMU

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_REQ_1_ID dec:130 hex:0x82	Node ID	Data type	Service code	Message code				
	5	AS_NODATA (0x0)	IDS (0x0)	0	0	0	0	0

IMU ->HMI

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_RSP_1_ID dec:131 hex:0x83	Node ID	Data type	Service code	Message code				
	5	AS_UCHAR_2 (0x13)	IDS (0x0)	Error Flag	IMU_Hw_rev	IMU_Sw_rev	0	0

DTS (Data Transfer Service)

This frame is used to configure the sending data. This frame contains the following fields organized as it follows:

- Data4: In this byte the flag *TransferMode* is inserted, whose value identifies a data transmission in streaming mode (*TransferMode* = 0) or in remote request (*TransferMode* = 1).
- Data5: in the case of streaming mode, in this field the value of the streaming frequency in Hz is inserted.
- Data6 and Data7: in these two bytes there are the variable *DataSelection*, whose each individual bits identify the presence or absence of the corresponding variable (see Table 17) in the data streaming. A bit set to 1

means that the variable is sent, otherwise the bit set to 0 means that the variable is not present in the data streaming.

DataSelection	Variable
Bit 0	Roll
Bit 1	Pitch
Bit 2	Yaw
Bit 3	Quat-Q0
Bit 4	Quat-Q1
Bit 5	Quat-Q2
Bit 6	Quat-Q3
Bit 7	AccX
Bit 8	AccY
Bit 9	AccZ
Bit 10	GyrX
Bit 11	GyrY
Bit 12	GyrZ
Bit 13	MagX
Bit 14	MagY
Bit 15	MagZ

Table 17 - *DataSelection* values

HMI ->IMU

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_REQ_1_ID dec:130 hex:0x82	Node ID	Data type	Service code	Message code			DataSelection	
	5	AS_UCHAR_4 (0x10)	DTS(0xD4)	0	Transfer Mode	Datarate [Hz]	MSB	LSB

IMU ->HMI

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_RSP_1_ID dec:131 hex:0x83	Node ID	Data type	Service code	Message code			DataSelection	
	5	AS_UCHAR_4 (0x10)	DTS(0xD4)	Error Flag	Transfer Mode	Datarate [Hz]	MSB	LSB

RTS (Raw Data Transfer Service)

This frame is used to configure the sending of raw data, i.e. the data coming from the IMU sensors (16bit) without any modifications. This frame contains the fields organized as it follows:

- Data4 and Data5 : these bytes have the same meaning of the DTS frame.
- Data7: in this byte the *RawDataSelection* variable will be transmitted, whose each individual bit identifies the presence or absence of the corresponding variable (see Table 18) in the raw data streaming.

RawDataSelection	Variable
Bit 0	Degub Frame 1
Bit 1	Degub Frame 2
Bit 2	Degub Frame 3
Bit 3	Degub Frame 4
Bit 4	Degub Frame 5

Table 18 – RawDataSelectionvalues

HMI ->IMU

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_REQ_1_ID dec:130 hex:0x82	Node ID	Data type	Service code	Message code				
	5	AS_UCHAR_3 (0x1B)	RTS(0xD8)	0	RawTransfer Mode	Raw Datarate [Hz]		RawData Selection

IMU ->HMI

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_RSP_1_ID dec:131 hex:0x83	Node ID	Data type	Service code	Message code				
	5	AS_UCHAR_3 (0x1B)	RTS(0xD8)	Error Flag	RawTransfer Mode	Raw Datarate [Hz]		RawData Selection

SSS (Start and Stop data transfer Service)

This frame is used to manage both the data streaming and the raw data streaming, where they are in streaming mode. Each stream has a flag, respectively *StartStopData* (Data6) and *StartStopRawData* (Data7), whose values represent the following meaning:

- 1, start streaming.
- 2, stop streaming.
- 0, streaming unchanged.

HMI ->IMU

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_REQ_1_ID dec:130 hex:0x82	Node ID	Data type	Service code	Message code				
	5	AS_UCHAR_2 (0x13)	SSS(0xD5)	0	0	0	StartStop Data	StartStop RawData

IMU ->HMI

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_RSP_1_ID dec:131 hex:0x83	Node ID	Data type	Service code	Message code				
	5	AS_UCHAR_2 (0x13)	SSS(0xD5)	Error Flag	0	0	StartStop Data	StartStop RawData

CDS (Control Parameters download service)

This frame is used to load the configuration parameters in the inertial platform. The selection of the parameter depends on the value of *parameter_identifier*, in according with the following table:

parameter_identifier	Description	Parameter name
0	Ellipsoid eccentricity X	EccX
1	Ellipsoid eccentricity Y	EccY
2	Ellipsoid eccentricity Z	EccZ
3	Ellipsoid radius X	ErX
4	Ellipsoid radius Y	ErY
5	Ellipsoid radius Z	ErZ
6	X component magnetic field	EmfX
7	Y component magnetic field	EmfY
8	Z component magnetic field	EmfZ
9	X component gravity field	GX
10	Y component gravity field	GY
11	Z component gravity field	GZ
12	Offset Roll	offset_Roll
13	Offset Pitch	offset_Pitch
14	Offset Yaw	offset_Yaw

Table 19 - Parameter identifier values

HMI ->IMU

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_REQ_1_ID dec:130 hex:0x82	Node ID	Data type	Service code	Message code	Variable name: Param_x			
	5	AS_FLOAT(0x02)	CDS(0xA3)	Parameter identifier	Float MSB	Float2	Float3	Float LSB

IMU ->HMI

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_RSP_1_ID dec:131 hex:0x83	Node ID	Data type	Service code	Message code	Variable name: Param_x			
	5	AS_FLOAT(0x02)	CDS(0xA3)	Parameter identifier	Float MSB	Float2	Floa_3	Float LSB

CUS (Control Parameters upload service)

This is the dual frame of the CDS. It provides the upload of a given parameter from the IMU to the HMI. The value of the variable *parameter_identifier* is the same of the CDS frame.

HMI ->IMU

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_REQ_1_ID dec:130 hex:0x82	Node ID	Data type	Service Code	Message code				
	5	AS_NODATA(0x0)	CUS(0xA5)	Parameter identifier	0	0	0	0

IMU ->HMI

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_RSP_1_ID dec:131 hex:0x83	Node ID	Data type	Service Code	Message code	Variable name: Param_x			
	5	AS_FLOAT (0x02)	CUS(0xA5)	Parameter identifier	Float MSB	Float_2	Float_3	Float LSB

PRS (Parameters Reset Service)

This service is used to set the default value of a parameter (chosen in according to Table 19).

HMI ->IMU

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_REQ_1_ID dec:130 hex:0x82	Node ID	Data type	Service code	Message code				
	5	AS_NODATA(0x0)	PRS(0xCB)	Parameter identifier	0	0	0	0

IMU ->HMI

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_RSP_1_ID dec:131 hex:0x83	Node ID	Data type	Service code	Message code	Variable name: Param_x			
	5	AS_FLOAT(0x02)	PRS(0xCB)	Parameter identifier	Float MSB	Float_2	Float_3	Float LSB

CMU (Change Measurement Unit)

With this service it is possible to set the unit of measurement of the Euler angles, respectively in radians (*Unit* = 0) or in degrees (*Unit* = 1).

HMI ->IMU

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_REQ_1_ID dec:130 hex:0x82	Node ID	Data type	Service code	Message code				
	5	AS_UCHAR (0x0A)	CUM(0xD3)	0	0	0	0	Unit

IMU ->HMI

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_RSP_1_ID dec:131 hex:0x83	Node ID	Data type	Service code	Message code				
	5	AS_UCHAR (0x0A)	CUM(0xD3)	Error Flag	0	0	0	Unit

ROS (Reset Orientation Service)

Using this service the attitude of the IMU (roll and pitch) is reset. This frame is useful when there is a rotation between the reference system fixed to the airframe and the reference system fixed to the IMU.

HMI ->IMU

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_REQ_1_ID dec:130 hex:0x82	Node ID	Data type	Service code	Message code				
	5	AS_NODATA(0x0)	ROS(0xD6)	0	0	0	0	0

IMU ->HMI

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_RSP_1_ID dec:131 hex:0x83	Node ID	Data type	Service code	Message code				
	5	AS_NODATA(0x0)	RCS(0xD6)	Error Flag	0	0	0	0

RHS (Reset Heading Service)

This frame provides a reset of the yaw, that in this case it does not point to north anymore.

HMI ->IMU

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_REQ_1_ID dec:130 hex:0x82	Node ID	Data type	Service code	Message code				
	5	AS_NODATA(0x0)	RHS(0xD7)	0	0	0	0	0

IMU ->HMI

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_RSP_1_ID dec:131 hex:0x83	Node ID	Data type	Service code	Message code				
	5	AS_NODATA(0x0)	RHS(0xD7)	Error Flag	0	0	0	0

RCS (Reset CPU Setting Service)

This last service executes the reset of the onboard microcontroller.

HMI ->IMU

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_REQ_1_ID dec:130 hex:0x82	Node ID	Data type	Service code	Message code				
	5	AS_NODATA(0x0)	RCS(0xD2)	0	0	0	0	0

IMU ->HMI

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
NS_RSP_1_ID dec:131 hex: 0x83	Node ID	Data type	Service code	Message code				
	5	AS_NODATA(0x0)	RCS(0xD2)	Error Flag	0	0	0	0

NOD frames

The following frames are used to send the calibrated data of the IMU via CANAerospace.

Pitch:

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
BODY_PITCH_ANGLE_ID dec:311 hex: 0x137	Node ID	Data type	Service code	Message code	Variable name: Pitch [rad]			
	5	AS_FLOAT (0x02)	0	0	Float MSB	Float_2	Float_3	Float LSB

Roll:

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
BODY_ROLL_ANGLE_ID dec:312 hex: 0x138	Node ID	Data type	Service code	Message code	Variable name: Roll[rad]			
	5	AS_FLOAT (0x02)	0	0	Float MSB	Float_2	Float_3	Float LSB

Yaw:

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
HEADING_ANGLE_ID dec:321 hex: 0x141	Node ID	Data type	Service code	Message code	Variable name: Heading [rad]			
	5	AS_FLOAT (0x02)	0	0	Float MSB	Float_2	Float_3	Float LSB

Longitudinal Acceleration (X):

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
BODY_LONG_ACC_ID dec:300 hex:0x12C	Node ID	Data type	Service code	Message code	Variable name: acc_X [m]/[sec²]			
	5	AS_FLOAT (0x02)	0	0	Float MSB	Float_2	Float_3	Float LSB

Lateral Acceleration (Y):

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
BODY_LAT_ACC_ID dec:301 hex: 0x12D	Node ID	Data type	Service code	Message code	Variable name: acc_Y[m]/[sec²]			
	5	AS_FLOAT(0x02)	0	0	Float MSB	Float_2	Float_3	Float LSB

Normal Acceleration (Z):

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
BODY_NORM_ACC_ID dec:302 hex:0x12E	Node ID	Data type	Service code	Message code	Variable name: acc_Z[m]/[sec²]			
	5	AS_FLOAT (0x02)	0	0	Float MSB	Float_2	Float_3	Float LSB

Quaternion q0:

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
BODY_QUAT_Q0_ID dec:1500 hex:0x5DC	Node ID	Data type	Service code	Message code	Variable name: Quat_q0			
	5	AS_FLOAT(0x02)	0	0	Float MSB	Float_2	Float_3	Float LSB

Quaternion q1:

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
BODY_QUAT_Q1_ID dec:1501 hex:0x5DD	Node ID	Data type	Service code	Message code	Variable name: Quat_q1			
	5	AS_FLOAT(0x02)	0	0	Float MSB	Float_2	Float_3	Float LSB

Quaternion q2:

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
BODY_QUAT_Q2_ID dec:1502 hex:0x5DE	Node ID	Data type	Service code	Message code	Variable name: Quat_q2			
	5	AS_FLOAT(0x02)	0	0	Float MSB	Float_2	Float_3	Float LSB

Quaternion q3:

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
BODY_QUAT_Q3_ID dec:1503 hex:0x5DF	Node ID	Data type	Service code	Message code	Variable name: Quat_q0			
	5	AS_FLOAT(0x02)	0	0	Float MSB	Float_2	Float_3	Float LSB

Gyroscope X:

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
BODY_GYR_X_ID dec:1504 hex:0x5E0	Node ID	Data type	Service code	Message code	Variable name: gyr_X [DEG]/[sec]			
	5	AS_FLOAT(0x02)	0	0	Float MSB	Float_2	Float_3	Float LSB

Gyroscope Y:

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
BODY_GYR_Y_ID dec:1505 hex:0x5E1	Node ID	Data type	Service code	Message code	Variable name: gyr_Y[DEG]/[sec]			
	5	AS_FLOAT(0x02)	0	0	Float MSB	Float_2	Float_3	Float LSB

Gyroscope Z:

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
BODY_GYR_Z_ID dec:1506 hex:0x5E2	Node ID	Data type	Service code	Message code	Variable name: gyr_Z[DEG]/[sec]			
	5	AS_FLOAT(0x02)	0	0	Float MSB	Float_2	Float_3	Float LSB

Magnetometer X:

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
BODY_MAGN_X_ID dec:1507 hex:0x5E3	Node ID	Data type	Service code	Message code	Variable name: Magn_X [G]			
	5	AS_FLOAT(0x02)	0	0	Float MSB	Float_2	Float_3	Float LSB

Magnetometer Y:

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
BODY_MAGN_Y_ID dec:1508 hex:0x5E4	Node ID	Data type	Service code	Message code	Variable name: Magn_Y [G]			
	5	AS_FLOAT(0x02)	0	0	Float MSB	Float_2	Float_3	Float LSB

Magnetometer Z:

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
BODY_MAGN_Y_ID dec:1509 hex:0x5E5	Node ID	Data type	Service code	Message code	Variable name: Magn_Z [G]			
	5	AS_FLOAT(0x02)	0	0	Float MSB	Float_2	Float_3	Float LSB

DSD frames

The following frames are used to send the uncalibrated raw data of the IMU via CANAerospace.

Debug Frame 1, raw data accelerometer, x e y axis:

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
DEBUG_FRAME_1_ID dec:1920 hex:0x780	Node ID	Data type	Service code	Message code	raw_Acc_X		raw_Acc_Y	
	5	AS_USHORT_2 (0x0D)	0	0	MSB	LSB	MSB	LSB

Debug Frame 2, raw data accelerometer z e gyroscope x axis:

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
DEBUG_FRAME_2_ID dec:1921 hex:0x781	Node ID	Data type	Service code	Message code	raw_Acc_Z		raw_Gyr_X	
	5	AS_USHORT_2 (0x0D)	0	0	MSB	LSB	MSB	LSB

Debug Frame 3, raw data gyroscope y e z axis:

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
DEBUG_FRAME_3_ID dec:1922 hex:0x782	Node ID	Data type	Service code	Message code	raw_Gyr_Y		raw_Gyr_Z	
	5	AS_USHORT_2 (0x0D)	0	0	MSB	LSB	MSB	LSB

Debug Frame 4, raw data magnetometer x e y axis:

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
DEBUG_FRAME_4_ID dec:1923 hex:0x783	Node ID	Data type	Service code	Message code	raw_Mag_X		raw_Mag_Y	
	5	AS_USHORT_2 (0x0D)	0	0	MSB	LSB	MSB	LSB

Debug Frame 5, raw data magnetometer z e temperature:

ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
DEBUG_FRAME_5_ID dec:1924 hex:0x784	Node ID	Data type	Service code	Message code	Temperature		raw_Mag_Z	
	5	AS_USHORT_2 (0x0D)	0	0	MSB	LSB	MSB	LSB

References

[1] Petricca, Luca, Per Ohlckers, and Christopher Grinde. "Micro-and nano-air vehicles: State of the art." International Journal of Aerospace Engineering 2011 (2011).

[2] Valavanis, Kimon P., ed. Advances in unmanned aerial vehicles: state of the art and the road to autonomy. Vol. 33. Springer, 2008.

[3] Valavanis, Kimon P., and George J. Vachtsevanos. "Future of Unmanned Aviation." Handbook of Unmanned Aerial Vehicles. Springer Netherlands, 2014. 2993-3009.

[4] Quaritsch, Markus, et al. "Networked UAVs as aerial sensor network for disaster management applications." e & i Elektrotechnik und Informationstechnik 127.3 (2010): 56-63.

[5] Maza, Iván, et al. "Experimental results in multi-UAV coordination for disaster management and civil security applications." Journal of intelligent & robotic systems 61.1-4 (2011): 563-585.

[6] Grenzdörffer, G. J., A. Engel, and B. Teichert. "The photogrammetric potential of low-cost UAVs in forestry and agriculture." The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 31.B3 (2008): 1207-1214.

[7] Metni, Najib, and Tarek Hamel. "A UAV for bridge inspection: Visual servoing control law with orientation limits." Automation in construction 17.1 (2007): 3-10.

[8] Winkvist, Stefan, Emma Rushforth, and Ken Young. "Towards an autonomous indoor aerial inspection vehicle." Industrial Robot: An International Journal 40.3 (2013): 196-207.

[9] Murphy, Douglas W., and James Cycon. "Applications for mini VTOL UAV for law enforcement." Enabling Technologies for Law Enforcement and Security. International Society for Optics and Photonics, 1999.

[10] Casbeer, David W., et al. "Cooperative forest fire surveillance using a team of small unmanned air vehicles." International Journal of Systems Science 37.6 (2006): 351-360.

[11] G. Muscato et al., "Volcanic Environments: Robots for Exploration and Measurement in Volcanic Environments", Robotics & Automation Magazine, IEEE , vol.19, pp 40-49,March 2012.

[12] Caltabiano, Daniele, et al. "Architecture of a UAV for volcanic gas sampling."Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on. Vol. 1. IEEE, 2005.

[13] Hausamann, Dieter, et al. "Monitoring of gas pipelines–a civil UAV application."Aircraft Engineering and Aerospace Technology 77.5 (2005): 352-360.

[14] Enac normative documentation http://www.enac.gov.it/repository/ContentManagement/information/N1311250085/Bozza_Circolare_APR_140502.pdf last accessed November 2014.

[15] ICAO normative documentation http://www.icao.int/Meetings/UAS/Documents/Circular%20328_en.pdf last accessed November 2014.

[16] Volcan UAV documentation http://www.robotic.diees.unict.it/robots/uav/docs/Volcan%20UAV%20Project.pdf last accessed November 2014.

[17] Catena et al., "A New Modular Architecture for the Control of the VOLCAN RPAS for Volcanic Activity Monitoring", in IROS2012 Workshop on Robotics for Environmental Monitoring, October 2012.

[18] Aerosonde webpage http://www.aerosonde.com/ last accessed November 2014.

[19] CANbus documentation by BOSCH http://esd.cs.ucr.edu/webres/can20.pdf last accessed November 2014.

[20] ISO 11898 web page: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=33422 last accessed October 2014.

[21] CANAerospace webpage http://www.stockflightsystems.com/canaerospace.html last accessed October 2014.

[22] A.Catena, "Sviluppo, implementazione e simulazione di architetture embedded di controllo per UAV su piattaforma hardware in the Loop", Master Thesis, Universistà degli Studi di Catania, October 2011.

[23] L. Sciavicco,B. Siciliano, "Robotica Industriale", THE MCGRAW-HILL COMPANIES, 2002/2003.

[24] B. L. Stevens, F. L. Lewis, "Aircraft Control and Simulation, 2nd Edition", Wiley, ISBN: 978-0-471-37145-8, 2003.

[25] A. Catena, C. D. Melita, and G. Muscato. "Automatic Tuning Architecture for the Navigation Control Loops of Unmanned Aerial Vehicles." Journal of Intelligent & Robotic Systems 73.1-4 (2014): 413-427.

[26] U.S. Department of Transportation, Federal Aviation Administration, "Pilot's Handbook of Aeronautical Knowledge", FAA Handbooks, ISBN-10: 1560277505, 2009.

[27] Karl J. Åström and Tore Hägglund, "PID Controllers: Theory, Design, and Tuning, 2nd Edition", pp. 230-270, 1995.

[28] Yang Shengyi et al., "Design and Simulation of the Longitudinal Autopilot of UAV Based on Self-Adaptive Fuzzy PID Control", International Conference on Computational Intelligence and Security, pp. 634 - 638, Dec. 2009.

[29] Wu-fa Liu et al., "Online Fuzzy Self-Adaptive PID Attitude Control of a Sub Mini Fixed-Wing Air Vehicle", International Conference on Mechatronics and Automation, pp. 153-157, Aug. 2007.

[30] T. Sangyam et al. "Autonomous path tracking and disturbance force rejection of UAV using fuzzy based auto-tuning PID controller" Electrical Engineering/Electronics Computer Telecommunications and Information Technology (ECTI-CON), 2010 International Conference on , vol., no., pp.528,531, 19-21 May 2010.

[31] A. Kirli et al., "Self tuning fuzzy PD application on TI TMS320F28335 for an experimental stationary quadrotor", Education and Research Conference (EDERC), 2010 4th European , vol., no., pp.42,46, 1-2 Dec. 2010.

[32] G. Astuti et al., "HIL tuning of UAV for exploration of risky environments", International Journal on Advanced Robotic Systems, Vol.5, N.4, December 2008.

[33] Chang-Sun Yoo et al., "Hardware-In-the-Loop simulation test for actuator control system of Smart UAV", Control Automation and Systems (ICCAS), 2010 International Conference on , vol., no., pp.1729,1732, 27-30 Oct. 2010.

[34] A. Leva, "Comparative study of model-based PI(D) autotuning methods", American Control Conference, 2007. ACC '07 , vol., no., pp.5796,5801, 9-13 July 2007

[35] J. How et al., "Flight Demonstrations of Cooperative Control for UAV Teams", AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop and Exhibit, 20 - 23 September 2004, Chicago, Illinois.

[36] N. Regina, M. Zanzi, "Fixed-wing UAV guidance law for surface-target tracking and overflight", Aerospace Conference, 2012 IEEE , vol., no., pp.1,11, 3-10 March 2012.

[37] STMicroelectronics INEMO® system-on-board Datasheet, http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/DM00056715.pdf last accessed October 2014.

[38] Kalman, Rudolph Emil. "A new approach to linear filtering and prediction problems." Journal of Fluids Engineering 82.1 (1960): 35-45.

[39] Welch, Greg, and Gary Bishop. "An introduction to the Kalman filter." (1995).

[40] Marins, João Luís, et al. "An extended Kalman filter for quaternion-based orientation estimation using MARG sensors." Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on. Vol. 4. IEEE, 2001.

[41] Sabatini, Angelo M. "Quaternion-based extended Kalman filter for determining orientation by inertial and magnetic sensing." Biomedical Engineering, IEEE Transactions on 53.7 (2006): 1346-1356.

[42] Euston, Mark, et al. "A complementary filter for attitude estimation of a fixed-wing UAV." Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on. IEEE, 2008.

[43] Metni, Najib, et al. "Attitude and gyro bias estimation for a VTOL UAV." Control Engineering Practice 14.12 (2006): 1511-1520.

[44] Diebel, James. "Representing attitude: Euler angles, unit quaternions, and rotation vectors." Matrix 58 (2006): 15-16.

[45] Robert Mahony, Vijay Kumar, Peter Corke, "Multirotor aerial vehicles, modeling, estimation, and control of quadrotor".

[46] ACI Remote Code Documentation webpage: http://www2.asctec.de/aci/remote_code_doku/index.html, last accessed October 2014.

[47] ACI Asctec, list of all predefined variables, commands and parameters http://wiki.asctec.de/display/AR/List+of+all+predefined+variables%2C+commands+and+parameters, last accessed October 2014.

[48] LabView webpage http://www.ni.com/labview/last accessed October 2014.

[49] PictureBox Rotation webpage http://www.codeproject.com/Articles/58815/C-Image-PictureBox-Rotations last accessed October 2014.
[50] GMap.NET webpage http://greatmaps.codeplex.com/ last accessed October 2014.
[51] GPS Position Accuracy Measures, NovAtel, http://www.novatel.com/assets/Documents/Bulletins/apn029.pdf last accessed November 2014.
[52] DIEEI Robotics web site: http://www.robotic.diees.unict.it/ last accessed November 2014.
[53] X-Plane by Laminar Research webpage: http://www.X-Plane.com, last accessed October 2014.
[54] Xsens products webpage: http://www.xsens.it/prodotti.php, last accessed October 2014.
[55] PCAN-USB webpage http://www.peak-system.com/PCAN-USB.199.0.html?&L=1last accessed October 2014.
[56] PCAN-USB Labview Driver http://www.peak-system.com/Lab-View-Driver.255.0.html?&L=1last accessed October 2014.
[57] FTDI webpage http://www.ftdichip.com/last accessed October 2014.
[58] FTDI labview vi download link http://www.ftdichip.com/Support/SoftwareExamples/CodeExamples/LabVIEW/LabVIEW_Byte_7.0.zip last accessed October 2014.
[59] Vicon motion capture webpage http://www.vicon.com/ last accessed October 2014.
[60] Hummingbird by Asctec webpage http://www.asctec.de/en/uav-uas-drone-products/asctec-hummingbird/ last accessed October 2014.
[61] Bormatec webpage http://bormatec.com/ last accessed November 2014.
[62] ETH webpage https://www.ethz.ch/en.html last accessed November 2014.
[63] CATEC webpage http://www.catec.aero/ last accessed November 2014.
[64] Ambition Power webpage http://www.ambitionpower.org/ last accessed November 2014.
[65] Microchip webpage http://www.microchip.com/ last accessed November 2014.
[66] Honeywell webpage http://honeywell.com/Pages/Home.aspx last accessed November 2014.

[67] KUKA manipulators webpage http://www.kuka-robotics.com/ last accessed November 2014.
[68] Matlab-Simulink webpage http://www.mathworks.com/ last accessed November 2014.