UNIVERSITÀ
DEGLI STUDI
FIRENZE

Università degli Studi di Firenze

Dipartimento di Ingegneria dell'Informazione (DINFO)

Corso di Dottorato in Ingegneria dell'Informazione

Curriculum: Electronics, Electromagnetics and Electrical Systems

---

Development of navigation techniques and algorithms for small UAVs (*Unmanned Aerial Vehicle*) able to follow trajectories with centimeter precision

Candidate
Luca Bigazzi

Supervisors
Prof. Michele Basso

Prof. Massimiliano Pieraccini

Dr. Giacomo Innocenti

PhD Coordinator
Prof. Fabio Schoen

---

ciclo XXXIII, 2017-2020

Università degli Studi di Firenze, Dipartimento di Ingegneria
dell'Informazione (DINFO).

Ai miei genitori Luciano e Franca

Acknowledgments

I would like to thank my supervisors, Prof. Michele Basso, Prof. Massimiliano Pieraccini and Giacomo Innocenti for following and helping me throughout my PhD course. Furthermore, I would like to thank Dr. Stefano Gherardini, who gave me an important help during the laboratory tests and also for the support he has given me over the years. Finally, I would like to make a special thanks to Dr. Luigi Pannocchi with whom it was possible to develop the hardware-in-the-loop technique proposed in the chapter 5.

Contents

Chapter 1

Introduction

In the last decade, research in multi-rotor UAVs drawn increasing interest and funding from both academic and industrial communities, thanks to their versatility, low-cost realization, and promising unexplored applications [1, 2]. Indeed, UAV research led to technological advances in mechatronic servo-systems, microelectronics, and sensors, which, combined with novel economically affordable and high-performing micro-controllers and embedded boards, rapidly increased their general performance [3–5]. Notable examples of successful application of UAV technologies can be found in precision agriculture [6–8], where drones equipped with Real-Time Kinematic Global Navigation Satellite Systems (GNSSs-RTK) [9] are used to minimize human intervention.

Other important applications, instead, are related to monitoring and patrolling. For example, drones are well suited to explore or inspect large environments and buildings [10, 11] aiming, for instance, to 3D reconstruction. Moreover, drones high maneuverability in small spaces fits both urban missions [12–15, 28], such as monitoring road traffic, and rescue missions [29–32,34,35], such as patrolling dangerous zone after natural disasters. In many of these applications Vision-Based Navigation (VBN) algorithms are often used to improve accuracy in the positioning [36–43]. Film makers and video studios are successfully using UAVs for aerial shooting [44, 45]. These drones are usually big, because they need to load heavy cameras and their gimbals. The size, the weight, and the large number of engines make them very stable, and so, under the driving of skillful pilots, they are able to accomplish high precision maneuvers of the order of tens of centimeters.

On-board systems are only used for improving stability and assisted maneuvering, while the navigation performance is completely left to the pilot's ability and sensitivity.

In the field of autonomous UAVs, the actual precision for outdoor applications is limited by the mostly used sensing technology, i.e., by standard GNSS devices, which today are able to locate the drone within a one meter radius at the very best [46, 47]. The development of novel applications is severely limited by this performance level, and, therefore, the demand for better technologies is growing by the day. When higher performance is required, then, a finer positioning system turns out necessary, as it already happens for acrobatic drones [48–50]. In this case the most effective approach consists in exploiting a ground-assisted positioning system. For example, in indoor applications, the typical solution involves a large number of fixed cameras, which are able to detect special markers on the drone (motion capture). A ground station processes the data from the cameras by a computer vision algorithm that generates a high-precision position estimate at high-frequency. Possibly, the cameras can be replaced by other ground localization systems without changing the overall paradigm [51–59]. The ground station either can send the computed information to the drone, or, more usually, can pilot directly the UAV exploiting this data. Whichever the case, this solution prevents a completely autonomous drone.

The degree of autonomy of a UAV is a non-secondary factor when developing new applications, because any necessary environmental set up represents a cost and a delay. Recent projects, indeed, are investigating solutions, which may lessen the limits of the actual technologies recurring to an extensive use of sensor fusion techniques [60–62]. Nevertheless, precision and complete autonomy are yet difficult features to get together.

1.1 My objectives

The objectives that I set myself during the PhD course were to develop autonomous navigation techniques and algorithms for UAVs, capable of moving with centimeter accuracy and to follow three-dimensional trajectories. Studying the literature, I soon realized that in order to apply research in this field it would be necessary to develop a hardware and software platform, through which realize the algorithms of interest. This is because currently there are no valid and cheap platforms on the market able to meet the re-

quirements for this type of application. For this purpose, part of my research has focused on the development of the technology by which it is possible to test the autonomous navigation algorithms.

1.2 Novel contributions

Among the novelties shown in this research work it is possible to mention the innovative techniques and methodologies developed for the tracking of 3D trajectories in the UAV field, through the use of systems based on artificial vision. A not insignificant aspect is the fact that all the technologies presented are of the on-board type and do not rely on external sensors (such as fixed cameras in the environment that create motion capture systems), this guarantees high performance (thanks to computer vision technique) and high flexibility (thanks to the fact that autonomous UAV prototypes do not need external help to function). The software engineering approaches are also mentioned, which in their latest implementations are able to make UAVs capable of carrying out missions with an unknown path. Finally, the design phase of the multilevel hardware architecture also constitutes an important innovative contribution, as it provides a solid platform for doing research in the autonomous UAV field.

1.3 Tables of notations and abbreviations

Main notations	
$\widehat{\phi}_k$	Attitude estimate
$\widehat{\phi}_{k+1}$	Updated attitude estimate
Ω_k	Vector of the angular velocities
$\widehat{Q}^f$	Estimate of the drone position in fixed frame
Q^b	Drone position in body frame
Q^c	Position of the marker in camera frame
$\vec{Q}_{cb}$	Offset vector between the camera and the body
a^f	Acceleration vector in fixed frame
a^b	Acceleration vector in body frame
a_g	Gravity acceleration
ψ_d^f	Desired heading angle in fixed frame
ψ_b^f	Heading angle of the body in fixed frame
ψ_g^c	Heading of the gate in camera frame
φ_b^f	Pitch angle of the body in fixed frame
θ_b^f	Roll angle of the body in fixed frame
$\vec{O}^f$	Offset vector in fixed frame
$\vec{O}^c$	Offset vector in camera frame
$\vec{P}_{id}^c$	Position of the gate in camera frame
$\vec{P}_b^f$	Position of the body in fixed frame
$R_x(\phi)$	Simple rotation of an angle ϕ around the x axis
$R_y(\phi)$	Simple rotation of an angle ϕ around the y axis
$R_z(\phi)$	Simple rotation of an angle ϕ around the z axis
$R_{xyz}(\phi)$	Complex rotation of an angle ϕ around the three axes
$\vec{E}^c$	Endpoint in camera frame
$\vec{E}^g$	Endpoint in gate frame
$\vec{E}^f$	Endpoint in fixed frame
$\vec{e}^b$	Error vector in body frame
$\vec{e}^f$	Error vector in fixed frame
$\otimes$	Hamilton rule

Main abbreviations	
DOF	Degrees of freedom
IMU	Inertial measurement unit
UAV	Unmanned aerial vehicle
PID	Proportional-integral-derivative
PPM	Pulse position modulation
SBUS	Serial bus
NED	North-east-down
FIFO	First input first output
FPS	Frame per second
CSI	Camera serial interface
MIPI	Mobile industry processor interface
UART	Universal asynchronous receiver-transmitter
MOCAP	Motion capture
VBN	Vision based navigation
GPS	Global positioning system
GNSS	Global navigation satellite systems
RTK	Real time kinematic
UWB	Ultra wide band
ROS	Robotic operating system
CAD	Computer aided design
CNC	Computer numerical control
SLAM	Simultaneous localization and mapping
GPIO	General purpose input-output
EEP	Engine efficiency percentage
OSD	On screen display
FCF-CF	Fixed camera frame complementary filter
FCF-MF	Fixed camera frame Madgwick filter
SCF-MF-2DOF	Stabilized camera frame Madgwick filter 2DOF
SCF-MF-3DOF	Stabilized camera frame Madgwick filter 3DOF
FPGA	Field programmable gate array
ID	Identifier
VISP	Visual servoing platform
API	Application programming interface
UDP	User datagram protocol
TCP	Transmission control protocol

Chapter 2

Overview on autonomous UAVs

In this chapter the technologies used on autonomous UAVs in the academic field will be presented. Furthermore, we will discuss the current state of the art for these systems and on which themes the research community is currently focused. Finally, the concept of multilevel hardware and the type of boards most cited in literature will be introduced.

2.1 Usable technologies

In recent years the number of technologies that can be used in the field of autonomous drones have had a significant increase. The first technology implemented on the drones for position estimation was GPS. However, this technology hardly guaranteed sub-meter accuracies and was limited to outdoor environments (as GPS triangulation is not able to work inside buildings). Given the poor accuracy obtainable through the use of this approach, for a long time autonomous drones have been able to carry out limited missions in pursuit of 2D trajectories, formed by a set of points called "waypoints" [79, 80].

Over the time there has been an exponential increase in the features of the UAVs, which has made them increasingly "smart". An example is the use of additional sensors such as: (i) ultrasonic sensors, useful for avoiding obstacles at short range; (ii) optical flow, minimal optical sensors able to refine the rough positioning estimate provided by the GPS. Lately, thanks to technological evolution, VBN systems are becoming increasingly popu-

lar. This technology, unlike the previous one, can make UAVs capable of following 3D trajectories in space. The performances obtainable through the use of these systems are superior to those that could be obtained with the use of GPS technology alone, moreover it is able to function also indoors. These features greatly increase the autonomous driving capabilities of UAVs, making them able to execute increasingly complex and research-oriented missions. The VBN technology is divided into two conceptually different branches: (i) off-board VBN techniques [83, 88] (realized through the use of *motion capture systems*), where both the cameras and the computing units for autonomous navigation are mounted externally to the UAV in a fixed position; (ii) on-board VBN techniques, where there is no ground aid and all autonomous navigation hardware is mounted on board the UAV. Both configurations have their weaknesses and their advantages.

Off-board VBN techniques have the advantage of guaranteeing extremely high estimation accuracies (of the order of a millimeter), this mainly derives from the fact that the cameras are mounted externally in a fixed position and are able to maintain visual contact of the UAV, avoiding the vision noise that would be present if they were mounted on board the drone. Obviously the weaknesses of this approach are related to the low flexibility of the technique, as it can only work in controlled environments suitably set up. As for on-board VBN techniques, they are able to guarantee lower accuracy than off-board ones, however they have the advantage of being flexible and usable in any environment. Lately, in the literature, there are references on the use of "event-based" cameras, which are used for the development of on-board VBN techniques for robotic applications [84,85]. The advantages that event-based technology offers compared to cameras based on standard technology are mainly related to the very low latency they can guarantee. However, this technology is currently very expensive when compared to classic vision systems. This issue unfortunately widely limits its use.

Another recent technology, usable for precision autonomous navigation, uses multiple UWB sensors, where each module is called *node* [86,87]. Conceptually it is no different from off-board VBN techniques, as only the technology used changes but not the technique itself. Specifically, a constellation of nodes (UWB modules) mounted in a fixed position in the environment is used and a node is mounted on board the drone. This constellation of fixed nodes, joined to the mobile node, is able to triangulate with high precision the position of the UAV within the constellation itself. As for the off-board

VBN techniques, also the localization techniques based on UWB technology are able to guarantee high performance at the expense of the flexibility of the system itself.

2.2 State of the art

In the scientific community, one of the topics of greatest interest in the UAV field concerns autonomous navigation based on computer vision. In the academic field, many works are done through the use of motion capture systems (mocap) [83, 88], as they guarantee the highest performance. Given the high precision that these systems are able to achieve, they are often used to compare different control techniques. This approach allows to establish in a very accurate way the small differences in performance existing between the various control algorithms to be tested on the UAV, providing a powerful measurement tool useful for research purposes.

However, often the researches that are done through the use of mocap systems are not reflected in the industrial sector since most of the navigation algorithms and control techniques that exploit mocap technology can hardly be implemented on drones that must be able to work even in environments without this technology. Recently, thanks to the technological evolution and the proliferation of the open-source board market such as *Raspberry Pi*, *Nvidia Jetson* etc. (able of running operating systems such as Linux on board), the research world has become increasingly interested in on-board VBN techniques [63,81]. Given the possibilities currently offered by technology, interest has shifted to how to solve the problem of autonomous navigation of UAVs without ground assistance.

To investigate this new formulation of the problem, the classic UAV architecture consisting of a microcontroller, sensors connected to it and a receiver capable of receiving commands is no longer sufficient. It is therefore necessary to use a multilevel architecture, composed of both classic microcontrollers and boards capable of processing images and complex algorithms (Raspberry Pi, Nvidia Jetson, etc.). In addition, each board must be able to communicate with the adjacent hardware layer. This aspect highlights the fact that to solve this type of problem the hardware complexity of the drone inevitably increases.

2.3 Multilevel Architecture

A very common autopilot board for UAV robotic applications is the *PixHawk*. Indeed, most of the scientific papers dealing with the development of technology and algorithms for autonomous UAVs use this platform [95]. The reasons are more than one, among the most important are: (i) the PixHawk board was one of the first and most complete open-source platforms for drones; (ii) some custom versions of the firmware give native support for *Robot Operating System* (*ROS*) [88]. The latter is a set of frameworks for robot development and programming.

In fact, as mentioned in the previous section, the development of autonomous UAV applications based on on-board VBN techniques involves the use of multiple hardware levels, which must be able to communicate with each other [88]. In particular, on these UAVs (based on multilevel hardware) an additional high level card is used (which can be a Rasberry PI card) on which ROS is installed. The high-level hardware is then able to communicate with the PixHawk platform through a UART port, which uses the MAVLINK protocol. While there is currently no standard for these systems, it can be generally said that complex tasks such as autonomous navigation and building exploration are performed by the high-level hardware, while the task of attitude stabilization is performed by the PixHawk card.

2.4 Conclusion

From the previous sections it is evident how the technologies applicable on drones are many and in continuous evolution. However, there is no standard plug and play platform on which to develop innovative techniques and algorithms. This forces research groups to develop home-made platforms through which carry out their topics of interest.

The PixHawk board partially solves the problem, giving the possibility to be driven also through the MAVLINK protocol (standard compatible with ROS). However, it cannot be the definitive solution, since it would not be possible to replace the PixHawk autopilot board with another if it were necessary (currently, there are no immediate alternatives for interfacing additional boards to other flight controllers known to the researcher). In this regard, in the next chapter five different UAVs will be shown in detail, which make use of a modular hardware and software architecture, allowing

Figure 2.1: The figure shows the PixHawk open-source autopilot board. This microcontroller is suitably customized for use on drones, in normal use it is able to automatically stabilize the attitude and control the altitude. If required, it can carry out simple automatic missions based on the use of GPS technology.

the simple replacement of individual components.

Any autopilot card can be used with this approach. In fact, as will be explained, the hardware architecture in question will be able to generate the autonomous signals to be sent to the autopilot board in the same standard used by the wireless receivers (normally used to receive the manual commands sent by the human pilot). This method theoretically makes possible to use proprietary controllers, such as the *DJI Naza N3*, which is compatible with standard drone receivers currently on the market.

Chapter 3

Developed hardware platforms

In this chapter we describe the architecture of the developed hardware and all derived versions. The nature of the proposed hardware is designed for autonomous navigation and thus provides an excellent basis to develop computer vision algorithms and advanced methods for navigating complex spaces. This hardware platform, called DART, allows to focus on developing high-level algorithms, leaving low-level tasks such as attitude stabilization to the flight controller. In particular, all the versions of the platform developed over time are shown, starting from the first up to the most advanced version. Each version is associated with a different acronym, this will be useful in subsequent chapters to explain on which versions the various algorithms have been developed and tested. Furthermore, all the technical choices made during the development of the various prototypes are motivated.

3.1 Introduction of the hardware architecture of the prototype designed for autonomous flight

DART project is aimed to investigate if suitable tweaks of standard technologies can be used to implement a completely autonomous high-precision drone, i.e., a drone able to follow a reference trajectory with centimetric precision exploiting only on-board systems without any assistance from the

ground, neither for sensing nor for computing. On a different perspective, it is also intended to measure how this kind of drones is far from mass market. Indeed, low-cost technologies are considered a key factor to boost the market of many innovative drone applications [64, 65].

Autonomy comes from a digital signal mixer that allows the on-board navigation system to override the human commands and to replace the pilot, as long as this latter decides to get the lead back. This way, the navigation system just substitutes the human pilot, and thus, they share the same interface with the drone. The concept behind DART project is to separate the simpler tasks all concerning attitude stabilization from the so-called "smart" tasks related to complex navigation routines, such as the tracking of a complex trajectory in the 3D space. For this purpose it is referred to as "low-level hardware", the flight controller that only performs attitude stabilization and which can be a commercial board. Instead, "high-level hardware" is defined as all the additional and necessarily non-commercial electronics useful for autonomous navigation.

This choice provides a two-fold advantage: (i) it requires the least number of modifications and the smallest customization; (ii) it allows one to focus only on the development of intelligent "high-level" algorithms for computer vision and tracking. Here, it is worth noting that in the DART platform, high and low-level tasks are performed by two different hardware, since attitude stabilization, unlike high-level tasks needs a control high-band to be successfully carried out. The first hardware is able to execute complex calculations at a lower rate (generally, a few hundred Hz) with respect to a standard micro-controller, while the second is optimized just to perform simple calculations but at a very high-frequency (in the order of KHz).

Indeed, the attitude variation is much faster than the dynamics of the drone position. For the reasons shown in the following sections, the solution that we adopted to design the hardware architecture is, to our knowledge, the closest to the actual mass market of UAV technologies. Due to its peculiar nature the automatic pilot is here referred to as "cyber-pilot". Moving backward from the process output up to the inputs, the signal chain can be summarized as follows.

- Engines
- Flight Controller
- Signal Mixer

- Receiver
  - Human pilot
- Navigation System (Cyber-pilot)
  - Sensors
    - IMU
    - Stabilized Camera

Over the time, many hardware upgrades have been made and more advanced versions of the DART platform have also been developed. For clarity we will define an acronym for each hardware version. This will make it easier to explain in later chapters which versions the various algorithms have been tested on.

3.2 DART-1.0 platform

In this paragraph, the first version of the DART platform is introduced, which from now on we define *DART-1.0*. This platform is based on a standard class 250 frame. This means that the diagonal of the drone measures $250mm$, this measurement is usually defined by the position of the motors. After some preliminary tests, emerged as the smallest kind of drones, where one can carve out sufficient space and payload for some additional on-board devices. Besides the basic electronics for engines and batteries, the drone features a $2.4GHz$ remote receiver, and a flight controller, which has the sole task of stabilizing the attitude.

In particular, this version uses a *Raspberry Pi 3B+* with a *Raspicam* camera mounted in fixed position with respect to the frame (all the software for autonomous navigation runs on the Raspberry board). The signal mixing operation is carried out by an *Arduino Nano* board, which is connected between the Raspberry and the low-level flight controller (*CC3D revo*). The hardware architecture and the interactions between the various levels will be explained in detail in the following subsections.

3.2.1 High-level board: Raspberry Pi 3B+

In the first version of the platform, the high-level part has been developed using a Raspberry PI 3 B + board. In fact, this board, unlike Arduino microcontrollers, uses a Linux operating system and is able to process images

coming from a camera. At the beginning of the project it was the best choice, also considering the small size and weight (essential characteristics for use on small drones). The Raspberry board also has an indispensable *General Purpose Input/Output* (*GPIO*) for communicating with the mixer board and the IMU. The specifications of the card are shown below.

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
- Extended 40-pin GPIO header
- Full-size HDMI
- 4 USB 2.0 ports
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- 4-pole stereo output and composite video port
- Micro SD port for loading your operating system and storing data
- 5V/2.5A DC power input
- Power-over-Ethernet (PoE) support (requires separate PoE HAT)

3.2.2 Mid-level board: Arduino Nano

The signal mixing operation (autonomous and manual) is carried out by using an Arduino Nano board (chosen for its small size and low cost), on which a custom firmware has been written. This firmware is able to communicate with the high-level card, receive manual commands (coming from the $2.4GHz$ receiver of the drone) and at the same time generate a second command signal to be sent to the low-level controller (the details of this approach are explained in subsection 3.2.4). The characteristics of the Arduino Nano are shown below.

- Microcontroller: ATmega328
- Architecture: AVR
- Operating Voltage: 5 V
- Flash Memory: 32 KB of which 2 KB used by bootloader
- SRAM: 2 KB
- Clock Speed: 16 MHz
- Analog IN Pins: 8
- EEPROM: 1 KB
- DC Current per I/O Pins: 40 mA (I/O Pins)
- Input Voltage: 7-12 V
- Digital I/O Pins: 22 (6 of which are PWM)
- PWM Output: 6
- Power Consumption: 19mA
- PCB Size: 18 x 45 mm
- Weight: 7 g

3.2.3 Low-level flight controller: CC3D revo

As discussed in section 2.3, currently in the literature, the most widely used flight controller is the Pixhawk, based on the open-source *Ardupilot* project. In the scientific works of the sector, the approach used for the communication of additional cards with this flight controller involves the use of ROS [88]. However, the approach precludes the choice of other flight controllers, since the firmware running on Pixhawk is the only one that natively supports communication with ROS. In fact, the only way to use flight controllers other than Pixhawk is to modify the low-level firmware. Considering that the dimensions of the Pixhawk are considerable compared to the size of our drone, we preferred to design the aforementioned multilevel architecture (high / medium / low-level). Which, in addition to being able to pilot any flight controller available on the market, avoids modifying the low-level

firmware. In this regard, a very small flight controller was chosen, called CC3D-Revo. On this board runs *LibrePilot* firmware, a parallel project to Ardupilot, of open-source type. As already mentioned, this module has the sole task of stabilizing the attitude of the drone with respect to the set-points received from the signal mixer. Since neither the altitude nor the position are controlled by the low-level module, another controller is required to carry out these tasks, as it will be shown in subsection 3.2.4. As a further remark, it is worth noting that the low-level flight controller allows for hovering the drone even in case of faults from the high-level navigation system. The specifications of the flight controller are shown below.

- Processor: STM32F405RGT6 ARM Cortex-M4 microcontroller 168 MHz/1 MB Flash
- Sensors:
  - InvenSense MPU6000 IMU (accel, gyro)
  - Honeywell HMC5883L compass
  - MS5611 barometer
- Power: 4.8 V - 10 V input power provided through ESC connection
- Default Interfaces:
  - 8 PWM outputs (1 - 6 on PWM output pins, 7 and 8 on Flex-IO / RC input port).
  - RC input (requires PPM/SBUS) on Flex-IO / RCInput ports CH3 pin.
  - analog to digital inputs for battery voltage and current monitoring (set pins 12,11 in params), more adcs possible on arbitrary pins.
  - GPS (SERIAL3) on Flexi Port.
  - Telemetry (SERIAL1) on Mainport.
  - USB (SERIAL0) port.
  - SWD Port for flashing and debugging, including 3.3 V output for optional periphereals.
  - MMCX antenna connector for integrated HopeRF RFM22B 100 mW 433 MHz.
  - OPLink port.

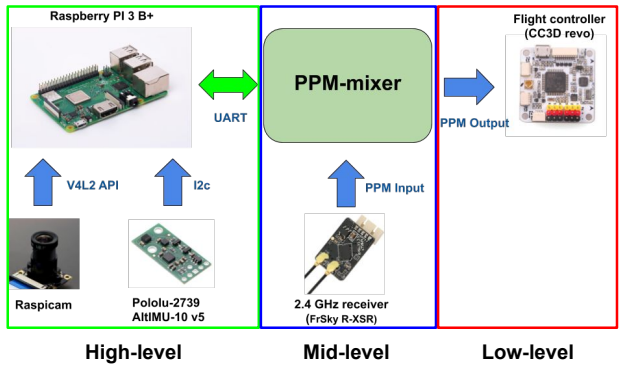

Figure 3.1: Hardware configuration and dependencies. The PPM-mixer board is an interface between the low-level flight controller and the Raspberry PI board, on which the high-level navigation software is implemented.

3.2.4 Hardware architecture

The architecture of the hardware is shown in Fig. 3.1, where one can observe the interconnections between all components. The Raspberry PI 3 B+ board is connected with the Raspicam camera and the IMU, sensor units from which the vision and inertial data are obtained. The idea is to run the autonomous navigation software on the Raspberry board generating the set-points for attitude and motors thrust, which are sent to the low-level flight controller. Currently the low-level board has the sole task of stabilizing the attitude by following the set-points from the cyber-pilot (high-level board) or from the human one.

Between the Raspberry PI and the low-level, the custom signal mixer board is employed, also called mid-level board. The mixer board takes in input the attitude set-points coming from the Raspberry (by using bidirectional UART protocol to convey the data stream) and the manual set-points coming from the 2.4 GHz receiver through the *Pulse Position Modulation* (PPM) protocol [16]. As output the mixer generates a second PPM signal, that is obtained by properly elaborating the inputs. It is important to note that the use of the PPM standard guarantees a maximum command band of $44Hz$, which for human commands are more than sufficient but can represent a bottleneck for the cyber-pilot. However, in the following paragraphs it will be shown how in subsequent versions of the platform this limitation

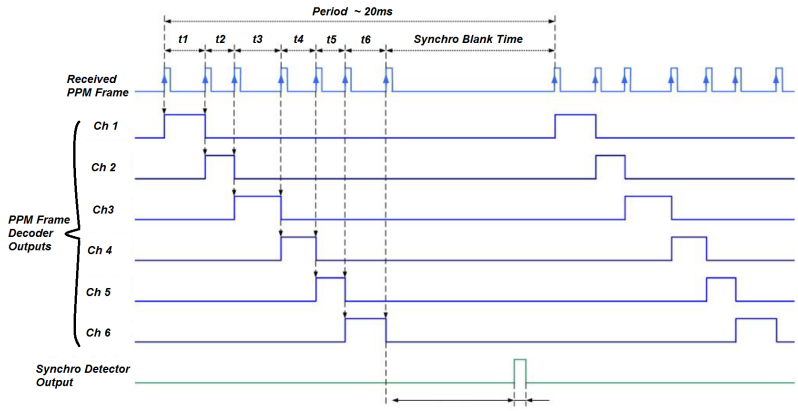

Figure 3.2: Example of the PPM modulation implemented on the mixer board. The information part relating to a single channel is contained within the time interval between two consecutive peaks. Once the status of each channel has been sent, a time value greater than the maximum value that the single channel can assume is expected. This is done to separate one frame of pulses from the next.

has been eliminated. The resulting device is able to switch safely between autonomous and manual flight modes. The autonomous flight mode is still a hybrid mode where the cyber-pilot commands are overlapped to the manual controls.

The concurrence between manual and autonomous controls also allows for the rectification of possible anomalous behaviors of the drone along the chosen trajectory, thus improving the safety. In Fig. 3.3 we show the block diagram of the algorithm representing the operating logic of the mixer board. In details, the algorithm takes as input signals the manual PPM and the autonomous contribute transmitted via UART protocol. The PPM signal is decoded in a vector of time intervals, which are expressed as integer numbers where each unit increment equals to 1 μs. Each time interval identifies a specific set-point that has been converted into the low-level flight controller format. Instead the autonomous contribute is already coded as time intervals by the Raspberry PI, in order to decrease the computational effort of the mixer and, thus, to improve the rate of the output signals. Then, we sum together the time intervals coming respectively from the manual PPM signal

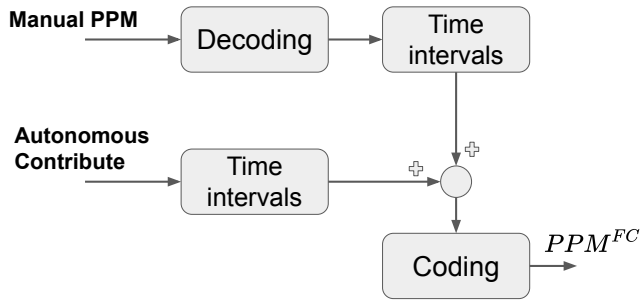

Figure 3.3: Functional scheme of the mixer board. The two input command signals, converted in time intervals, are merged into a single signal at the output of the board. We have named this operating logic as *Hybrid Mode*.

and the autonomous contribute, by also taking into account their sign. The resulting time intervals are finally re-encoded in a PPM signal, which is then sent to the low-level flight control board.

In other terms, one can see the mixer board as an interface between the high-level navigation system (managed by the Raspberry board) and the attitude stabilization algorithm, which is implemented on the low-level controller. The main advantage of this architecture is to generate command signals directly in the standard of flight controllers for drones. This allows one to use any flight controller available on the market without making any changes to the low-level firmware, eliminating the possibility of introducing catastrophic and uncontrolled bugs.

3.2.5 Sensors units

The sensors units employed in this project are composed by (i) the IMU connected to the Raspberry PI through the I2C serial bus, and (ii) a Raspicam camera for the purpose of computer vision. In particular, the IMU is a Pololu AltIMU-10 v5 that implements several standard sensors commonly used to estimate the pose of smart devices: Gyroscope, accelerometer, compass, and altimeter. It is worth saying that the algorithms, which we are going to present in the following chapters, only use the camera, the gyroscope and the accelerometer, but not the other sensors. Instead, regarding

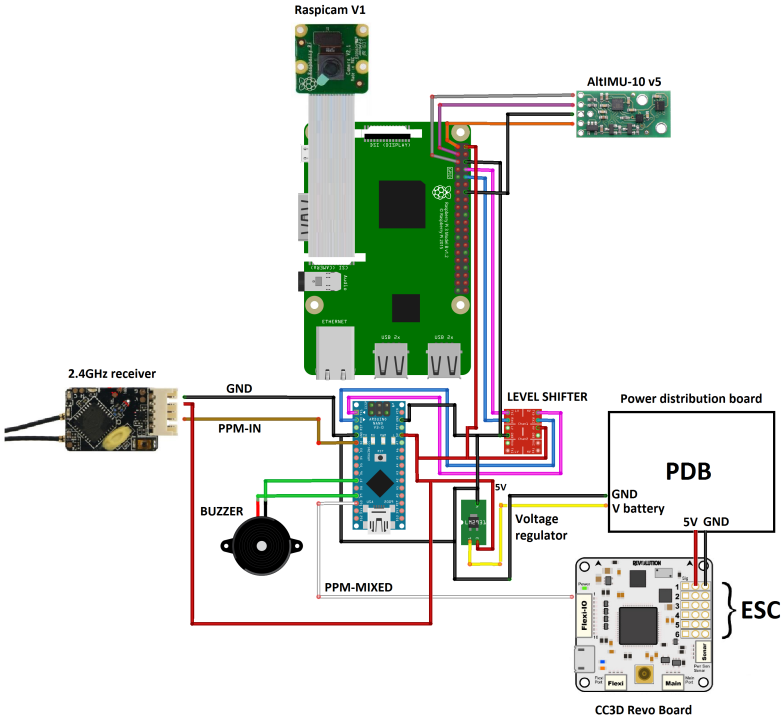

Figure 3.4: Connection diagram of the high and mid-level components with the low level controller.

the Raspicam camera V1, it is connected to the Raspberry PI through a *Mobile Industry Processor Interface* (*MIPI*) cable.

3.2.6 Wiring of high-level electronic components

In this subsection we focus briefly on the wiring of high-level electronic components. Fig. 3.4 shows the wiring diagram of how the high and mid-level blocks are connected to the low-level block. It is important to note that the pins of the Raspberry are not $5V$ tolerant, for this reason it is preferable not to connect it directly to the Arduino Nano (which pins work at a logic level of $5V$). For this purpose, a bidirectional level shifter was used, capable of adapting the different logic levels of the two cards ($3.3V$-$5V$). In the wiring diagram of Fig. 3.4 a buzzer connected to the Arduino nano board is also

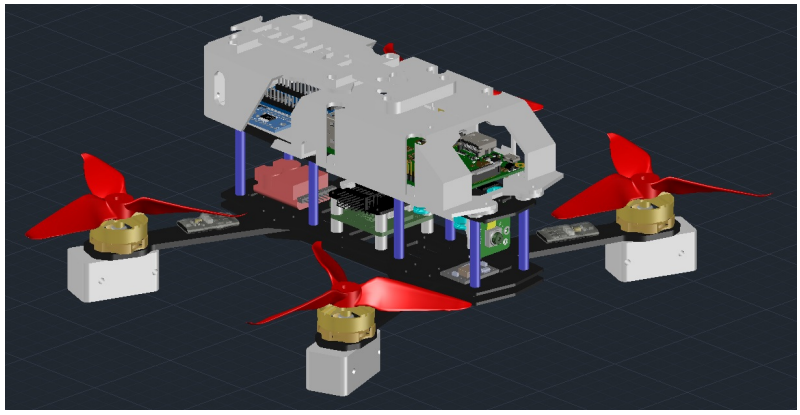

Figure 3.5: CAD view of the complete model related to the DART-1.0 platform. The custom parts were made using 3D printing.

visible, it generates an acoustic signal if the Raspberry board fails during autonomous flight mode.

3.2.7 Mechanical structure

The DART-1.0 body frame is mostly composed by laminated standard carbon fiber parts which improve the rigidity of the frame and reduce its weight. The body frame hosts other custom parts which have been 3D printed at low density, leaving empty spaces inside the material to make it lighter. These parts serve to assembly the additional hardware necessary for autonomous navigation and computer vision tasks. In this hardware version, the motors used are 2205-2550kV, combined with 5-inch propellers and with a 3-cell lipo battery pack (the choice of these three components will be detailed in paragraph 3.2.8). Fig. 3.5 shows a CAD view of the first version of the prototype, complete with all the custom parts designed for the assembly of high-level electronics. It is important to note that in this first version of the platform, the artificial vision camera is fixed with respect to the drone. This implies a greater entity of the vision noise due to the mechanical vibrations generated by the rotation of the motors. To reduce vision noise, silicone dampers were fitted between the camera body and the UAV frame. This allowed the decoupling of the two parts, almost totally canceling the vibrations captured by the camera. Fig. 3.6 shows in detail the method of fixing the camera

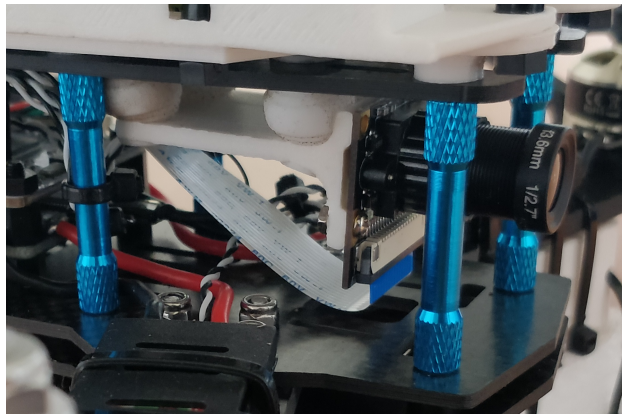

Figure 3.6: Support for fixing the "Raspicam" camera for computer vision, complete with silicone dampers.

to the UAV frame using the silicone dampers [17]. Fig. 3.7 shows the real prototype, once assembled with all the custom parts.

3.2.8 Brushless motors for UAV

It is worth briefly investigating the identification code of the motors used, thus clarifying the logic on the basis of which they were chosen. Please note that the identification code of the engines used in the DART-1.0 platform is 2205-2550kV. The first part of the code, that is the number 2205, identifies the diameter and height of the stator in millimeters respectively. Increasing the stator diameter also increases the diameter of the bell-shaped rotor where the propeller is fixed. This implies that by varying the diameter of the bell-shaped rotor, the torque of the motor also varies. This happens because the arm of the applied lever varies, for this reason propellers of a specific diameter must be coupled with motors having an adequate lever arm.

If this combination is not done carefully the result is low motor efficiency [18], which is usually measured as $\frac{g}{W}$. Instead the second part of the identification code (in the specific case 2550) defines the number of rotation per minute for each Volt supplied to the engine. This quantity is measured in kV and is defined by eq. 3.1.

$$kv = \frac{rpm}{V} \tag{3.1}$$

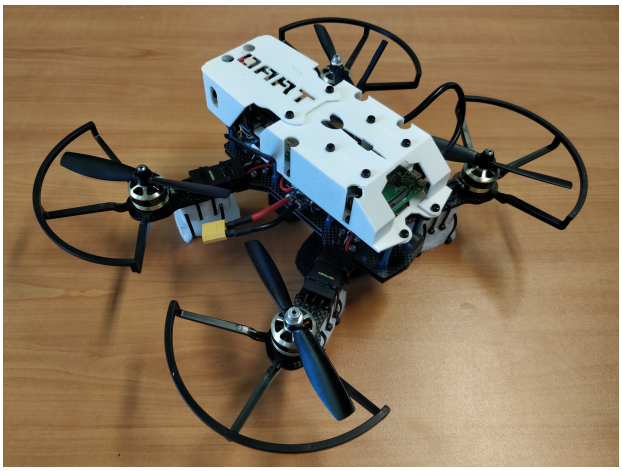

Figure 3.7: A picture of the DART-1.0. To increase safety during indoor flight, para-propellers have been fitted.

The number of kV together with the diameter of the propeller determines the maximum thrust obtainable from the motor-propeller couple. Normally the motor data are provided by the manufacturer through specific tables. These tables show the maximum thrust and power consumption values referred to some types of propeller. It is good practice to size the maximum thrust of the UAV considering that the *thrust/weight* ratio must be at least 2/1. The DART-1.0 platform including the 3-cell battery has a weight of $1Kg$, so the motors should be sized to be able to provide at least $2Kg$ of thrust. Fig. 3.9 shows the table provided by the motor manufacturer where the row relating

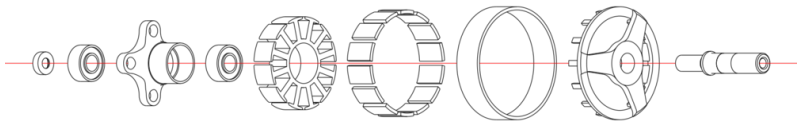

Figure 3.8: Exploded diagram showing the architecture of a brushless motor for UAV. The bell-shaped rotor is shown including the permanent magnets. This architecture allows to obtain a high torque, since it maximizes the lever arm.

to our use case is highlighted in red.

MOTOR TECHNICAL DATA:

	NO LOAD		ON LOAD				LOAD TYPE
VOLTAGE	CURRENT	SPEED	CURRENT	Pull	Power	EEP	Battery/prop
V	A	rpm	A	g	W	%	
11.1	0.7	28416	2.8	200	31.1	6.4	LiPo X 3/5045
			5.8	400	64.4	6.2	
			13.2	570	146.5	3.9	
			3.1	200	34.4	5.8	LiPo X 3/6045
			7.6	400	84.4	4.7	
			19.1	750	212.0	3.5	
			4.5	200	50.0	4.0	LiPo X 3/T5045
			11.3	400	125.4	3.2	
			15.5	600	172.1	3.5	
			3.6	200	40.0	5.0	LiPo X 3/T6045
			9.3	400	103.2	3.9	
			21.8	700	242.0	2.9	

Figure 3.9: Table provided by the manufacturer of the motors used in the DART-1.0 platform. The operating conditions in our setup when the engine is providing the maximum thrust is highlighted in red.

As can be seen in our configuration, the maximum thrust of a single engine is $570g$ which multiplied by 4 reaches $2280g$, therefore the thrust/weight relationship is respected. Another interesting data shown in Fig. 3.9 is the *EEP*, which is the value of the engine efficiency in a specific operating condition. In the specific case we have an EEP value of $3.9\frac{g}{W}$ relative to the maximum throttle, which represents one of the best values among those tabulated. This indicates that the motor-propeller-battery combination is correct.

3.3 DART-1.1 and DART-1.2 platforms

To reduce the vision noise due to the UAV maneuvers, it was decided to introduce a 2-DOF gimbal. The gimbal is able to decouple the attitude of the drone from that of the camera, keeping it leveled with respect to the pitch and roll angle. However, the gimbal adds weight to the UAV and requires additional space to be properly installed. This raises two problems: (i) motors with 5-inch props are no longer the choice that guarantees the best efficiency; (ii) the frame of the UAV must be modified for the installation

Item No.	Prop	Throttle	Thrust (G)	Volts (V)	Amps (A)	RPM	(W)	Efficiency (G/W)
2508 KV1200	GF7042 Two-blade 16V	50%	330.70	15.56	3.26	9475	50.71	6.52
		55%	390.77	15.53	4.01	10264	62.22	6.28
		60%	466.76	15.51	4.92	11071	76.25	6.12
		65%	539.35	15.49	5.99	11882	92.76	5.81
		70%	620.45	15.48	7.23	12705	111.87	5.55
		75%	738.57	15.79	9.01	13815	142.32	5.19
		80%	830.75	15.83	10.54	14535	166.79	4.98
		85%	913.29	15.80	12.07	15251	190.62	4.79
		90%	1003.60	15.76	13.89	15970	218.80	4.59
		95%	1093.58	15.72	15.82	16643	248.70	4.40
		100%	1182.52	15.68	17.89	17305	280.52	4.22

Figure 3.10: Table provided by the manufacturer of the motors used in the DART-1.1 platform. The operating conditions in our setup when the engine is providing the maximum thrust is highlighted in red.

of the gimbal; The addition of the gimbal in fact implies a weight increase of at least $280g$, consequently the constraint of the thrust/weight ratio of 2 to 1 is no longer respected. For this reason it is necessary to increase

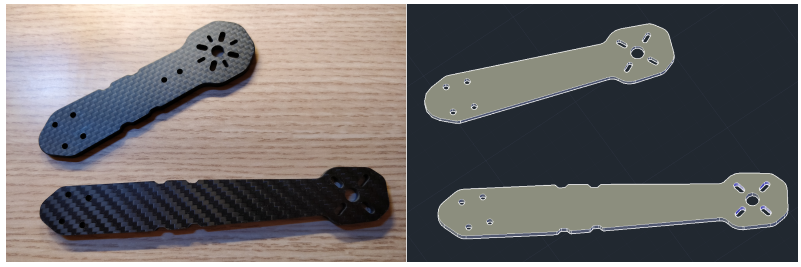

Figure 3.11: In the image it is possible to observe the two versions of arms, where the custom arm is the longer one. The finished carbon fiber arms are shown on the left side of the figure (standard version and customized version). On the right side of the figure there is a CAD view of the two versions of the arm. For the realization of the customized arm, a carbon fiber plate suitably worked by means of a CNC milling machine was used.

the maximum thrust that the UAV can generate [18]. For this purpose in the DART-1.1 platform the motor-propeller combination has been changed, passing from the old 2205-2550kV motors with 5-inch propellers, to the new 2508-1200kV motors with 7-inch propellers. This new type of engine having

a greater lever arm than the previous one allows the mounting of propellers with a larger diameter that provide a much higher thrust than the previous configuration. Since the number of kV is lower than that of the old motors, the battery was also replaced with one with a higher voltage. In other words, we have switched from a lipo with 3 cells ($11.1V$) to a lipo with 4 cells ($14.8V$). Fig. 3.10 shows the table provided by the manufacturer for the configuration adopted in the DART-1.1 platform. The current configuration makes it possible to generate a maximum thrust for each engine equal to $1.182Kg$ (with an efficiency coefficient equal to $4.22\frac{g}{W}$), which multiplied by 4 corresponds to a maximum thrust of $4.728Kg$. Since the weight of the DART-1.1 platform is $1.370Kg$, the thrust/weight constraint is largely respected. The frame of the DART-1.1 platform has also undergone changes, the most important was the creation of custom arms for the motors, able to support the 7-inch propellers. The new arms are longer than the original ones and increase the size of the frame taking it from $250mm$ to $350mm$. In

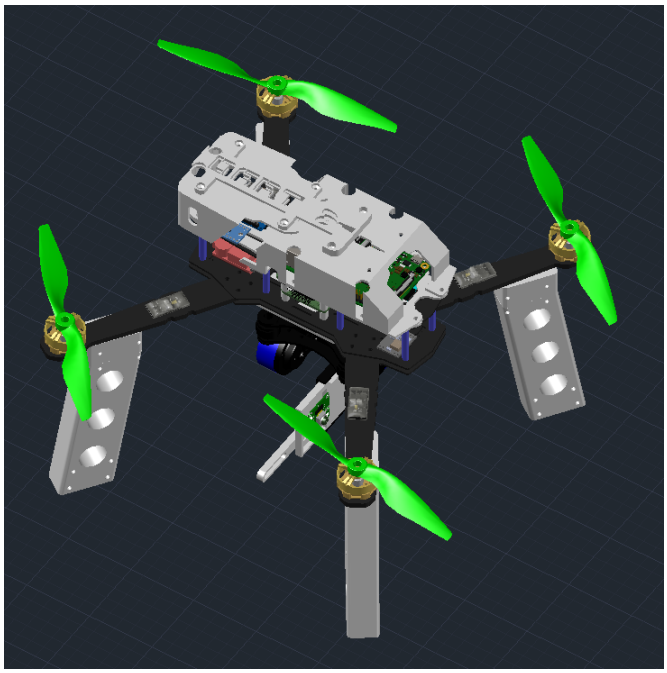

Figure 3.12: CAD view showing the DART-1.1 platform in all its parts.

Fig. 3.11 it is possible to see a CAD view of the two versions of arms, while in Fig. 3.12 the CAD view of DART-1.1 is shown. In addition to designing the support necessary for the correct fixing of the gimbal to the frame and the support necessary to adapt the "Raspicam" camera, it was also necessary to draw pins to lift the frame from the ground. In fact, as can be seen in Fig. 3.13, the size of the gimbal is considerable and it was necessary to mount it under the drone. Also in Fig. 3.11 the two versions of the finished arms are shown. As already mentioned, the custom version is the longest one and was made from a 4mm thick carbon fiber sheet. Once the CAD model of the arm was finished, a *Computer Numerical Control* (*CNC*) cutter was used to process the carbon plate, thus obtaining the finished piece. The Fig. 3.13 shows the gimbal fixed under the drone once the update is finished, as can be seen the fixing method to the drone frame is equivalent to the CAD model.

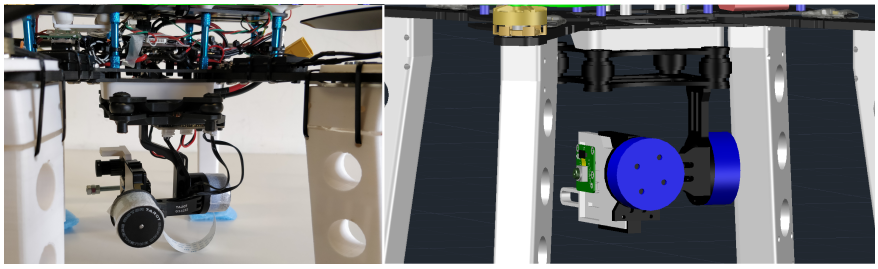

Figure 3.13: The figure shows the engineering done to be able to mount the 2-DOF gimbal under the UAV. in fact it is important to underline that commercial gimbals are designed to be mounted on UAVs with dimensions much larger than ours. Normally no part engineering is required for correct assembly. Please note that this image refers to the DART-1.1 platform.

The electronic architecture is the same as the one presented in paragraph 3.2, that is, it is the same as the DART-1.0 platform. However, the gimbal used in the DART-1.1 platform is a 2-DOF gimbal. This means that it can stabilize the roll and pitch angle, but it cannot stabilize the yaw angle. The latter is in fact fixed with the UAV frame. It can be easily deduced that following variations in the drone yaw angle there may be vision noise directly coupled with the horizontal axis (this phenomenon will be explored in the chapter on experimental tests). To improve the performance of the vision system by further reducing the vision noise, we then switched to a 3-DOF

gimbal that also stabilizes the yaw angle. In fact, a 3-DOF gimbal is able to decouple the yaw angle of the camera from that of the drone. This version of the platform will hereafter be called DART-1.2. To adapt the new gimbal to 3-DOF the fixing system to the frame has been modified as shown in the CAD view of Fig. 3.14.

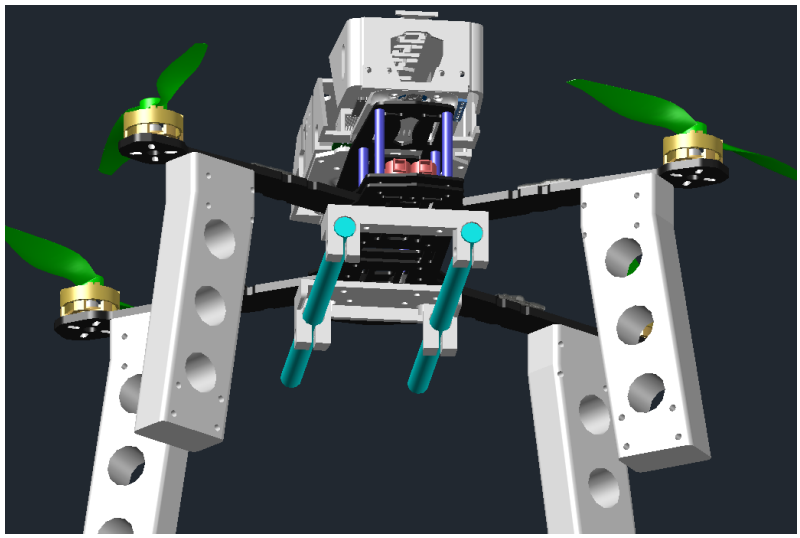

Figure 3.14: CAD view of the DART-1.2 platform showing the 3-DOF gimbal attachment method. The two aluminum tubes (visible in blight lue) improve the stiffness by further decreasing vision noise.

Here it is possible to observe the two supports (front and rear) that hold two aluminum tubes to which it is possible fix the new gimbal, as shown in Fig. 3.15. This method, in addition to being faster in the assembly phase, guarantees a higher stiffness than the method of fixing the 2-DOF gimbal of the DART-1.1 platform (using a single plastic support).

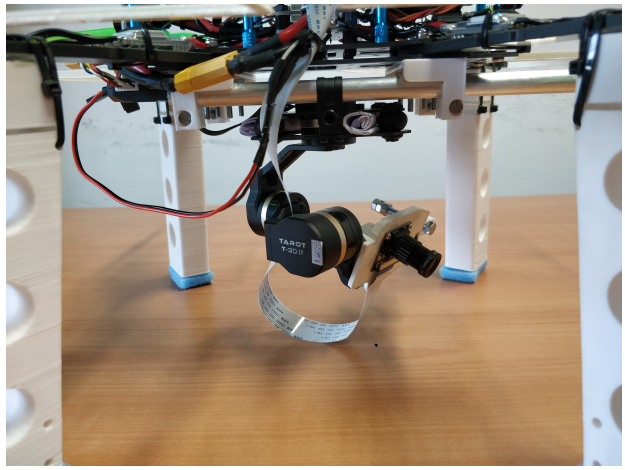

Figure 3.15: 3-DOF gimbal mounted on the DART-1.2 platform.

In Fig. 3.16 it is possible to see the finished DART-1.2 platform complete of the 3-DOF gimbal. Compared to the CAD model, para-propellers (also 3D printed) have been added for indoor flight.

Figure 3.16: DART-1.2 platform.

3.4 DART-2.0 and DART-2.1 platforms

In this paragraph, we show the second version of the DART platform, which uses a completely different technology than the previous versions. In fact, the Raspberry PI 3B+ board has been replaced with an *Nvidia Jetson Nano* board. Another technological advancement of this platform is due to the implementation of a stereoscopic camera produced by Intel, called *Intel Realsense t-265*. Even the mid-level has been improved, in fact the PPM-mixer (which in previous versions was realized through the use of an Arduino Nano board) has been replaced by a new SBUS-mixer, which can guarantee a higher communication band with the low-level.

3.4.1 Intel Realsense t-265 camera

The Intel Realsense t-265 is a stereoscopic camera, which means it has two distinct optical sensors. This system simulates human vision and is able to locate itself in the environment [21, 22]. It also implements a *Bosch*

Figure 3.17: Intel Realsense t-265 stereoscopic camera capable of providing its position and attitude at a frequency of $200Hz$ and with a latency of 5-6ms.

BMI055 IMU that provides the inertial data to which the camera is subjected. Through a *Simultaneous Localization and Mapping* (*SLAM*) algorithm [19, 20], which runs on board the camera, it is able to provide its position with a frequency of $200Hz$ and a latency of 5-6ms. This feature is very important, as it does not use the computing power of the high-level board (Nvidia Jetson Nano), which remains free to perform other tasks. In addition, it can provide the video stream of the two fisheye lens sensors at a frequency of $30Hz$. Since the hardware that makes up the camera is of excellent quality and the electronics are specially designed to run the *SLAM* algorithm, the precision it is able to provide on the position estimation is

sub-centimeter. This makes it an excellent tool to be employed on drones (currently representing the state of the art for these systems), as it is an order of magnitude more accurate than GPS sensors and is capable of operating in indoor enviroments (where GPS sensors cannot work). It is also necessary to consider that unlike GPS sensors, which are able to provide only the position on the plane (2D position), systems that use computer vision are able to provide the complete state vector (3D position). Thanks to these sensors and algorithms that exploit computer vision, today it is possible to use drones to carry out advanced missions that were unthinkable until a few years ago. The data that this stereoscopic camera can provide is not limited to just the position vector and the inertial vector. The complete list of all the data that it can provide (if requested) and the relative update frequencies is shown below.

- 3D position vector. - $200Hz$.
- 3D vector of linear velocities. - $200Hz$.
- 3D vector of linear accelerations. - $200Hz$.
- Quaternion of attitude. - $200Hz$
- Angular velocities. - $200Hz$
- Angular accelerations. - $200Hz$
- Tracker confidence (variable whose value indicates the degree of reliability of the estimated data). - $200Hz$
- Video stream of left and right lens. - $30Hz$

3.4.2 High-level board: Nvidia Jetson Nano

The Nvidia Jetson Nano development platform has a much higher computing power than the Raspberry board and has slightly larger dimensions (which makes it possible to mount it on small drones). It is currently the smallest development platform that has a dedicated *GPU* [23, 74]. In fact, the most interesting feature of the Jetson platform is the dedicated GPU with 128 cuda cores, making it ideal for developing computer vision applications. It has 4GB of *LPDDR4* RAM and is mounted on a carrier-board through a special connection slot. The carrier-board is useful for development, plus

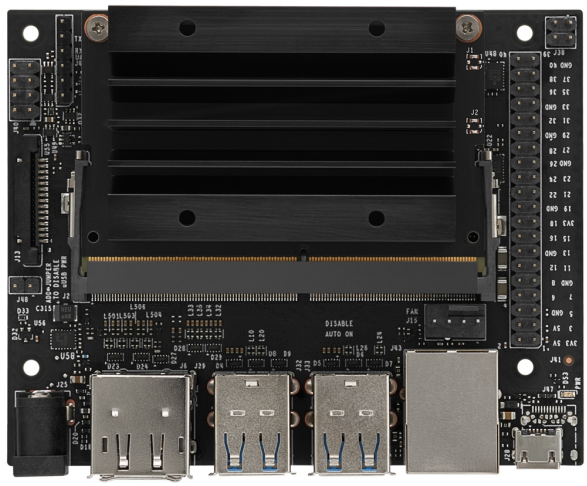

Figure 3.18: Nvidia Jetson Nano module complete with development carrier board.

it implements a GPIO compatible with the Raspberry standard. Like the Raspberry board, the Jeston Nano carrier-board also has a MIPI connector, which allows mounting of *CSI* (Camera Serial Interface) cameras (such as the Raspicam camera). The specifications of this development platform produced by Nvidia are shown below.

- GPU: NVIDIA Maxwell 128 core.
- CPU: ARM A57 quad-core a 1,43 GHz.
- RAM: LPDDR4 4 GB 64-bit 25.6GB/s.
- Storage space: microSD.
- Encoder video: 4Kp30/4x 1080p30/9x 720p30 (H.264/H.265).

- Decoder video: 4Kp60/2x 4Kp30/8x 1080p30/18x 720p30 (H.264/H.265).
- Connectivity: Gigabit Ethernet, M.2 Key E.
- Camera: 1 or 2 connectors MIPI CSI-2 (depends on the carrier board version).
- Display: HDMI and DP.
- USB connector: 4 USB 3.0, USB 2.0 Micro-B.
- Others: 40-pin (GPIO, I2C, I2S, SPI, UART), 12 pin (power supply and related signals, UART), fan 4-pin.
- Size: $100mm * 80mm * 29mm$

3.4.3 Mid-level board: Teensy 4.0

One of the major limitations of previous DART platforms was the mixer board. In fact, in previous versions an Arduino Nano board was used, which has a limited computing power, having a clock of only $16MHz$. Furthermore, the PPM protocol was used (which can guarantee a maximum band of $44Hz$) to communicate with the low level. The use of this protocol has two disadvantages: (i) The information is contained in the time intervals between consecutive signal peaks. This means that the information that must be sent will have an uncertainty due to the computing power of the card that generates the signal. In fact, the greater the computing power, the greater the temporal precision of the signal peaks generated by the board. Considering that the Arduino Nano board has little computing power, the PPM signal generated by it has a non-negligible amount of noise; (ii) The signal bandwidth is not high and depends on the number of channels to be transmitted. In addition to the poor computing power of Arduino Nano, there are at least two other limitations due to the use of this board: (i) the presence of only one serial port available (therefore it is possible to use the UART protocol to communicate with only one device and it is not possible to use the serial monitor during debugging); (ii) it has only two pins that can handle interruptions and one of them is already used for receiving the PPM signal coming from the 2.4GHz receiver. For these reasons, in the new version of the drone, the Arduino Nano has been replaced by a Teensy 4.0 micro-controller, which is currently one of the fastest micro-controllers available on the market. In

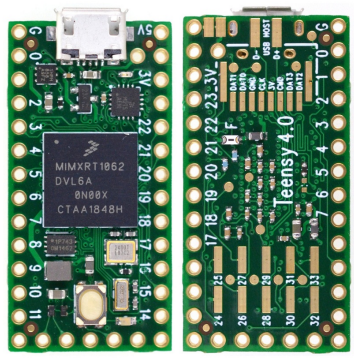

Figure 3.19: The figure shows the upper and lower side of the Teensy 4.0 micro-controller.

fact it has a ARM Cortex CPU with a clock frequency of 600 MHz, all its pins can handle interruptions and unlike the Arduino Nano it has 7 UART ports. The specifications of this micro-controller are shown below.

- ARM Cortex-M7 at 600 MHz
- 1024K RAM (512K is tightly coupled)
- 2048K Flash (64K reserved for recovery and EEPROM emulation)
- 2 USB ports, both 480 MBit/sec
- 3 CAN Bus (1 with CAN FD)
- 2 I2S Digital Audio
- 1 S/PDIF Digital Audio
- 1 SDIO (4 bit) native SD
- 3 SPI, all with 16 word FIFO
- 3 I2C, all with 4 byte FIFO
- 7 Serial, all with 4 byte FIFO

- 32 general purpose DMA channels
- 31 PWM pins
- 40 digital pins, all interrrupt capable
- 14 analog pins, 2 ADCs on chip
- Cryptographic Acceleration
- Random Number Generator
- RTC for date/time
- Programmable FlexIO
- Pixel Processing Pipeline
- Peripheral cross triggering
- Power On/Off management

Based on this new hardware it was decided to replace the PPM standard with a more performing *SBUS* (*Serial BUS*) protocol, which represents the most advanced communication protocol used on UAVs. The SBUS standard is based on the use of one of the two pins of a UART port (Rx-Tx) for sending channels. The advantages of this protocol are mainly three: (i) it is a digital protocol (unlike the PPM standard) and therefore it is not affected by noise. (ii) the usable communication band is greater than the PPM protocol and is $100Hz$. (iii) There is a higher resolution than the PPM protocol. In fact, in the PPM protocol it is possible to distinguish a maximum of 1000 values (the time range is 1000μs and the unitary value is 1μs), while in the SBUS protocol it is possible to send up to 2047 different values (11-bit integers). An SBUS frame can transmit up to 16 channels and is made up of 24 bytes. The first byte represents the header, the bytes between the second and the 22nd contain the status of the 16 channels, while the 23th byte contains information about packet loss and failsafe status, finally the last byte represents the closing of the frame. Fig. 3.20 shows the shape of an SBUS frame.

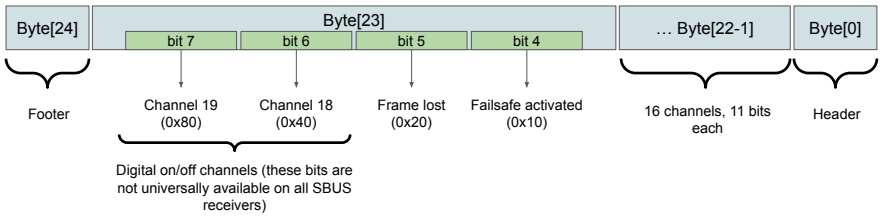

Figure 3.20: The figure shows the frame architecture according to the SBUS standard, which unlike older standards is a digital protocol. It is currently the best choice for sending commands to the flight controller, as in addition to guaranteeing a greater useful band, it is not affected by noise.

3.4.4 Hardware architecture

In the DART-2.0 platform, the hardware architecture has evolved from previous versions. This is not only due to the replacement of the hardware, but is also due to the implementation of faster and more performing communication protocols such as the SBUS protocol. This protocol, unlike the less performing PPM, also allows bidirectional communication. In fact, it uses only one of the two pins of the UART port, leaving the other free to be used for the transmission of telemetry (through appropriate communication standards, such as the FrSky *SmartPort*). This means that the low-level can send telemetry data (which may contain, the estimated attitude, height estimated by the on-board barometer, any error messages, battery status, flight mode, etc.) at the mid-level. Consequently, it is possible to send all this information at the high-level, which in addition to providing a good level of redundancy, can add useful information, (such as the battery status) on the basis of which behavior policies of the autonomous part can be defined. Fig. 3.21 shows the hardware architecture scheme of the DART-2.0 platform, where all the connections with the sensors can be seen. In the high-level block the stereoscopic camera and the Raspicam are connected (through the Realsense and GStreamer libraries), the latter in the DART-2.0 version is facing the ground. Since the Realsense t-265 camera has an IMU inside, the Pololu AltIMU-10 v5 board has been eliminated.

However, it was thought that in some experimental conditions it might be interesting not to use the Intel stereoscopic camera, for this purpose a second MPU-6050 IMU (which is not normally used) was connected to the system.

As can be seen from diagram 3.18 this IMU is no longer connected directly to the high-level module, but is connected to the mid-level. This choice was made with the intention of improving attitude estimation performance, since the inertial data from an IMU have dynamics of the order of KHz. As presented in the paragraph 3.1, these data are more easily processed by a micro-controller, which by its nature is able to work at high frequency (unlike the high-level module). This allows to exploit the entire information band of inertial data (which is concentrated at high frequencies).

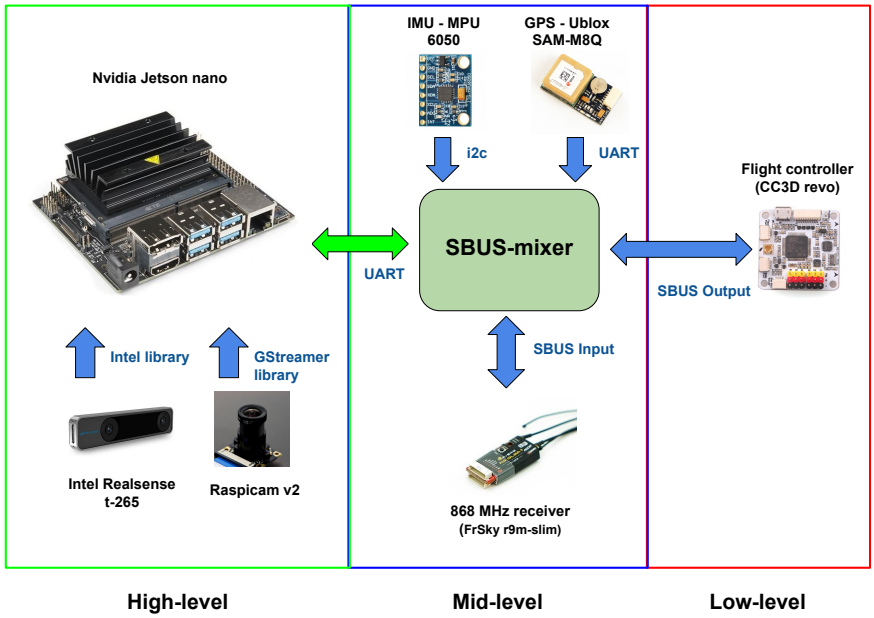

Figure 3.21: Hardware architecture of the DART-2.0 platform.

A GPS has also been added to the mid-level block, which can be useful for providing an absolute position reference in open environments. As is known, unlike inertial sensors, GPS work at lower frequencies (at most $20Hz$). In this regard, one might think that this sensor should be connected directly to the high-level module. However, hardware devices such as Nvidia Jetson cards have a limited number of I/O pins. Therefore, to get around

this limitation, it was decided to use the mixer board also as a HUB for the connection of additional sensors. The adoption of a mid-level managed by a micro-controller guarantees many advantages not achievable with the classic architectures reported in the literature. In fact, the multilevel architecture of the DART platform allows you to exploit both the advantages offered by the Jetson/Raspberry modules (designed for computer vision and artificial intelligence applications) and the advantages offered by micro-controllers (designed to work at high frequencies and have many GPIO pins).

3.4.5 Wiring of high-level electronic components

Fig. 3.22 shows the diagram of the connections between the mid and high-level blocks with the low-level. As already mentioned, in this version the IMU was connected directly to the mixer through an i2c port. 4 UART ports were used to generate the 2 SBUS signals (input and mixed output), to manage communication with the high-level and to allow connection with a GPS. Also in this version the alarm buzzer is used to signal a possible failure of the high-level module. The stereoscopic camera, as it is possible to observe in the diagram, is connected via USB 3.0 cable, while the Raspicam via the usual MIPI cable. It is interesting to note that the *First Person View FPV* camera, useful for ground video transmission, is not connected to the high-level module. However, an *On Screen Display OSD* board is used which communicates with the low-level controller, to overlay some useful information on the video stream (which is transmitted to the ground). Such as, for example, the battery status and any error messages relating to the low-level controller. Finally, it should be noted that the level shifter present in previous versions of the platform has been eliminated. This is because unlike Arduino Nano, the Teensy board works with a logic level of $3.3V$ (which is the same logic level as the Jatson Nano board).

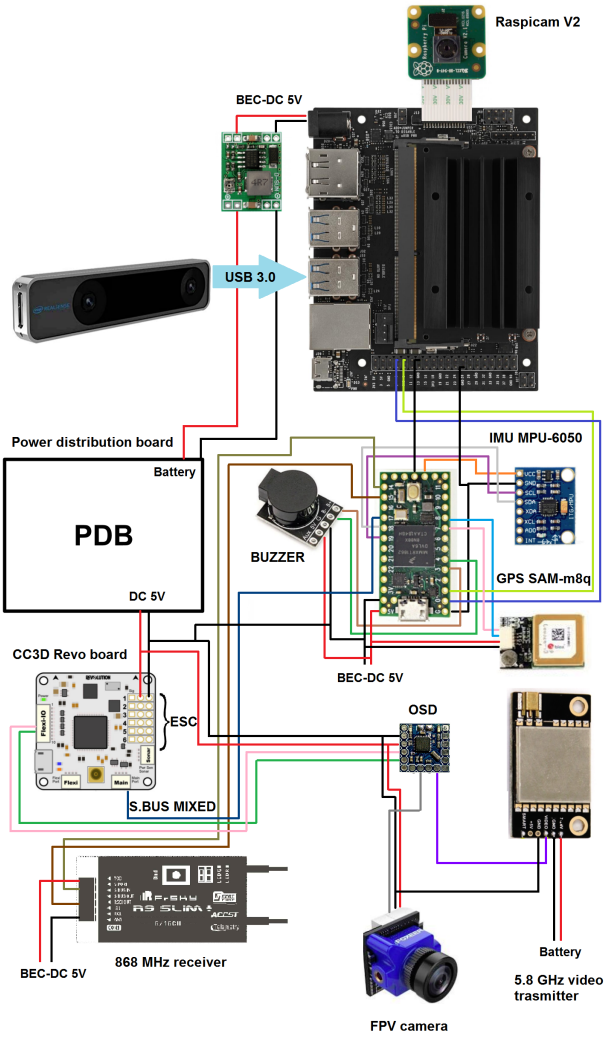

Figure 3.22: Wiring diagram of the DART-2.0/2.1 platform. Where all the connections between the various boards and sensors are visible.

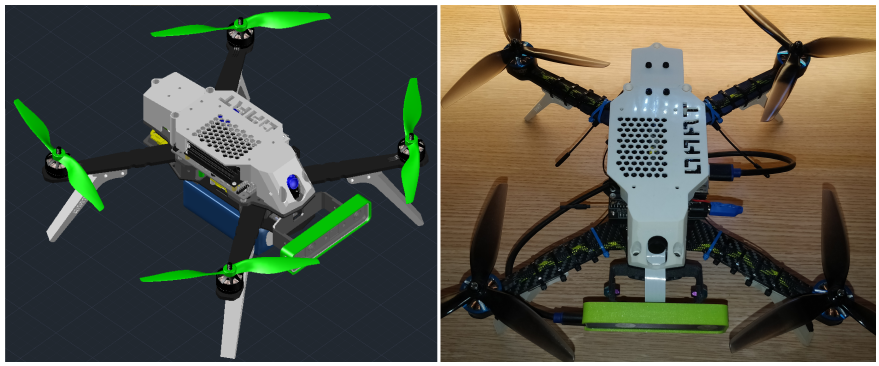

Figure 3.23: CAD and real view of the DART-2.0 platform, in this image it is possible to observe the engineering work that has been done for the assembly of the new hardware. It should be noted that the small blue camera serves the sole purpose of transmitting the view of the drone to the ground and is not connected to the high-level block. It is commonly referred to as an FPV camera.

3.4.6 Mechanical structure

In the DART-2-0 and DART-2.1 platforms the carbon fiber parts and the motors are the same as in the previous version. What changes is how these parts have been assembled together to make it possible to fit the new hardware. Fig. 3.23 shows the DART-2.0 platform, where significant changes can be appreciated compared to previous versions. In particular, the case has been redesigned to accommodate the new hardware. In addition, a plate was designed capable of fixing the carrier board of the Jetson Nano module to the drone frame (this plate was then made through 3D printing). In the front it is possible to observe the Intel Realsense t-265 camera, which is fixed with respect to the frame.

In fact, in this new version the Gimbal has been eliminated, since the Intel camera does not suffer from problems related to the vision noise. This saved the weight of the gimbal and the relative space under the drone, used for mounting the battery. The Intel stereoscopic camera (used as a sensor position) has very short focal lenses (fisheye), which are effective for locating very lateral objects but useless for detecting distant objects. For this reason it was decided to move the Raspicam camera to the front position, in order to have the possibility of receiving streaming video from cameras with different

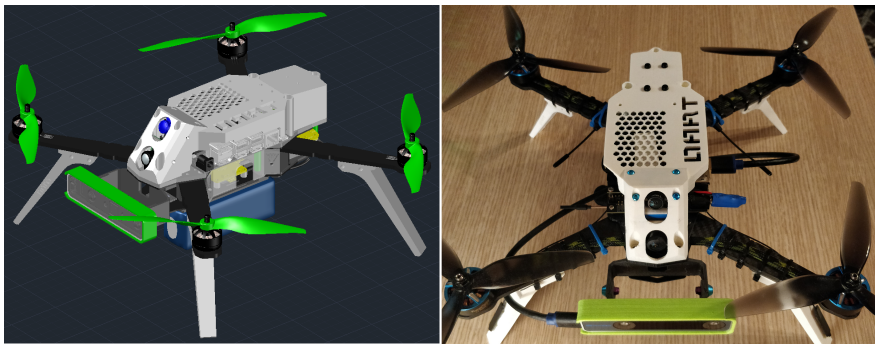

Figure 3.24: In the left part of the figure the CAD view of the DART-2.1 platform is visible, while in the right part the final prototype is visible once all the parts have been made and assembled. As you can see, the Raspicam camera has been moved to the front position (under the FPV camera).

focal lengths. This increases the field of view of the drone, which remains able to see very lateral objects, but can also detect distant objects, which would not have been visible with the use of the stereoscopic camera alone. This latest variant of the platform is called DART-2.1 and is shown in Fig. 3.24.

3.5 Conclusions

In this chapter, five different versions of the DART platform and their development phases have been presented. The technologies they use were explained and the multilevel hardware architecture was shown, which allows the separation of the low-level controller from the high-level electronics. Furthermore, given the presence of several cards on board the UAVs, it was necessary to program protocols capable of allowing communication between the various hardware levels. For this purpose, the different communication standards compatible with the flight controllers available on the market have been studied. In fact, the biggest bottleneck for UAVs based on multilevel architectures is represented by the communication band width with the low-level controller. It should be noted that this bottleneck does not exist when a human pilot drives the drone, as it is not able to generate commands with frequencies higher than $2 \sim 3Hz$, unlike the "cyber-pilot", who is able to send commands at much higher frequencies. Currently the best performing

communication standard that can be used to communicate with low-level flight controllers is the SBUS protocol, which guarantees $100Hz$ of bandwidth, more than double compared to sending 8 channels via PPM protocol. Finally, this approach was fundamental to obtain a solid platform on which was then possible to develop the autonomous navigation and computer vision algorithms that will be shown in the following chapters.

Chapter 4

Software architectures and algorithms for autonomous navigation based on artificial vision

This chapter shows the software architectures and algorithms that have been developed on the first 3 versions of the platforms (DART-1.0, DART-1.1 and DART 1.2). In particular, the critical issues that have led from time to time to the development of alternative techniques are explained. All techniques make use of computer vision for the creation of an odometry system [63], which allows the drone to locate itself in the environment (no technique uses GPS). At the same time, the software architectures that have been developed over time are also explained. Finally, a comparison will be made between the performances obtained through the use of the various algorithms [33].

4.1 Introduction

The main elements of the navigation system, i.e., the computer vision system, the multi-PID controller, and a *Madgwick* sensor fusion filter are here discussed. Before continuing, we stress the fact that we use two different

representation for the drone attitude: In section 4.5.2, for the purpose of implementing the Madgwick sensor fusion filter, it is more convenient at the software level to describe the attitude via the quaternion formalism, while the representation with rotation matrices is adopted in Sec. 4.5 to simply get the estimate of the drone position. Since to maximize the performance of the software (which runs on cards with limited computing power), we have to prefer solutions that allow to reduce as much as possible the complexity of the software implementation.

4.2 Computer vision system

A computer vision algorithm is used to provide the drone with absolute references for position and attitude. This way, the drone can be accurately driven in the 3D space in a desired manner by referring the time-variation of its pose (position and orientation) to these fixed references captured from the environment.

In the first three versions of the UAV prototype, the computer vision system is the main element of the drone navigation system. The vision algorithm, that manages the acquisition of the images and that takes care of stitching together the acquired frames, is set to detect one or more known markers in the environment. The algorithm estimates the relative pose of the drone with respect to the markers, and, by knowing their absolute pose, is able to infer the absolute pose of the drone, as well. The camera has been tested both as a built-in device inside the body frame and mounted on a stabilized gimbal (see Section 4.5), as shown in Figure 3.15. It is worth noting that the lens frame and the one referring to the center of the thrust do not usually have the same orientation.

The marker used in this implementation of the drone are boards featuring black and white squares. Their recognition is achieved, according to a standard practice [67], by evaluating for each pixel of the image the local magnitude of the pixels gradient, given by point-to-point differences in the pixel colour scale. Then, the gradient direction is evaluated, and pixels with similar gradient directions and magnitudes are grouped into sets by using graph-based methods. A line segment is eventually fit to each clustered pixel set. Such sets of pixels identify in the image edges from which the algorithm searches for the correct marker sequence, whose position is defined in pixel coordinates by the two-dimensional vector $[u, v]^T$, as shown in Figure 4.1.

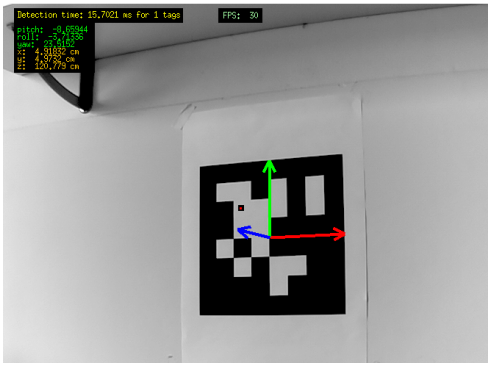

Figure 4.1: Detection and tracking process: the marker reference frame is shown in red, green and blue for x, y and z axes, respectively. The computer vision algorithm runs on Raspberry PI with a frequency rate of $30Hz$ and provides the marker attitude and position in the lens frame [63].

The position of the marker is located in the 3D space by referring $[u, v]^T$ with respect to the lens frame. In such a frame, the coordinates

$$Q^c \equiv [x^c, y^c, z^c]^T \tag{4.1}$$

of the marker in camera frame (denoted by means of the superscript c in the formulas) are computed through the following relations (in this regard, see also Ref. [63]) that also take into account the barrel distortion:

$$\frac{x^c}{z^c} = \frac{(u - u_0)(1 + k_{ud} r^2)}{\rho_x} \tag{4.2}$$

$$\frac{y^c}{z^c} = \frac{(v - v_0)(1 + k_{ud} r^2)}{\rho_y} \tag{4.3}$$

where

$$r^2 = \frac{(u - u_0)^2}{\rho_x^2} + \frac{(v - v_0)^2}{\rho_y^2}, \tag{4.4}$$

and $[u_0, v_0]^T$ denotes the coordinates in pixel of the principal (reference) point in respect of which the image is calibrated. Instead, the parameters ρ_x e ρ_y are the ratio between the focal length and the size of the pixel, while k_{ud} is the parameter that corrects lens distortions. Note that k_{ud}, ρ_x and ρ_y are intrinsic camera parameters, which are commonly obtained through an

iterative calibration process involving the acquisition of frames of a known image in different poses. The calibration process has been performed off-line and is based on acquiring at regular time intervals at least 5 images of a chessboard with known dimension in different poses.

As further remark, observe that, if the geometrical properties (shape and dimension) of the marker are known, it is possible to retrieve additional information also with a monocular camera, such as the distance z_m between the camera lens and the marker or the relative orientation among the marker and lens frames.

Therefore, the computer vision module is definitely able to provide information both on the position vector and on the attitude vector of the marker with respect to the drone. Hence, if the marker has a known position and attitude, also the position and orientation of the drone can be straightforwardly obtained. From here on, we will define with

$$\Phi \equiv [\varphi, \theta, \psi]^T \tag{4.5}$$

the attitude vector of the marker with respect to the camera lens. Note that we have removed the subscript c from each element of Φ for the sake of simplicity of notation.

4.3 PID-based control system

The autonomous driving module, i.e., the core of the cyber-pilot running on the Rasperry board, computes the commands in the same form of those coming from the remote control receiver, i.e., as proper reference values for roll, pitch, yaw and thrust. During the preliminary implementation stage of the project, they were conceived to maintain the drone over time in a desired position, denoted as $\overline{Q} \equiv [\overline{x}, \overline{y}, \overline{z}]^T$, with zero yaw relative angle $\overline{\psi} = 0$, intending that the drone was facing the marker. More generally, the computer vision system provides both relative orientation and position with respect to the marker frame. This information is further integrated by means of a fusion algorithm with data coming from the on-board IMU to improve the estimate from the sole image processing, as shown in next paragraph. The final result of the sensor fusion process is a refined estimate of the drone position $Q \equiv [x, y, z]^T$. Four different algorithms have been tested to compute Q with different performance. As illustrated in Figure 4.2, the information on the errors of the drone pose, respectively given by the

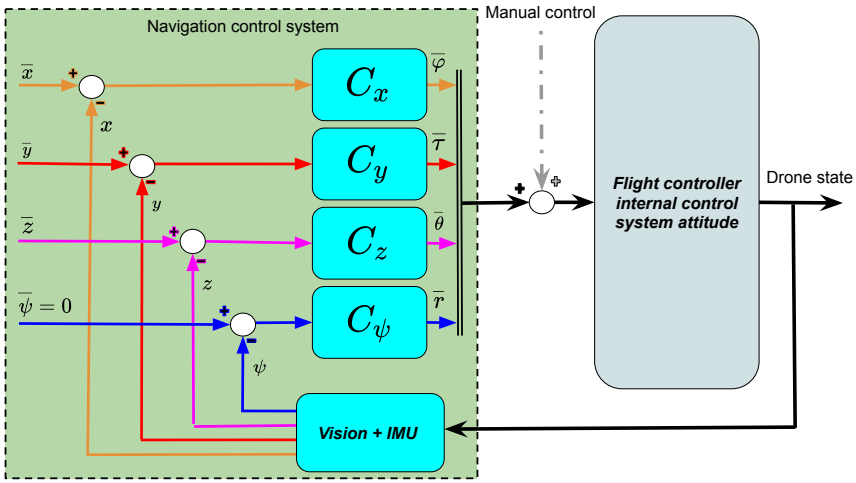

Figure 4.2: Control architecture. The navigation control system is a feedback control loop composed by four PID controllers, respectively C_x, C_y, C_z and C_ψ one for each pose degree of freedom, and a position estimation module fusing together stream data from the IMU and the Raspicam camera. C_k, with $k \in \{x, y, z\}$, have as input signals drone position errors and provide in output an attitude reference signal to be sent to the low-level module. Instead, C_ψ takes in input an error signal on ψ and returns a reference for the yaw angular velocity.

differences $\overline{Q} - Q$ and $\overline{\psi} - \psi$, are used to feed four distinct (decoupled) PID controllers (C_x, C_y, C_z and C_ψ), which respectively generate the driving commands that the low-level module uses as references inputs to control roll and pitch angles, yaw angular velocity, and thrust. One can observe that the control architecture of the autonomous driver is composed by simple modules that individually act on a different pose degree of freedom. This decoupled control architecture does not directly consider the mutual interconnections among the components of the drone pose, as the fact, for example, that a change in the pitch modifies the net thrust. However, this solution for the control system has to be preferred for its reliable implementation, robustness and low computational cost. Moreover, also note that, in a first phase, each PID controller has been tuned after numerical simulations based on a simplified model of the actual drone. Then, experiments have been carried

out thanks to a large number of repeated flights.

4.4 Software architectures

In this paragraph we show two software architectures which have been developed for the first 3 versions of the platform (DART-1.0, DART-1.1 and DART-1.2). In particular, the first version uses a single-task architecture (which was developed first), while the second, more advanced than the first, uses a multi-task architecture.

4.4.1 Single-task architecture

The first software architecture developed is also the simplest, in fact it is based on a single task model where the code is sequential and separated into functions. In this model, the biggest limitation is the existence of a bottleneck on the frequency of software execution. This bottleneck is imposed by the slower function, in this case by the computer vision function, which is the most onerous from a computational point of view. For this reason, the maximum execution frequency on the Raspberry PI 3 B + platform cannot exceed 35 Hz. Furthermore, this sampling frequency is not managed by the operating system scheduler and therefore the perfect periodicity of the "main task" is not guaranteed. This phenomenon is shown in Fig 4.4, where peaks on the sampling time can be observed. However, as shown in Fig 4.5, this architecture is not completely sequential, but the computer vision function generates four threads (one for each Raspberry cpu core) to improve the performance of this part of the software. These threads are managed directly by the libraries required for the computer vision, which in this implementation are the *VISP* (*Visual Servoing Platform*). It is important to note that the IMU function is performed before the computer vision function, this is done since the data extrapolated from the vision function is affected by a latency, while in the inertial ones the latency is negligible. To minimize this time gap it was decided to perform the functions in the order shown in Fig 4.5.

4.4.2 Multi-task architecture

After the implementation of the single task architecture, it was decided to move to a more performing multi-task architecture. The advantages are many, but the most important are 3: (i) the sampling time of each single

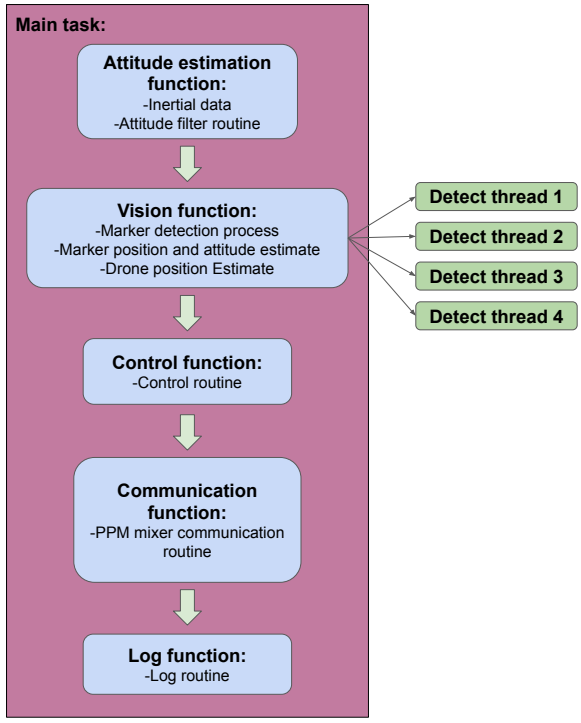

Figure 4.3: In the figure it is possible to see the single task software architecture. It is important to note that the *attitude estimation function* is performed before the computer vision function. This is because the latency of the images is higher than that of the data coming from the IMU.

task is decided by the operating system scheduler; (ii) the flexibility of the software is increased as it is possible to dynamically manage which tasks must be performed; (iii) it is possible to separate tasks that need to be performed at a higher frequency from others that don't need them. In fact, the inertial data coming from the IMU must be performed at high frequency since their information content is contained there, while the vision data can be processed more slowly. In this software architecture, the drone navigation system stack comprises five distinct principal tasks. They are managed by a standard Linux scheduler set as SCHED FIFO (meaning "first input first output scheduler"), such that threads with the same priority are managed

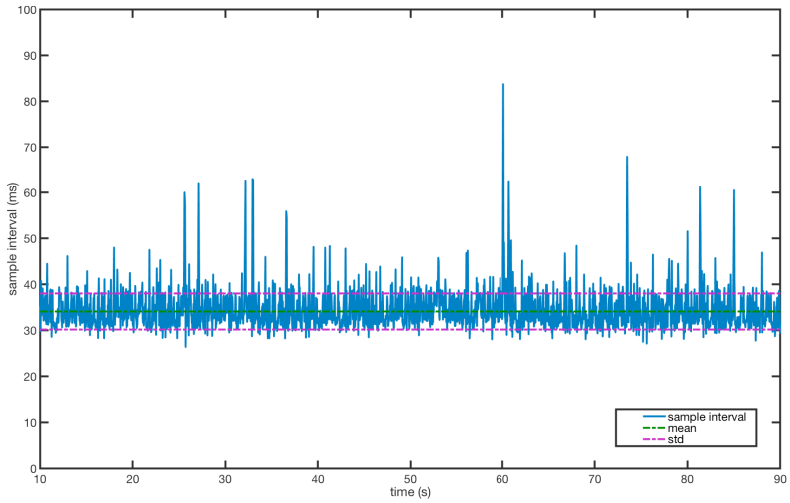

Figure 4.4: The figure shows how the sampling time obtained with the single task architecture is not perfectly periodic. This phenomenon occurs because the software is not managed directly by the operating system scheduler.

with FIFO policy. This mode can be used to implement real-time policies and it can be activated only with root permissions. It is usually adopted to reduce the time variability of the execution period of individual tasks, which is a desired feature when working with sampled processes. All the software in the higher level of the navigation system is written in C++ language and has been implemented on the Raspberry Pi 3 model B+ platform, as already described in Chapter 3.

The *main task* manages the computer vision system and it is responsible to carry out the processes for the marker detection and the estimation of its position and attitude. In particular, the computer vision is an aperiodic task that works at about 30 FPS (frame per second) with a resolution of 660 $\times$ 660 pixels. After many experimental tests on different configurations, this working condition has been found a good compromise between the number of FPS (i.e., the computational load) and the precision of the results. To improve the performance, the main task has been parallelized in four sub-tasks, one for each Raspberry core, which process part of the same frame at the same time. This way, a better use of the available calculation resources is reached, and the computations are performed faster.

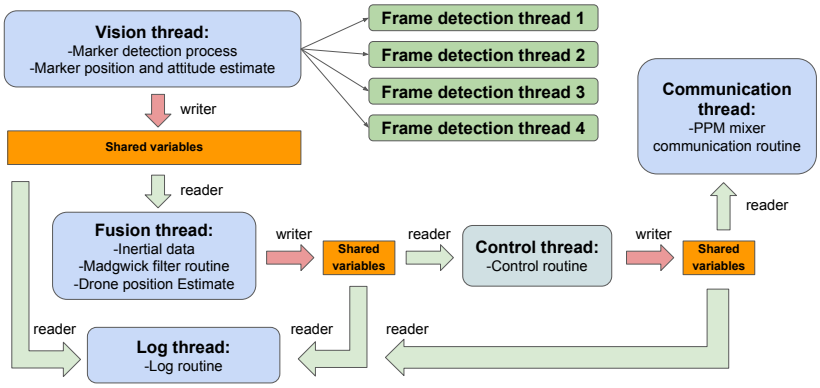

Figure 4.5: The software architecture is composed by 5 concurrent tasks: the vision, Fusion, log, control and communication threads. To increase the performance, a readers-writers synchronization method is used. The latter allows simultaneous access to multiple reading tasks of the same shared variables. In addition, thanks to the implemented software architecture enabling the parallel calculation, all the cores of the Raspberry platform are exploited, thus reducing overall the computation time.

The *second task* is a thread that manages the IMU and performs (i) the acquisition of the inertial data from the IMU itself, (ii) the Madgwick filter (explained in subsection 4.5.2) routine for the estimation of the drone attitude, and (iii) the drone position estimation process. Since the inertial data from the IMU is informative at high frequencies (unlike computer vision), the frequency of this thread has been increased as much as possible. In particular, the fusion thread is periodic and works at $200Hz$.

Instead, the *third task* is the thread for the control of the position trajectory: Given the desired (reference) trajectory, which in the simplest case the drone has to track point-by-point with possible constraints on the velocity profile, the control routine uses the position displacement error to generate the attitude set-points to be sent to the signal mixer. The control samples are computed only when the autonomous flight mode is activated. This control routine is managed as a periodic task forced to work at $22Hz$. The frequency is decided by the PPM protocol: Since the maximum bandwidth ensured by the PPM protocol is $44Hz$, the control routine cannot occupy more than $22Hz$, i.e., half of the maximum bandwidth, so to avoid aliasing

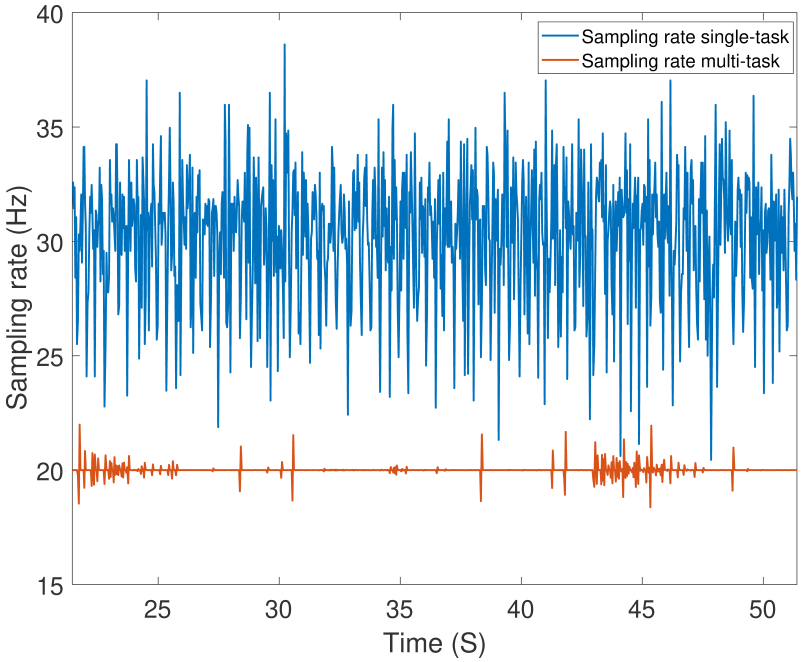

Figure 4.6: The figure shows the trend of the sampling frequency of a thread running in the multi-task architecture (whose execution is decided by the Linux scheduler) and the trend of the global sampling frequency of the single task architecture (which it is not managed by the scheduler). As can be seen from the graph, the thread running in the multi-task architecture, and which is managed by the scheduler, has a more stable trend.

effects. However, as it will be shown in the next section on experimental results, the band of the dynamics of the drone position is comparable with the working frequency of the control task, thus making this architecture sufficient to accurately control the drone, especially when gimbal suspension are implemented. On a technical note, the integral component of the PID controllers is activated only at specific instants, i.e., whenever the autonomous flight mode is enabled (event reported by a specific variable). This choice is motivated by the need to avoid discontinuities caused by the wind-up effect, when the autonomous mode is activated.

As *fourth task*, a communication thread coordinates the transmission of

the set-points to the mixer, that, in the autonomous mode, will send them to the flight controller. To reduce the computational load of the mixer, the attitude set-points are first converted into time intervals (see also Section 3.2.4) and then sent to the other boards. The communication task is a periodic routine, working at $44Hz$ to exploit all the available bandwidth provided by the PPM protocol.

Finally, the *fifth task* generates the mission log by saving all the data in a txt file for offline processing. This task has the least priority and due to the access to the memory it works almost periodically at $20Hz$.

As a final technical remark, let us stress that the tasks architecture is based on a *readers-writers* synchronization method, where several tasks, which need to read shared variables (reader task), can access them simultaneously. Having implemented this distinction between readers and writers tasks, the software has better performance with respect to the standard case (*mutual exclusion* synchronization method, i.e., readers and writers can access to the shared variables separately and one-at-a-time) and, thus, the data exchange process between threads is sped up.

4.5 Position estimation methods

In this paragraph, four different methods which have been tested to estimate the drone position exploiting the marker frame as reference are presented. The algorithms that are explained in the following paragraphs have been designed and developed for the different DART platforms. For this purpose, the following table shows on which hardware and software platform the various algorithms have been tested and developed.

Algorithm	Hardware platform	Software architecture
FCF-CF	DART-1.0	single-task
FCF-MF	DART-1.0	single-task, multi-task
SCF-2DOF	DART-1.1	single-task, multi-task
SCF-3DOF	DART-1.2	multi-task

In particular in the FCF-M algorithm it will be shown how the performance of the madgwick filter changes according to the type of software architecture used, which consequently determines the maximum sampling frequency.

4.5.1 FCF-CF position estimation method

The first algorithm developed, called Fixed Camera Frame Complementary Filter (FCF-CF) [63], Was implemented on the DART-1.0 platform. This algorithm only uses the computer vision system and the gyroscope. The schematic representation of the algorithm is depicted in Figure 4.7. To merge the information flows from the two sensors, we adopt a complementary filter that works according to the following relation:

$$\widehat{\Phi}_{k+1} = \lambda(\widehat{\Phi}_k + T\,\Omega_k) + (1-\lambda)\Phi_k \ , \tag{4.6}$$

where λ is a real number belonging to $[0,1]$, T is the actual sampling period, $\Phi_k \equiv [\varphi_k, \theta_k, \psi_k]^T$ is the attitude vector from the vision system and

$$\Omega_k = [\omega_x, \omega_y, \omega_z]^T \tag{4.7}$$

is the vector of the angular velocities around the three main drone axes coming from the gyroscope. The new attitude estimate $\widehat{\Phi}_{k+1}$ at the discrete time instant $(k+1)T$ provided by Eq. (4.6) (from now on $\widehat{\Phi}_{k+1}$ will be abbreviated with $\widehat{\Phi}$) can now be employed in the following coordinates transformation returning the estimate of the drone position:

$$\begin{aligned} \widehat{Q}^f &= \begin{bmatrix} x^f \\ y^f \\ z^f \end{bmatrix} = R_{\mathrm{xyz}}(\widehat{\Phi}) \begin{bmatrix} x^c + x_{\mathrm{cb}} \\ y^c + y_{\mathrm{cb}} \\ z^c + z_{\mathrm{cb}} \end{bmatrix} \\ &= R_{\mathrm{XYZ}}(\widehat{\Phi})\,[Q^c + Q_{\mathrm{cb}}] \ . \end{aligned} \tag{4.8}$$

In Eq. (4.8), the rotation matrix

$$R_{\mathrm{xyz}}(\widehat{\Phi}) = \begin{bmatrix} c_\varphi c_\psi & c_\varphi s_\psi s_\theta + s_\varphi c_\theta & -c_\varphi s_\psi c_\theta + s_\varphi s_\theta \\ -s_\varphi c_\psi & -s_\varphi s_\psi s_\theta + c_\varphi c_\theta & s_\varphi s_\psi c_\theta + c_\varphi s_\theta \\ s_\psi & -c_\psi s_\theta & c_\psi c_\theta \end{bmatrix} \tag{4.9}$$

transforms the coordinates defined in the body frame into a fixed frame, while the vector $\widehat{Q}^f$ is the estimate of the drone position in fixed frame. Instead, Q_{cb} is a constant offset vector that takes into account the distance between the camera and the center of thrust of the drone. In this regard, by defining Q^{c} as the position of the marker in camera frame with respect to the center of thrust, Eq. (4.8) can be rewritten as

$$\widehat{Q}^f = R_{\mathrm{xyz}}(\widehat{\Phi})Q^{\mathrm{b}} \ . \tag{4.10}$$

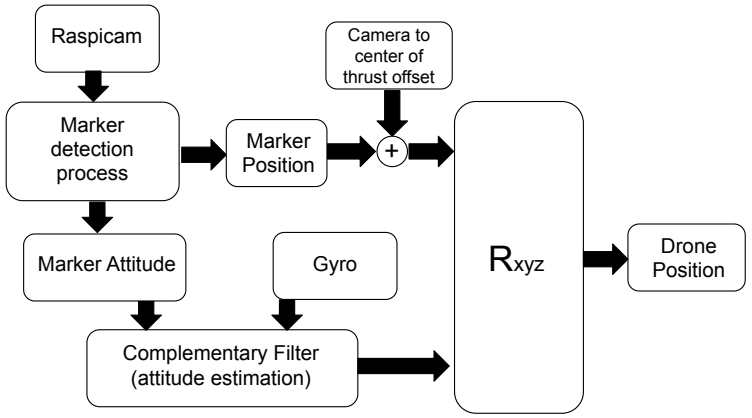

Figure 4.7: Schematic representation of the algorithm "Fixed Camera Frame Complementary Filter" (FCF-CF). The method adopts a complementary filter that fuses the stream data from the computer vision system and the gyroscope (here, the accelerator is not used). This allows to recover the information on the dynamics of the drone that is separately lost (not detected) by the two sensors. The complementary filter returns the estimate $\widehat{\Phi}$ of the drone attitude vector. In this way, by rotating of such an estimated angles the position of the marker (rectified by the offset vector Q_{cb}), one can also derive the estimate of the drone position.

It is important to observe that in this first algorithm (FCF-CF) the camera is mounted on the drone in a fixed position. Thus, no auxiliary system stabilizing the attitude of the camera, as a gimbal suspension, is used. This entails the drawback of persistent noise sources affecting the data stream coming from the Raspicam camera. Indeed, the coupling among the pose components makes each disturbance reflect on all of them. Moreover, the accuracy of the attitude provided by the camera degrades very quickly as the distance between the marker and the drone increases. Indeed, the estimate of the marker attitude depends on how its planar projection is warped within the image. Therefore, at greater distance (of the order of meters) its dimensions within the scene shot by the camera are smaller and the informative content of the data decreases very quickly. Also note that the degradation of the estimation accuracy, respectively for the position and attitude, is not uniform as a function of the distance to the marker. Therefore, for the FCF-CF

algorithm, taking the attitude from the vision system is a limitation that, however, has been overcome in this paper by the algorithms in the following. In conclusion, the FCF-CF algorithm is simple to implement and has low computational cost, but it is expected to be very noisy and, then, to suffer from distance.

4.5.2 Madgwick sensor fusion filter

The navigation system implemented on the DART-1.0 drone works without explicitly modeling the dynamics of the UAV. This choice is mainly dictated by the following two reasons. First, the computational cost has not to exceed a certain threshold in order to not overload the Raspberry processor. Second, the aim is to ensure that the response of the drone to control pulses is as fast as possible, as well as the convergence of the control error.

In the chosen architecture, all the information about the pose of the drone needs to be extrapolated solely from the on-board sensors data stream. In the subsequent algorithms it was chosen to replace the complementary filters with the Madgwick filter [68], that together with the Mahony filter [69, 70] it represents the state-of-art to efficiently fuse the data coming from the accelerometer and the gyroscope within the IMU, respectively for the tracking of the translational and rotational DOFs. On one hand, the gyroscope evaluates the angular velocities of the portion of the UAV on which the IMU is mounted. The measured angular velocities are referred to the frame chosen as reference for the IMU. In principle, the corresponding orientation of the drone could be computed by integrating over time the angular velocities; however, this solution is usually highly discouraged due to the error originating from such a calculation. On the other hand, the accelerometer indirectly measures the gravitational field of the earth [71], taken as absolute reference. Also the information from the accelerometer is affected by noises, and this especially holds true when the sensor is moving.

The Madgwick sensor fusion filter estimates the orientation of the drone by optimally fusing the data stream from the accelerometer and the gyroscope. The orientation processed and returned by the filter is described by the quaternion representation, as e.g. adopted in [72, 73]. The quaternion is a vector with four elements and generally describes the orientation of a coordinate frame with respect to another. For example, the relative orientation between the coordinate frame A and B by the angle α around the generic

axis $r \equiv [r_x, r_y, r_z]^T$ can be represented by quaternion

$$\begin{aligned} q_B^A &\equiv [\cos(\theta/2),\, r_x \sin(\theta/2),\, r_y \sin(\theta/2),\, r_z \sin(\theta/2)]^T \\ &= [q_1,\, q_2,\, q_3,\, q_4]^T \end{aligned} \tag{4.11}$$

that by definition is of unitary length. To each quaternion is uniquely associated the rotation matrix R_B^A that rotates the coordinate frame A towards B according to a sequence of at least 3 rotations of the so-called Euler angles around the x, y, z axes.

In the quaternion formalism, the angular velocities measured by the gyroscope are arranged in the quaternion

$$\omega_{\text{IMU}} \equiv [0, \omega_x, \omega_y, \omega_z]^T \ . \tag{4.12}$$

Thus, if we solely use the data coming from the gyroscope, the orientation of the drone, expressed in the IMU coordinate frame that in turn is referred to the North-East-Down (NED) coordinates, at the k-th discrete time instant is equal to

$$q_{g\,\text{NED}}^{\text{IMU}}[kT] = \hat{q}[(k-1)T] + \dot{q}_{\text{NED}}^{\text{IMU}}[kT]\, T \ , \tag{4.13}$$

where T is the sampling period, $\hat{q}[(k-1)T]$ denotes the estimate of the orientation at the previous time instant, and $\dot{q}$ is the quaternion derivative. For the specific case of the gyroscope, the quaternion derivative is just given by the quaternion product (for more details on the quaternions algebra the reader can refer e.g. to Refs. [68, 72, 73]) between the estimate $\hat{q}[(k-1)T]$ and the quaternion of angular velocities $\omega_{\text{IMU}}[kT]$ at discrete time kT.

On the other hand, the tri-axis accelerometer can measure a reaction force that counteracts gravity in the opposite direction and thus indirectly measure gravity [71]. This means that, the gravity earth field being known, the vector of measured accelerations

$$a_{\text{IMU}} \equiv [0, a_x, a_y, a_z]^T \tag{4.14}$$

can be automatically referred to earth coordinate frame, here given in the NED coordinates. As conventionally taken, we assume that the direction of the gravity field is along the vertical z axis and is thus defined by the quaternion

$$g_{\text{NED}} \equiv [0, 0, 0, 1]^T \ . \tag{4.15}$$

Then, the orientation of the accelerometer is obtained by numerically solving the following optimization problem: Minimize a cost function $f(a_{\text{IMU}}, g_{\text{NED}}, q_{a\,\text{NED}}^{\text{IMU}})$

that identifies the distance between the vector of measured accelerations a_{IMU} and the direction of the gravity field g_{NED} rotated by the quaternion $q_{a}{}^{\mathrm{IMU}}_{\mathrm{NED}}$, the unknown parameter to be determined. The explicit analytical expression of the cost function f can be found in Ref. [68]. Here, it is worth observing that the possibility to resort to the solution of a minimum problem comes from the evidence that the direction of the gravity field is uniquely defined in the NED coordinate frame. Thus, once measured the accelerations a_{IMU}, one can determine the unknown quaternion $q_{a}{}^{\mathrm{IMU}}_{\mathrm{NED}}$. As proposed in [68], to carry out the minimization

$$\min_{q^{\mathrm{IMU}}_{\mathrm{NED}}} f(a_{\mathrm{IMU}},\, g_{\mathrm{NED}},\, q_{a}{}^{\mathrm{IMU}}_{\mathrm{NED}}) \tag{4.16}$$

in our drone we have implemented an iterative mechanism based on the gradient descent algorithm. Hence, the orientation from the accelerometer at the k-th discrete time instant turns out given by

$$q_{a}{}^{\mathrm{IMU}}_{\mathrm{NED}}[kT] = \hat{q}[(k-1)T] - \mu[kT]\frac{\nabla f}{\|\nabla f\|}[kT]\ , \tag{4.17}$$

where ∇f denotes the gradient of the geometrical surface defined by the cost function f, and μ is the step-size (in general, a time-dependent parameter) associated to the minimization procedure. The latter parameter determines the rate of convergence of the optimization. In this experimental work, the value of μ has been chosen constant and large in magnitude, so as to ensure that the convergence rate is equal or greater than the physical rate steering the change of the sensor orientation. Then, the resulting estimate of the orientation provided by the Madgwick filter is obtained by fusing the orientations $q^{\mathrm{IMU}}_{\mathrm{NED}}$ (for each discrete time instant kT) as given by Eqs. (4.13) and (4.17), respectively from the gyroscope and the accelerometer. The fusion is practically attained according to the following relation:

$$\hat{q}[kT] = \gamma\, q_{a}{}^{\mathrm{IMU}}_{\mathrm{NED}}[kT] + (1-\gamma)\, q_{g}{}^{\mathrm{IMU}}_{\mathrm{NED}}[kT]\ , \tag{4.18}$$

with $\gamma \in [0,1]$. Also the value of γ, depending on the value of μ, has been empirically chosen, so that Eq. (4.18), which realizes the fusion of the gyroscope and accelerometer data streams, is properly balanced, i.e., $q_{a}{}^{\mathrm{IMU}}_{\mathrm{NED}}$ and $q_{g}{}^{\mathrm{IMU}}_{\mathrm{NED}}$ have on average the same convergence rate. On the experimental side, this assumption leads to a quite small value of γ that privileges the stream data coming from the gyroscope with respect to the ones from the accelerometer.

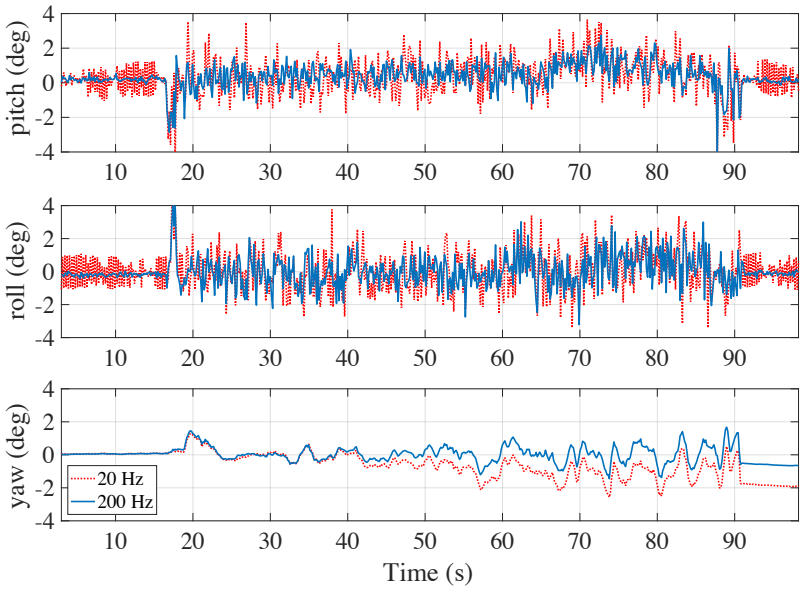

Figure 4.8: The attitude pitch, roll and yaw angles are here plotted during a flight which includes both a take-off and a hovering phase. The blue solid lines denote the output signals from the Madgwick fusion filter elaborated on board at the frequency rate of $200Hz$, whereas the red dotted lines are computed at $20Hz$.

In Figure 4.8 the plots of pitch, roll and yaw attitude angles, provided by the Madgwick filter implemented on DART-1.0 platform, are reported as functions of time. In particular, the blue solid lines refer to the orientation estimates provided by the Madgwick filter where the data from the IMU is updated with a frequency rate of $200Hz$. Instead, the red dotted lines are obtained implementing the same algorithm using a frequency rate of $20Hz$. In the figure one can observe that the red and blue lines have the same phase profile, though a greater amount of noise is present in the red curves. This difference can be immediately attributed to the different values of the frequency rate sampling the IMU data stream. Similarly, it is also worth noting that a higher sampling frequency rate leads to a less pronounced drift of the yaw orientation angle. This shows how in the multi-task architecture the attitude estimation guarantees higher performances compared to the

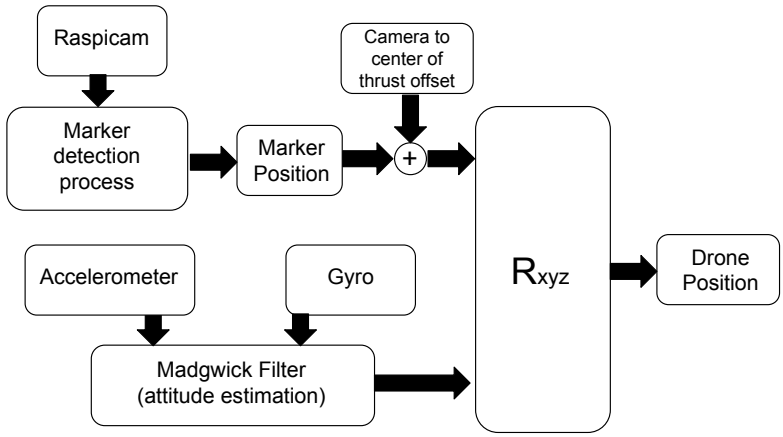

Figure 4.9: Schematic representation of the algorithm "Fixed Camera Frame Madgwick Filter" (FCF-MF). Unlike the algorithm FCF-CF, only the data stream from the accelerometer and the gyroscope are sent as input signals to the Madgwick sensor fusion filter. The latter provides an estimate of the drone orientation, whose precision does not depend on the distance between the drone and the marker.

single task implementation. In fact, in the multi-task implementation the attitude estimate runs at a frequency of $200Hz$, against 35 for the single task one.

4.5.3 FCF-MF position estimation method

In order to lessen the drawbacks of the first algorithm, the *Fixed Camera Frame Madgwick Filter* (FCF-MF) has been developed. In this new scenario, the Raspicam camera is still fixed with respect to the drone body, but the complementary filter has been replaced by the Madgwick filter, already mentioned in the previous paragraph. The schematic representation of this algorithm is shown in Figure 4.9. Since the Madgwick filter naturally exploits the intrinsic features of the inertial sensors, it is expected to be more robust with respect to the coupling effects on the pose, and, therefore, less noisy. It is worth also noting that, by adopting the Madgwick filter, it is no longer necessary to use the attitude vector coming from the vision. In this way, we are able to decouple the accuracy of the attitude estimate from the

marker distance, which, as previously explained, is the main limitation of the FCF-CF algorithm. The Madgwick filter takes as input raw data from the IMU (accelerometer and gyroscope) and estimates the attitude of the drone. Then, as in the previous method, the attitude estimate $\widehat{\Phi}$ is transformed into an estimate of the drone position by applying the rotation matrix R_{xyz}, which depends on all the three space axes.

4.5.4 SCF-MF-2DOF position estimation method

The third algorithm features the Raspicam camera mounted on a 2-DOF gimbal in order to stabilize the lens frame. It is important to note that the camera attitude estimate is made by the gimbal controller. Indeed being based on a microcontroller it is able to process the data of the IMU fixed with the camera at frequencies higher than *KHz*. This order of magnitude cannot be reached by non-real-time boards such as Raspberry. Therefore, the control of the gimbal is more efficient if delegated to its on-board electronics (separate from the rest of the drone). Thus, the position estimation method, named as *Stabilized Camera Frame Madgwick Filter 2DOF* (SCF-MF-2DOF), has been accordingly adapted. The block diagram of the SCF-MF-2DOF algorithm is shown in Figure 4.10, where it can be observed that the estimate made by the gimbal controller relative to the IMU fixed with the camera is not present. Indeed the electronics of the gimbal are closed and it is not possible to obtain any kind of information. Again, the algorithm uses the Madgwick sensor fusion filter to process the data stream from the accelerometer and gyroscope, but differently from the previous methods, here Q^c and Q_{cb}, denoting respectively the coordinates of the marker (in camera frame) and of the center of thrust with respect to the camera (constant offset vector between the frames of the lens and the center of thrust) can rotate independently. In particular, Q_{cb} is corrected by the complete rotation matrix R_{xyz} defined in Eq. (5.2) and applied to the estimate $\widehat{\Phi}$ of the attitude vector provided by the Madgwick filter. Instead, thanks to the use of the 2-DOF gimbal suspension, the coordinates of the marker need to be stabilized only by means of the rotation around the z-axis of the estimated attitude yaw angle $\widehat{\psi}$. More formally,

$$\widehat{Q}^f = R_{\text{Z}}(\widehat{\psi})Q^c + R_{\text{xyz}}(\widehat{\Phi})Q_{\text{cb}} \tag{4.19}$$

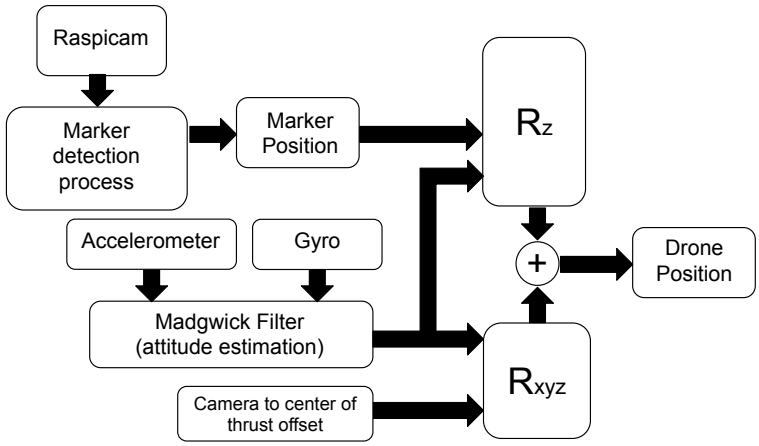

Figure 4.10: Schematic representation of the algorithm "Stabilized Camera Frame Madgwick Filter 2DOF" (SCF-MF-2DOF). Differently to previous methods, the 2-DOF gimbal suspension is used to stabilize the video stream data with respect to the pitch and roll angles of the attitude vector. As a result, the marker coordinates have to be corrected by means of a rotation just along the z-axis of the estimated attitude yaw angle $\widehat{\psi}$. This stabilization procedure has the advantage to reduce the noise in the video stream and thus improve the accuracy of position estimation [24].

where

$$R_z(\widehat{\psi}) = \begin{bmatrix} \cos\widehat{\psi} & \sin\widehat{\psi} & 0 \\ -\sin\widehat{\psi} & \cos\widehat{\psi} & 0 \\ 0 & 0 & 1 \end{bmatrix} . \tag{4.20}$$

It is worth noting that, since in this implementation the video stream is stabilized with respect to the pitch and roll angles by the 2-DOF gimbal, the effects caused by the noise sources affecting the camera are mitigated. The experiments reported in next section show that accuracy turns out improved almost by a factor of 5 with respect to previous methods.

4.5.5 SCF-MF-3DOF position estimation method

Finally, in the fourth estimation method, the 2-DOF gimbal is replaced by a 3-DOF gimbal. Hence, the marker coordinates are mechanically stabilized also with respect to the yaw angle ψ. The corresponding algorithm is here

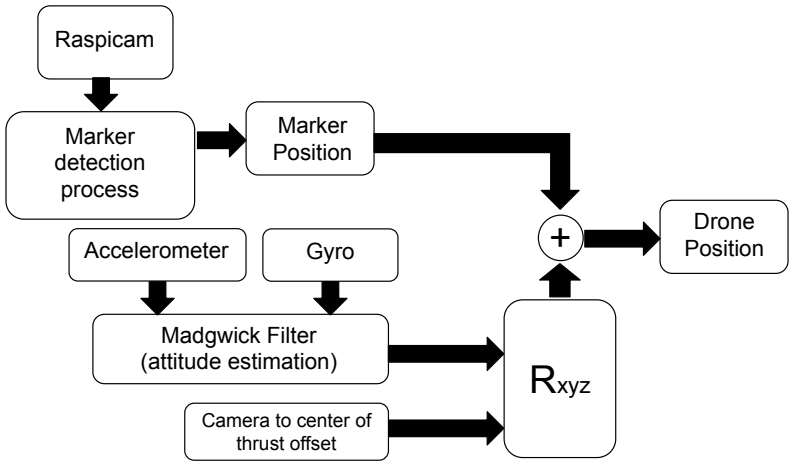

Figure 4.11: Schematic representation of the algorithm "Stabilized Camera Frame Madgwick filter 3DOF" (SCF-MF-3DOF). By improving the mechanical stabilization of the camera by means of a 3-DOF gimbal suspension, the marker coordinates Q^c are directly summed, without being corrected, to the term $R_{\mathrm{xyz}}(\widehat{\Phi})Q_{\mathrm{cb}}$ to obtain the estimation $\widehat{Q}$ of the drone position. In this way, the noise on the video stream data is further mitigated, and also less calculations are necessary to carry out the estimation procedure.

forth denoted as *Stabilized Camera Frame Madgwick filter 3DOF* (SCF-MF-3DOF). See Figure 4.11 for its schematic representation, whereby the block related to the correction of the marker position has been eliminated. Accordingly, to estimate the drone position, the marker coordinates do not need to be stabilized. This means that $\widehat{Q}$ is just provided by the following relation:

$$\widehat{Q}^f = Q^c + R_{\mathrm{xyz}}(\widehat{\Phi})Q_{\mathrm{cb}}. \tag{4.21}$$

Therefore, in addition to reducing the effects of noise on vision stream data, the 3-DOF gimbal has also the advantage of reducing the computational load, thus improving the precision in the estimate of both attitude and position vectors.

4.6 Experimental tests

The reference scenario used for the experimental tests presented in this section consists in tracking of a straight trajectory with triangular velocity profile. To carry out these tests, the drone takes off manually and is driven to a position where the Raspicam camera is able to identify the marker. The drone is then switched to an autonomous hovering mode (i.e., a flying mode where the reference trajectory is a point and the velocity profiles is constantly equal to zero), and finally the mission begins: The navigation software generates (on the three environmental axes x, y and z) a rectilinear trajectories with respect to the marker, and its tracking starts.

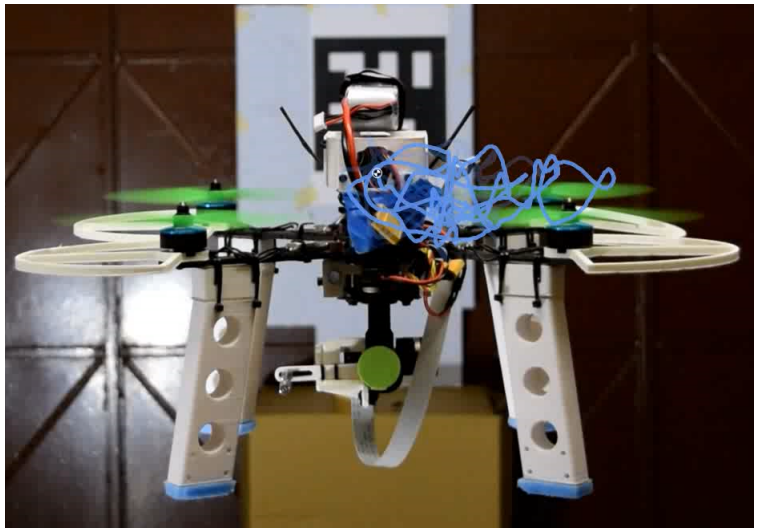

Figure 4.12: In the figure we show an image taken from the video made by a fixed camera that frames the drone during autonomous hovering. The image shows the tracking of the drone over time made by the "Kinovea" software.

4.6.1 Validation of the on-board computer vision system

Some preliminary tests were conceived to first verify the precision of the on-board computer vision system. To this aim, the drone was driven in front

of a marker and set to hovering at a distance of about $4m$. The navigation algorithm computed the drone camera position (here stabilized by the 3-axis gimbal) and the related data was logged with respect to the marker inertial frame. During the experiment, a specific tag on the back of the drone camera was recorded by a high precision external camera mounted on a tripod, and the corresponding video was processed off-line by the open-source software "Kinovea" [25, 26], which is able to infer the tag position with respect to the camera point of view. Such a signal, computed independently from the on-board system of the drone, was finally scaled to the marker reference system. A comparison between the off- and on-board measured trajectories along the y-axis (altitude) is reported in Figure 4.13. In the lower panel the difference between the two estimates is shown in yellow. In this experiment the standard deviation of this difference over the acquisition interval $[0, 70]$ seconds is about 4.2 mm, confirmed by other tests which witness similar values. In conclusion, the data from the on-board computer vision system turned out sufficiently informative and the very limited differences could be likely explained by the transient dynamics of the gimbal stabilization system.

4.6.2 Comparison between the position estimation methods

In this subsection, the performance reached with the proposed position estimation methods and their ability to be used in a reliable virtual position sensors for the drone navigation system are discussed.

In a first experiment, the drone has just been set to hovering in front of a marker at 4 m distance. Given the same control system described in Section IV-B, the experiment has been repeated alternatively using the algorithms FCF-CF, FCF-MF, and SCF-MF-2DOF, and eventually the drone ability to hover in the right position has been investigated. In Figure 4.14 the drone position p_y (altitude) as it has been estimated by the algorithms FCF-CF, FCF-MF and SCF-MF-2DOF is compared to the desired reference. As one can observe, the FCF-CF and FCF-MF algorithms achieve similar control performance, whereas the estimation yielded by the SCF-MF-2DOF method allows for an altitude profile much closer to the desired setpoint $p_y = 0$. To clearly quantify the performance of the algorithms, in the following table we provide the *corrected sample standard deviation* s_ε of the error ε computed as the difference between the estimated drone position and the

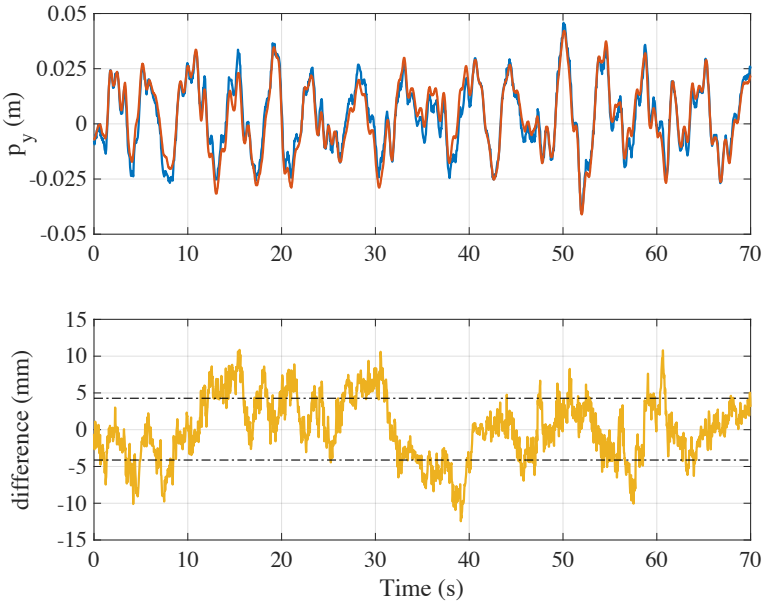

Figure 4.13: Comparison between the on-board and off-board estimations of the vision system data. In the upper panel, the estimation of the on-board altitude (blue solid line) is plotted together with the values (red solid line) measured by the fixed camera mounted on a tripod externally to the drone. Instead, the trend of the difference between the two estimates is reported in the lower panel. The mean deviation of such a difference is about 4.2 mm, while the maximum error value is around 1 cm.

desired trajectory within the time interval under investigation. The sample standard deviation is defined as

$$s_\varepsilon \equiv \sqrt{\frac{1}{N-1}\sum_{k=1}^{N}(\varepsilon - \bar{\varepsilon})^2} \tag{4.22}$$

where $N = 50$ is the number of performed experimental tests and $\bar{\varepsilon} \equiv \sum_{k=1}^{N} \varepsilon / N$.

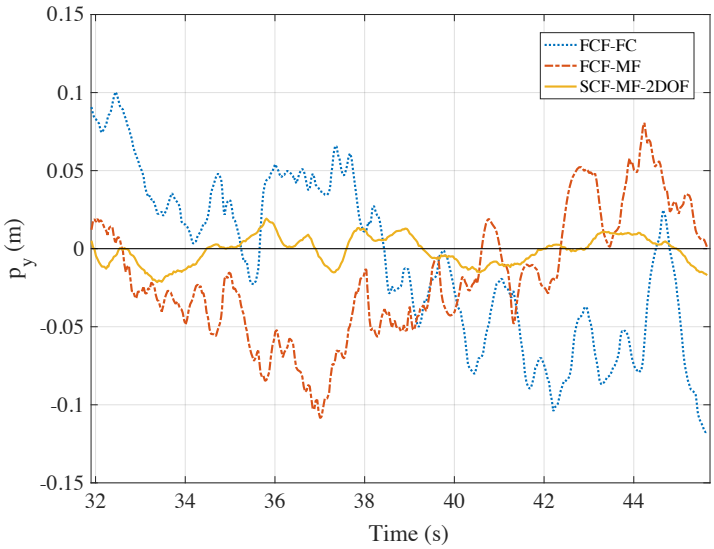

Figure 4.14: Comparison between the algorithms FCF-CF, FCF-MF and SCF-MF-2DOF for the estimation of the drone position along the altitude axis p_y. It can be observed that, if the SCF-MF-2DOF algorithm is used, the difference between the altitude estimates and the desired altitude profile ($p_y = 0$) is of an order of magnitude smaller than the other two analyzed methods.

Algorithm	Standard deviation (along y)
FCF-CF	5.35 cm
FCF-MF	4.01 cm
SCF-2DOF	1.77 cm

As shown in the table, the performance of the algorithm SCF-MF-2DOF are much better than the ones of the first two proposed estimation methods and, quantitatively, the sample standard deviations s_ε of the error are halved.

In Figure 4.15 it is reported, for a similar experiment, the comparison between the estimation algorithms SCF-MF-2DOF and SCF-MF-3DOF taking this time into account the horizontal position p_x. Indeed, in this latter case the camera yaw angle is now stabilized by the use of the 3-DOF gimbal suspension and is always pointing at the same direction. In terms of tracking precision, the performance of the SCF-MF-3DOF algorithm has to be mainly

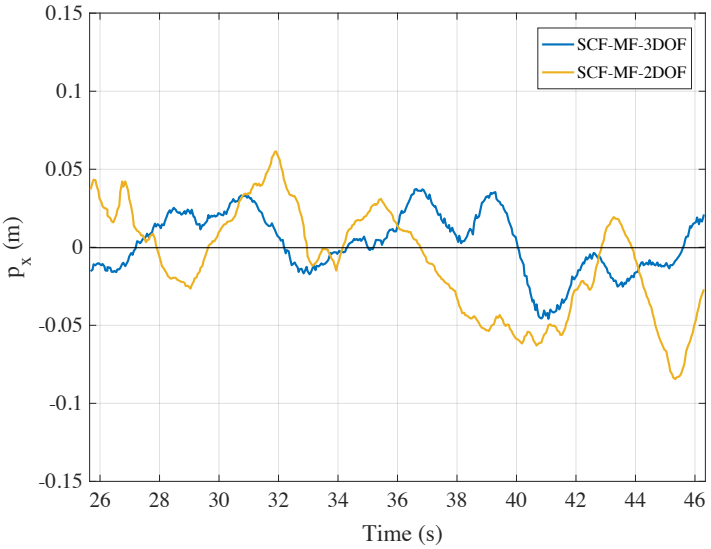

Figure 4.15: Comparison between the algorithms SCF-MF-2DOF (with 2-DOF gimbal) and SCF-MF-3DOF that uses a 3-DOF gimbal suspension for the stabilization of the yaw angle. In this case, the comparison is made between the estimates obtained along the horizontal axis x.

evaluated along the horizontal axis x, being unchanged for the other axes. The sample standard deviation s_ε of the estimation errors for the algorithms SCF-MF-2DOF and SCF-MF-3DOF have been computed again by repeating $N = 30$ times the same experiments with fixed working conditions. The values of the error standard deviations are provided in the following table:

Algorithm	Standard deviation (along x)
SCF-2DOF	3.42 cm
SCF-3DOF	2.54 cm

From the table, the error along the horizontal axis turns out larger than the one along the vertical axis. Nevertheless, the standard deviation of the error is about 2.5 cm, thus proving overall a few centimeter precision in controlling the position of the Dart drone prototype when exploiting the SCF-MF-3DOF on-board navigation system.

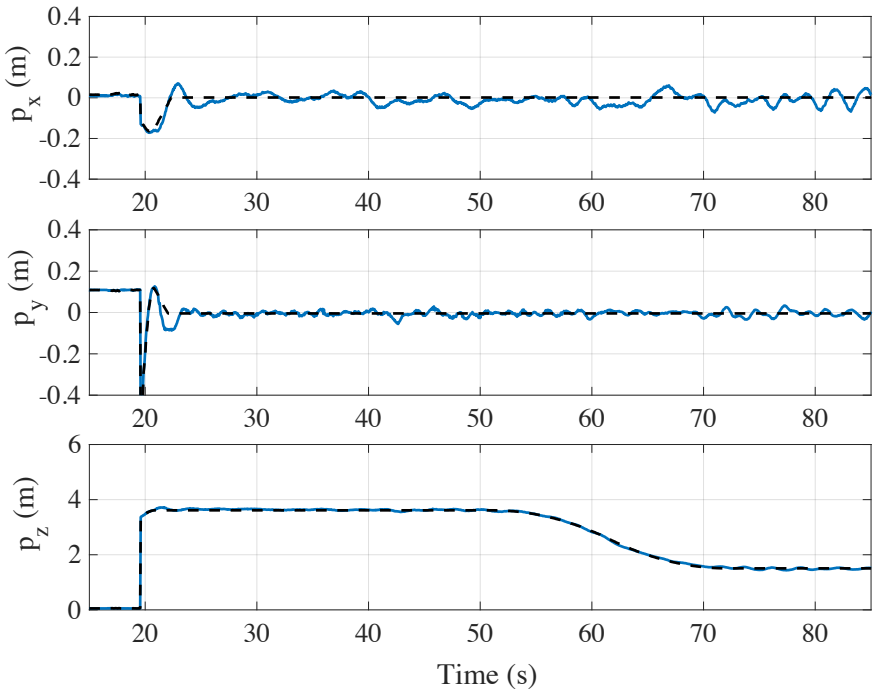

Figure 4.16: Drone position vs time: the blue solid curves represent the 3D-position of the drone in an experimental test with a specific straight trajectory having a triangular velocity profile (black dashed lines) starting at $t = 50$ s. The estimated positions of the drone are obtained on-board applying the implemented SCF-MF-3DOF algorithm to the data from the IMU and the camera during the flight.

4.6.3 Autonomous flight test

In Figure 4.16, we show the time-behaviour of the drone position (blue solid lines for the elements p_x, p_y, p_z) while the drone is in the autonomous flight mode, during the tracking of a preset rectilinear trajectory. The measured position of the drone is also compared with the desired trajectory (black dashed lines) that has to be tracked. In both the three panels in Figure 4.16, one can observe a transient regime (approximately in the time interval $[0, 25]$ seconds in which the drone is aligned to the marker. During the time evolution in the interval $[25, 50]$ seconds, the autonomous hovering mode is

enabled, which results in having an almost constant value of p_y (the distance between the drone and the ground). Instead, from $t = 50$ seconds until the end of the test, the autonomous flight mode was enabled, so as to allow for the tracking of a straight trajectory with a triangular velocity profile. In the test, the maximum value of the velocity is 0.1 m/s. This implies a variation of the desired trajectory (blue curve) along z. The mission ends when the drone reaches a distance of 1.5 meters away from the marker. At that distance, the drone automatically returns to the autonomous hovering mode and stops moving.

Both the hovering and autonomous modes mainly use the data stream from the computer vision system and from the IMU. Although signals from the sensors are filtered from external noise sources and fused together, we are able to achieve a correct tracking of the trajectory along the three axes but with a self-sustained oscillation perceptible on p_x and p_y. Such oscillations are originated by a delay of around 0.2 seconds in the video acquisition process (due to data buffer) and affects the performance of the navigation control system. This aspect, that has been already partially discussed in Ref. [55].

4.7 The latency problem in computer vision algorithms

As seen in the previous paragraphs, navigation techniques based on computer vision can guarantee high precision and the possibility of repeating the same trajectories with a deviation of a few centimeters. However, one of the biggest problems facing odometric systems based on computer vision algorithms is latency. Through experimental tests it has been determined that the computer vision algorithm implemented on board the Raspberry has a delay of about $200ms$. Figure 4.17 shows the acceleration signal coming from the IMU and the altitude estimated by the computer vision. As it can easily be observed, there is a delay in the signal processed by the computer vision algorithm.

To achieve high performance this delay is not negligible, therefore a second order complementary filter has been designed to solve this problem [27]. The logic of the complementary filter is simple and foresees the use of a low pass and a high pass filter, suitably designed so that the sum has unity gain. To recover the delay affecting the signal produced by the computer vision,

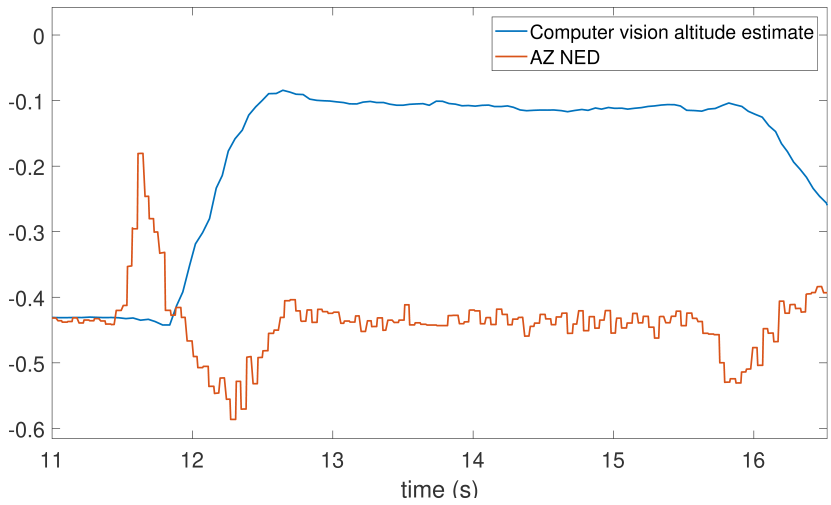

Figure 4.17: The figure shows the acceleration trend on the altitude axis, which comes from the IMU (red line). Instead, the blue line shows the trend of the altitude estimate produced by the vision algorithm. As can be seen, the position estimation coming from the computer vision algorithm is delayed with respect to the signal coming from the inertial sensor.

the idea is to reconstruct this signal with the acceleration coming from the IMU. In fact, the latter has negligible latency and has information content at high frequencies (unlike the signal coming from vision which has high precision at low frequencies). Figure 4.18 shows the block diagram of this filter.

$H_p(s)$ and $L_p(s)$ are the two transfer functions that implement the high pass and low pass filters respectively. Equations 4.23 and 4.24 define the two transfer functions in the Laplace domain.

$$H_p(s) = \frac{s^2}{s^2 + as + b} \tag{4.23}$$

$$L_p(s) = \frac{as + b}{s^2 + as + b} \tag{4.24}$$

The order of the filter is imposed by the type of signals to be merged, in fact the acceleration must be integrated twice before it can be used. In the

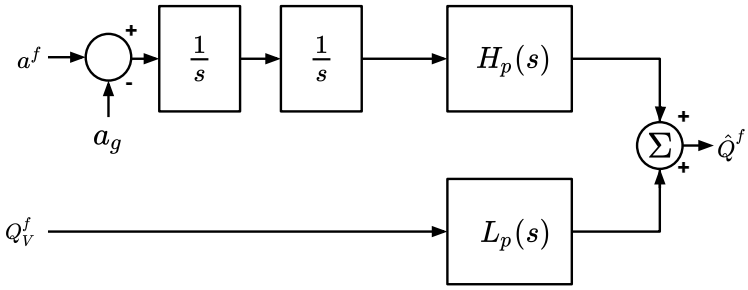

Figure 4.18: The figure shows the block diagram of the second order complementary filter designed to recover the delay of the estimate produced by the computer vision algorithm.

block diagram of Figure 4.18, a^f is the acceleration vector in fixed frame, obtainable from the acceleration vector in body frame by equation 4.25.

$$a^f = q \otimes a^b \otimes q^* \tag{4.25}$$

Where q is the quaternion of attitude estimated through the Madgwick filter, q^* represents its conjugate and a^b is the acceleration vector in body frame (sensor frame). The product between quaternions is denoted by $\otimes$, which can be done through the Hamilton rule. Instead, Q_V^f and $\hat{Q}^f$ respectively represent the position vector coming from the computer vision algorithm and the estimate produced by the second order complementary filter. Note that the gravity component must be subtracted from the vector a^f (in order to avoid an uncontrolled drift of the altitude estimate). In the block diagram of Figure 4.18, the gravity vector is called a_g and is defined by equation 4.26.

$$a_g = [0, 0, 1]^T \tag{4.26}$$

The filter was initially implemented on *Simulink*, inserting the vertical acceleration and altitude estimation signals logged during the experimental tests at its inputs.These signals were acquired by means of fast high-low

movements, this was done to be able to clearly see the acceleration peaks, in order to understand if the filter is able to eliminate the delay. As can

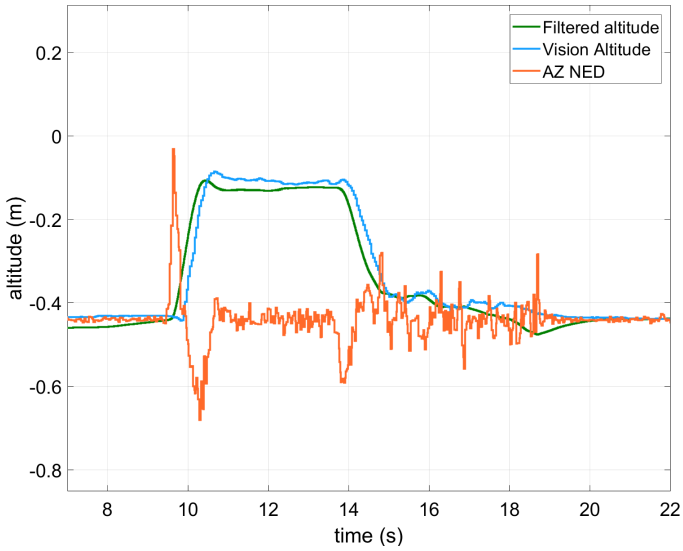

Figure 4.19: The figure shows the behavior of the filter (green line) during the simulation made with Simulink. The orange and blue lines respectively show the acceleration trend and the position estimated by the computer vision, which were acquired during experimental tests.

be seen from figure 2.19, the reconstructed signal (green line) has a trend very similar to that produced by the vision process, but it appears to be anticipated by a time value approximately equivalent to the vision delay. From the tests carried out using Simulink it has been seen that the value of the cutoff frequency that minimizes the delay and the distortion of the signal is equal to 0.1 Hz. From the value of the cut-off frequency it was possible to determine the coefficients of the continuous time transfer functions used for the simulation. These values are shown in the following table.

Coefficient	Value
a	0.8796
b	0.3948

4.7.1 Code implementation and testing

To carry out the software implementation of the filter explained above, first of all it is necessary to establish in which environment it will run. In this regard, it was decided to implement it within the IMU thread of the multi-task architecture. This information is indispensable for converting the transfer functions from the frequency to the time domain. Through Tustin's method, by setting a sampling frequency of 200 Hz (which is the sampling frequency of the IMU thread), the two time-discrete transfer functions 4.27, 4.28 were obtained.

$$L_p(z) = \frac{Y_0(z)}{X_1(z)} = \frac{a_0 z^2 + b_0 z + c_0}{a_1 z^2 + b_1 z + c_1} \tag{4.27}$$

$$H_p(z) = \frac{Y_2(z)}{X_3(z)} = \frac{a_2 z^2 + b_2 z + c_2}{a_3 z^2 + b_3 z + c_3} \tag{4.28}$$

Solving the equations with respect to the output, we obtain the relations to be implemented in code, which are shown in 4.29 and 4.30.

$$y_t^0 = -b_1 y_{t-1}^0 - c_1 y_{t-2}^0 + a_0 x_t^1 + b_0 x_{t-1}^1 + c_0 x_{t-2}^1 \tag{4.29}$$

$$y_t^2 = -b_3 y_{t-1}^2 - c_3 y_{t-2}^2 + a_2 x_t^3 + b_2 x_{t-1}^3 + c_2 x_{t-2}^3 \tag{4.30}$$

The numerical values of the coefficients calculated in order to respect the constraints of the cutoff frequency and the sampling frequency of the thread are shown in the following table.

Coefficient	Value
a_0	0.002197
b_0	$4.924 * 10^{-6}$
c_0	-0.002192
a_1	1
b_1	-1.996
c_1	0.9956
a_2	$6.236 * 10^{-6}$
b_2	$1.247 * 10^{-5}$
c_2	$6.236 * 10^{-6}$
a_3	1
b_3	-1.996
c_3	0.9956

Obviously, given the low cut-off frequency used, in order to have a behavior faithful to that obtained on Simulink and not to make the filter diverge, it is necessary to take the greatest possible number of decimal figures. For this purpose, these coefficients have been saved in the software in *DOUBLE* variables (double precision). In addition, the actual implementation of the filter must take into account very accurately the value of a_g. In fact, this vector not only depends on the calibration (which can be done only one time), but it can also vary over time due to physical changes of the same.

Since the performance of the filter, due to the double integrator and the cut-off frequency (which gives a lot of weight to the inertial quantities) are greatly affected by the value assumed by a_g, in the software implementation it was decided to write an initialization routine, which provides an auto-calibration of the sensor, based on the real-time acquisition of a sample of 1400 values. Once this additional piece of code was written (made available in the software in the form of a library), it was decided to carry out experimental autonomous flight tests. To evaluate the effective performance of the filter during autonomous flight, it was decided to reuse the same method shown in subsection 4.6.1. That is, the fixed camera and the Kinovea software were used to obtain an off-board reference of the position of the drone.

As can be seen from figure 4.20, the reference position provided by the Kinovea software is very close (in some moments in time it perfectly overlaps) to the estimate provided by the computer vision algorithm. However, from this experiment we see that the estimate given by the complementary filter

eliminates the delay problem but introduces the distortion problem. In fact, the trend of the filtered signal shown in figure 4.20 denotes a greater error than the starting signal. The presence of this distortion is mainly due to the cut-off frequency of $0.1Hz$, which is necessary to eliminate the delay. Such a low frequency in fact gives too much weight to the information coming from the accelerometer. Which, during flight is subjected to many vibrations, just think of the moving propellers on the multirotor. Although this method works, it does not present improvements that justify its use.

This experiment showed the hardware limitations of the technology, since, in order to reduce the latency problem produced by computer vision while maintaining the same level of precision, sensor fusion algorithms are not enough and it is necessary to change the type of technology. For this reason the DART-2.0 and DART-2.1 platforms (discussed in the next chapter) make use of the Intel Realsense stereoscopic camera. Which exploits a completely different concept, implementing an FPGA card on board. In this way it is able to process both the SLAM algorithm (which provides the odometric data) and the video stream on board, reducing the latency of the output data by two orders of magnitude.

4.8 Conclusions

This chapter presented the software architectures and algorithms developed for the DART-1.0, DART-1.1 and DART-1.2 hardware platforms. These prototype versions feature only mass market inexpensive components, which have been suitably optimized to achieve high accuracy in position and attitude estimation. The navigation system is based on "virtual sensing" obtained by fusing the data from a IMU and an on-board computer vision system. The resulting information is exploited by a simple control logic which makes the navigation system act as a "cyber-pilot" that overrides the human commands when it is in autonomous mode.

Experiments suggest that low-cost technologies, such as those used to implement these UAVs, are very close to enable the sought passage from meter- to centimeter-scale precision in autonomous maneuvering of multirotor drones that would represent a noteworthy generation change in their application range. Moreover, both hardware and software architectures are modular and they can easily be extended and enhanced, for instance, by replacing more refined algorithms into the programs, or by substituting a

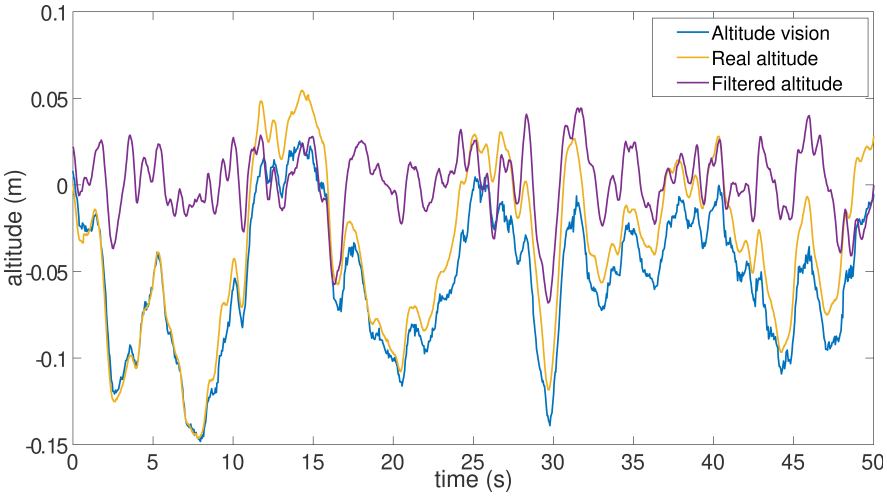

Figure 4.20: The figure shows the trend of the height estimate provided by the computer vision module (blue line) and the estimate provided by the filter (purple line). These two signals were compared with the vertical position provided by the Kinovea software (purple line). The results show that the filter is significantly affected by the noise coming from the IMU during the flight phase.

device with a better performing one. Such a feature is crucial for the maintenance of the project. In fact, the next chapter will show the software architectures and algorithms based on the use to novel devices, which have recently win over the mass market.

Chapter 5

Complex autonomous missions with unknown path

This chapter shows the algorithms and software architectures developed for the DART-2.0 and DART-2.1 platforms. Since the technology implemented on board is significantly different from the platforms discussed in the previous chapter, the threads and functions of the software are also different. The goal of this chapter is to carry out complex missions in autonomous flight mode. These missions are based on the detection of n *gates present in the environment and on the automatic planning of the route in real-time. Subsequently, the problem of managing a map containing the endpoints calculated during the execution of the mission and the need for a mission supervisor able to make choices will be discussed. Finally, the problem of simulation will be addressed, in particular a hardware in the loop technique will be presented to test the software algorithms in a synthetic environment, facilitating the development of new software for autonomous navigation.*

5.1 Introduction

With the use of the Intel Realsense camera, it is no longer necessary to base the odometry system on the vision of one or more markers. The camera itself acts as an odometric system, through on-board SLAM techniques [75]. The performances achieved by these cameras are currently not achievable

by any SLAM technique based on parallel programming [78] (programming techniques based on the use of hardware acceleration provided by the GPU). In fact, the biggest bottleneck currently present in the most advanced SLAM techniques is given by latency. Since, in traditional architectures, the images must first be preprocessed by the camera, then sent to the CPU and only subsequently copied to the GPU memory, which will have to perform any calculations. Intel has engineered the SLAM techniques by integrating them on FPGA boards [76, 77], so all these pre-processing and data copying steps are eliminated, this has led to a reduction in latencies to about 5-6 ms. Such a low latency value makes it the ideal tool on which to base autonomous navigation techniques and on which to close high-level control loops.

5.2 Desired autonomous mission

Specifically, the mission that the drone must carry out in autonomous flight mode involves the detection of n gates present in the environment with unknown positions and the determination of the trajectory necessary to cross them [89]. As we will see in paragraph 5.6 in order to carry out a mission of this type, it is necessary to create and manage a dynamic map, containing the position of the gates detected in the environment. Furthermore, even the orientation of the gates is unknown, for this purpose the control system proposed in chapter 4 must be extended, as it must also be able to operate with non-zero heading angles.

Each gate has an ID determined by a reference marker [66, 67], so a computer vision system similar to the one shown in chapter 4 is used. The new implementation of the computer vision algorithm (which has the task of detecting the gates) also has considerable latency, however in this configuration this latency is irrelevant. Because the control loop is closed through the state vector provided by the Intel stereoscopic camera, which has negligible latency. In this scenario the only variables known by the drone are represented by the desired sequence of gates to cross. Which is identified by a vector containing a sequence of IDs, each of which corresponds to a specific gate.

5.3 Computer vision algorithm

The new implementation of the computer vision algorithm which has the task of recognizing the markers, is no longer based on the VISP libraries [82], in fact it has been rewritten using the *OpenCv* libraries. In particular, the Aruco module was used, which allows greater flexibility in the parameter tuning phase. The OpenCv libraries currently represent the state of the art for writing computer vision algorithms, in addition some functions support CUDA acceleration, which on Nvidia Jetson board is an advantage.

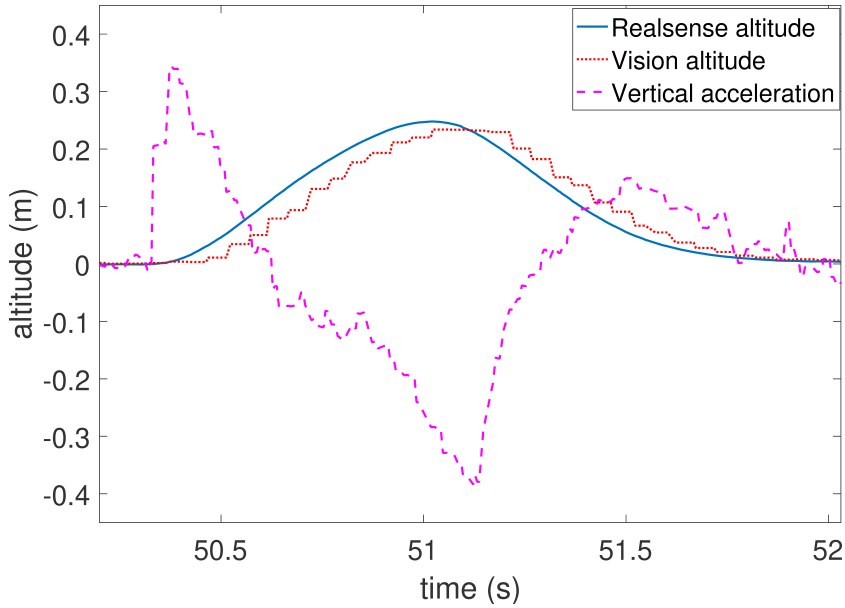

Figure 5.1: The experiment in the figure shows the latency between the altitude estimate provided by the Intel camera (blue line) and the OpenCv based computer vision algorithm (red dotted line). The trend of the acceleration signal (purple dotted line) from the IMU is also shown as a reference.

As mentioned in paragraph 5.2, the latency of this algorithm in the present architecture is not important, since the control loop is closed on the state provided by the Realsense camera. However, it is interesting to observe the latencies involved, for this purpose in figure 5.1 the trend of

the altitude estimated by the Realsense camera and the vision algorithm is shown, as a reference the acceleration trend is reported. The graph shows the delay generated by the computer vision algorithm, which (as expected) is not present in the estimate provided by the Realsense camera.

However, from the tests made it emerged that in the OpenCv implementation the delay was reduced to a value of $100ms$, against $200ms$ of the old algorithm. This improvement, is not completely given by the OpenCv libraries, as it may also depend on the higher performance of the Nvidia Jetson card. The video stream of the experiment in figure 5.1 was acquired by one of the two fisheye lenses of the Realsense camera. The same experiment was then repeated using the video stream coming from a Raspicam camera, obtaining an average latency of $150ms$. This showed that standard CSI cameras have an inherent latency of at least $50ms$.

5.4 Extended control system

The high control module shown in chapter 4 limits the set of possible drone maneuvers to those with zero orientation angle. This is due to the association made between the environment coordinates and the attitude angles of the drone. However, there are missions where it is convenient to vary the orientation of the drone, for this purpose the control module has been extended.

To generalize the high-level control module, it is necessary to avoid associating the PID's that controls the x-axis and the z-axis respectively to the generation of a roll angle and pitch angle setpoints. To do this it is necessary to consider the angle of the body heading ψ_b^f to mixing the input error of the PID's C_x and C_z. For this purpose it is possible to use a rotation matrix to rotate the error vector around the y-axis by a ψ_b^f angle. That is, doing a reverse rotation of the ψ_b^f angle, necessary to consider the orientation of the drone with respect to the fixed frame. The error vector of the control module shown in chapter 4 can be considered referred to the fixed frame, as it does not consider the orientation of the drone.

$$\vec{e}^{\flat} = R_y(\psi_b^f)^{-1}\vec{e}^f \tag{5.1}$$

In Eq. 5.1, the $R_y(\psi_b^f)^{-1}$ defines the reverse rotation matrix that rotates the angle of ψ_b^f around the y-axis (altitude axis), $\vec{e}^f$ is the error vector referred to the fixed frame and the $\vec{e}^{\flat}$ is the error vector that takes into

account the orientation of the drone. The reverse rotation matrix is defined in Eq. 5.2

$$R_y(\psi_b^f)^{-1} = \begin{bmatrix} c(\psi_b^f) & 0 & s(\psi_b^f) \\ 0 & 1 & 0 \\ -s(\psi_b^f) & 0 & c(\psi_b^f) \end{bmatrix} \tag{5.2}$$

Hence, from Eq. 5.1 and Eq. 5.2 it is possible to obtain the mixed error vector which considers how the drone is oriented with respect to the fixed frame.

$$\vec{e}^b = \begin{bmatrix} e_x^f \cos(\psi_b^f) + e_z^f \sin(\psi_b^f) \\ e_y^f \\ e_z^f \cos(\psi_b^f) - e_x^f \sin(\psi_b^f) \end{bmatrix} \tag{5.3}$$

Then it is possible to obtain the generalized control module shown in Fig. 5.3.

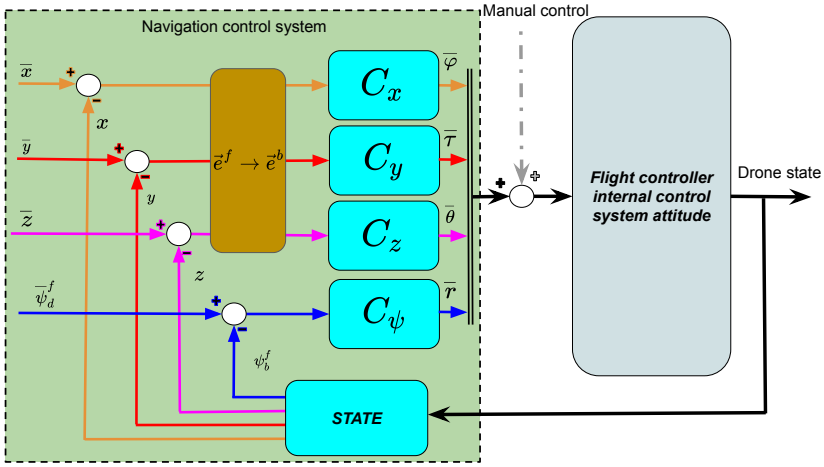

Figure 5.2: The figure shows the extension of the control module introduced in chapter 4. In the new architecture the orientation of the drone with respect to the fixed frame is considered by mixing the error vector.

Where the notations $\vec{e}^f \rightarrow \vec{e}^b$ is the Eq. 5.3. The new control architecture is able to follow endpoints with generic orientations. With the new control architecture, any desired ψ_b^f can be used as input.

5.4.1 Software implementation

The implementation of the code part that manages the control routine has been made more modular than the previous one (the old implementation is present on the DART-1.0, DART-1.1 and DART-1.2 platforms). As can be seen from Figure 5.3, there are 3 nested classes within the control thread: the *controlRoutine()* class calls the *pid3D()* class, which in turn calls the *pid()* class.

The higher level *controlRoutine()* class is able to manage the initialization phases of the control algorithm, avoiding the wind-up effect of the PIDs. In fact, PIDs can suffer from this problem if they are not managed and initialized correctly. The class *controlRoutine()* calls the object *pid3D()* by passing it the error vector in input and receiving the command signals in output. The *pid3D()* class has the task of iteratively creating the "simple" pid objects, which allow to generate the complete control algorithm.

This architecture allows to abstract more and more the control algorithm, starting from the low level, up to an increasingly higher level in which it is sufficient to pass the state of the drone to receive the "command" vector. Besides making high-level code easier, this approach allows to easily replace low-level modules in case of wanting to implement control methods other than PIDs.

In fact, supposing to replace the PID-based control algorithm with another one, it is sufficient to replace the software abstraction levels 2 and 3, leaving the code of the levels 0 and 1 unchanged. Therefore, the current software implementation guarantees a high degree flexibility, useful for the development of the platform.

5.5 Crossing gate method with generic orientation

To crossing of a gate by rectilinear paths, one possible approach is to identify two points defined as endpoints. The first endpoint is fixed before the gate and the second after, at a distance that can be defined a-priori. We define

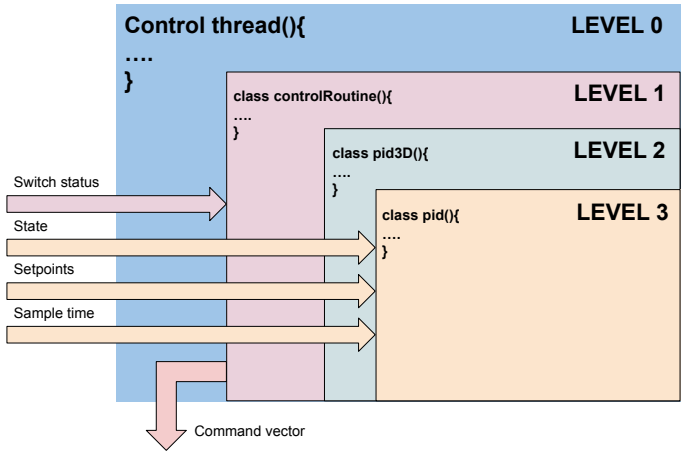

Figure 5.3: The figure shows the 3 levels of code abstraction within the control thread. The input signals to the thread are divided into 4 categories: (i) state vector; (ii) vector of set-points; (iii) status of the remote control switches (which define the flight modes) (iv) sampling time. It is important to note that the state of the switches is managed at a high level through the *controlRoutine()* class.

the heading of the drone in fixed frame as ψ_b^f, the gate heading in camera frame as ψ_g^c and the desired heading that the drone must have when passing through the gate as ψ_d^f, defined in fixed frame. The ψ_d^f is defined by the following Eq. (5.4)

$$\psi_d^f = \psi_b^f + \psi_g^c \tag{5.4}$$

As an implementation note it is important to observe that the sum must be made only between pairs of data obtained at the same iteration, otherwise the relationship is invalid. Indeed, the ψ_g^c defines only the orientation that exists between the camera and the gate. To obtain the ψ_d^f it is necessary to add to the ψ_g^c the ψ_b^f that defines the orientation of the drone with respect to the fixed frame. Although it may seem obvious there is to consider that the ψ_b^f is always available (because it comes from an inertial estimate) while the ψ_g^c is not (since it depends on the vision process). In fact, as it will be explained in section 5.6, the ψ_g^c must be present in the mission map, where all the endpoints generated frame by frame by the vision algorithm are saved

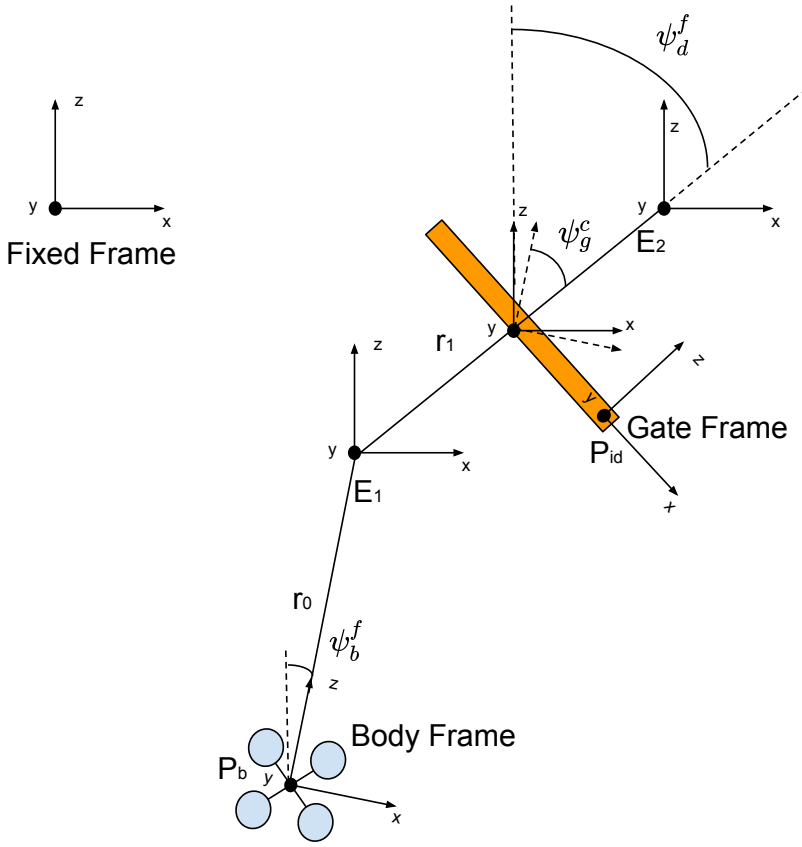

Figure 5.4: Generation of straight paths and endpoints to crossing gate with generic orientation. Where r_0 and r_1 are respectively the gate approach line and the crossing line.

and updated. P_{id}, E_1 and E_2 represents respectively the position of the gate marker, the first endpoint and the second endpoint. The two endpoints are calculated starting from the gate position defined by the AprilTag marker (introduced in chapter 4). In this regard we consider two constant offset vectors $\vec{O}_1^f$ and $\vec{O}_2^f$ defined in fixed frame. These vectors are shown in Eq. (5.5) and in Eq. (5.6).

$$\vec{O}_1^f = \begin{bmatrix} x_1^f \\ y_1^f \\ z_1^f \end{bmatrix} \tag{5.5}$$

$$\vec{O}_2^f = \begin{bmatrix} x_2^f \\ y_2^f \\ z_2^f \end{bmatrix} \tag{5.6}$$

The $\vec{O}_1^f$ and $\vec{O}_2^f$ are used to calculated the coordinates of the two endpoints. Since it is assumed that the gates are all in vertical position but with unknown orientation, to obtain the offsets in the camera frame it is sufficient to rotate around the altitude axis. The rotation must be equal to the angle ψ_g^c , which is the orientation angle between the camera and the gate frame. Since rotation must transform $\vec{O}_1^f$ and $\vec{O}_2^f$ vectors from fixed frame to camera frame, it must be a reverse rotation. The reverse rotation matrix shown in Eq. (5.7) is used for this purpose.

$$R_y(\psi_g^c)^{-1} = \begin{bmatrix} c(\psi_g^c) & 0 & s(\psi_g^c) \\ 0 & 1 & 0 \\ -s(\psi_g^c) & 0 & c(\psi_g^c) \end{bmatrix} \tag{5.7}$$

Multiplying the reverse rotation matrix with the two offset vectors defined in Eq. (5.5) and (5.6), it is possible to obtain the coordinates of the two constant offset vectors defined in camera frame.

$$\vec{O}_1^c = \vec{O}_1^f R_{\mathrm{y}}(\psi_g^c)^{-1} \tag{5.8}$$

$$\vec{O}_2^c = \vec{O}_2^f R_{\mathrm{y}}(\psi_g^c)^{-1} \tag{5.9}$$

At this point it is possible to calculate the coordinates of the two endpoints by adding $\vec{O}_1^c$ and $\vec{O}_2^c$ to $\vec{P}_{id}^c$, which defines the position of the gate in camera frame.

$$\begin{cases} \vec{E}_1^c = \vec{P}_{id}^c + \vec{O}_1^c \\ \vec{E}_2^c = \vec{P}_{id}^c + \vec{O}_2^c \end{cases} \tag{5.10}$$

In the Eq. (5.10) the $\vec{E}_1^c$ and $\vec{E}_2^c$ are the endpoints defined in camera frame. The feedback of the control system receives the status of the drone in fixed frame coordinates, so the setpoints must also be defined in the same reference system. In the Eq. (5.11) the relation that allows to define $\vec{E}_1$ and $\vec{E}_2$ in the gate frame is shown. Where $\vec{Q}_{cb}$ is the constant offset vector between the camera frame and the body frame, while $R_{xyz}(\psi_b^f, \varphi_b^f, \theta_b^f)$ is the xyz rotation matrix that considers the attitude of the body defined in the Eq. (5.12).

$$\begin{cases} \vec{E}_1^g = (\vec{E}_1^c + \vec{Q}_{cb}) R_{xyz}(\psi_b^f, \varphi_b^f, \theta_b^f) \\ \vec{E}_2^g = (\vec{E}_2^c + \vec{Q}_{cb}) R_{xyz}(\psi_b^f, \varphi_b^f, \theta_b^f) \end{cases} \tag{5.11}$$

$$R_{\mathrm{xyz}}(\psi_b^f, \varphi_b^f, \theta_b^f) = \begin{bmatrix} c_\varphi c_\psi & c_\varphi s_\psi s_\theta + s_\varphi c_\theta & -c_\varphi s_\psi c_\theta + s_\varphi s_\theta \\ -s_\varphi c_\psi & -s_\varphi s_\psi s_\theta + c_\varphi c_\theta & s_\varphi s_\psi c_\theta + c_\varphi s_\theta \\ s_\psi & -c_\psi s_\theta & c_\psi c_\theta \end{bmatrix} \tag{5.12}$$

Finally, to obtain the coordinates of $\vec{E}_1$ and $\vec{E}_2$ in a fixed frame, it is necessary to perform the translation shown in the Eq. (5.13). Where $\vec{P}_b^f$ is the position of the body defined in fixed frame.

$$\begin{cases} \vec{E}_1^f = \vec{P}_b^f - \vec{E}_1^g \\ \vec{E}_2^f = \vec{P}_b^f - \vec{E}_2^g \end{cases} \tag{5.13}$$

In Eq. (5.14) shows in compact form the complete equation that allows to obtain the endpoints to be used for the generation of the setpoints necessary for the control system.

$$\begin{cases} \vec{E}_1^f = \vec{P}_b^f - \{[(\vec{P}_{id}^c + \vec{O}_1^f R_{\mathrm{y}}(\psi_g^c)^{-1}) + \vec{Q}_{cb}] R_{xyz}(\psi_b^f, \varphi_b^f, \theta_b^f)\} \\ \vec{E}_2^f = \vec{P}_b^f - \{[(\vec{P}_{id}^c + \vec{O}_2^f R_{\mathrm{y}}(\psi_g^c)^{-1}) + \vec{Q}_{cb}] R_{xyz}(\psi_b^f, \varphi_b^f, \theta_b^f)\} \end{cases} \tag{5.14}$$

5.6 Mission map management

The mission map is managed by a new thread called *planner()*. To explain the management of the mission map, it is necessary to explain how the input data is generated. The vision thread takes care of detecting frame

by frame all gates present in the images and updating two dynamic array of data structures at each cycle. This dynamic array of data structures are called *gateContainer*, one defines the position and the attitude of each gate in camera frame and the other in fixed frame. Both arrays are dynamic, their size varies with each cycle and depends from the number of the gates detected within the current frame.

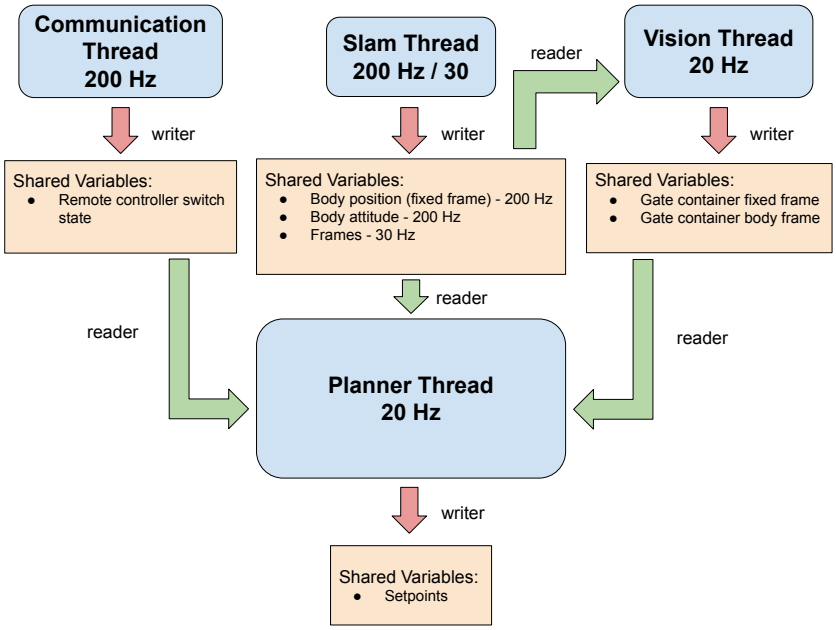

Figure 5.5: The figure shows how the Planner thread is interconnected in the software environment. In particular, it shows which threads it needs to communicate with (according to the synchronization method explained in chapter 4).

the transformation of the coordinates from the camera frame to the fixed frame is carried out as shown in the previous paragraph. The planner thread receive in input the two dynamic data structures defined previously and create a map. In particular the function that take care of create the map is called *updateMap()*. When it is called, it compares the elements of the map with the elements of *gateContainer*, which are relative only to the current

frame (so they do not consider the past). The elements not present are inserted, instead the elements present are compared with the updated elements. If the gap between the two elements exceeds the threshold value, they are replaced, otherwise the old elements are kept in the map. This is done to avoid recalculating the trajectory at each cycle, generating a noise effect on the desired trajectory.

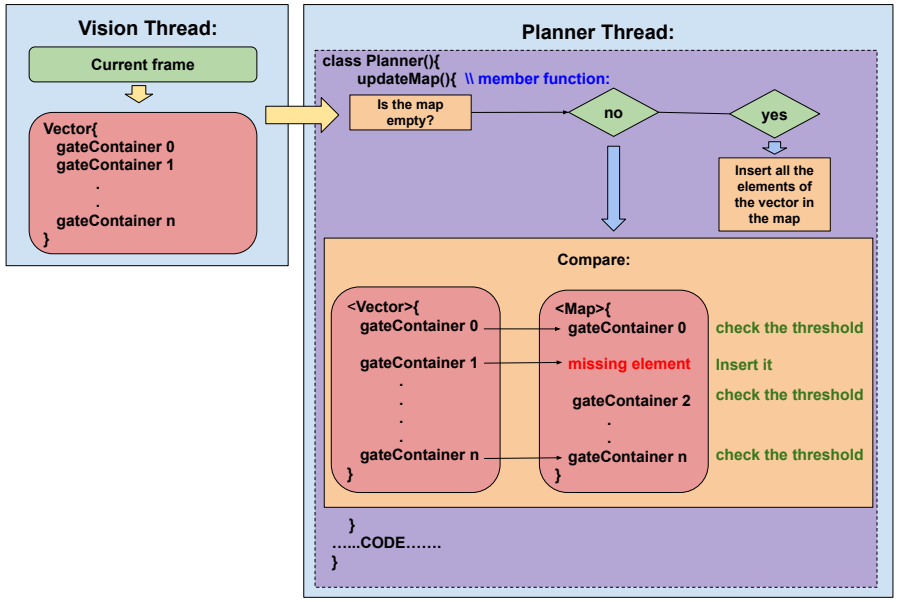

Figure 5.6: The figure shows the algorithm for creating and updating the mission map starting from the structures received by the vision task, whose information refers only to the current frame.

This way the trajectory is only recalculated when the endpoint is updated in the map. Finally, if the map is empty, the *updateMap()* function proceeds to insert all the structures present in *gateContainer*. The Eq. 5.15 defines the method used to decide if the element in the map should be updated or not.

$$\vec{d} = |\vec{p}_{current} - \vec{p}_{old}| < \vec{\eta} \tag{5.15}$$

Where $\vec{d}$ is the difference between the current position and the old posi-

tion saved in the map. If $\vec{d}$ is less than $\vec{\eta}$ the map is not updated, otherwise the map is updated with the new position present in *gateContainer*.

To create and manage the mission map we used the *std::map* class, which is part of the standard C++ libraries. The *std::map* element is a key-value associative container, i.e. the search is done on the basis of the key of interest. In particular, the ID of the gate (defined by the marker associated with the gate) is entered in the key field, while the coordinates of the endpoints of the associated gate are entered in the value field. The search in the mission map takes place using the ID of the gate to which the relative coordinates are associated. This approach guarantees a low computational cost (thanks to the use of the standard C++ library) and an easy management and updating of the mission map.

5.6.1 Operating logic of the mission supervisor

In this chapter, the drone's objective is to complete a complex mission based on the detection of n gates present in the environment. Since the drone does not know the position of the gates, the control tasks, estimate and generating setpoints are no longer sufficient for autonomous navigation. In fact, it is necessary a task able to supervise the autonomous flight phase during the execution of the mission, making appropriate choices.

For this purpose, the *plannerCore()* function was written (from now on it will be called *supervisor*), which in addition to calling the *updateMap()* function (introduced in the previous paragraph and responsible for creating and updating the mission map), supervises the generation of endpoints [90]. In particular, in the event that the drone is in autonomous hovering mode, it is limited to generating a fixed endpoint. On the other hand, if the mission is enabled, the function receives in input the desired sequence of gates to be crossed, (defined by the ID of the markers), supplying the two endpoints related to the first gate to be crossed in output. Once the current gate has been traversed the supervisor updates the endpoints with those relative to the next gate, this process is iterated until the mission is completed.

In the event that the coordinates of the next gate are not present in the mission map, the function makes the drone enter the hovering mode, waiting to detect the position of the gate of interest. Figure 5.7 shows the sequence of calls made by supervisor to complete the mission. In particular, after searching for endpoints (through calls to *searchGateInMap()* and *getElementByMap()*), the supervisor communicates them to the *updateEndpoint()*

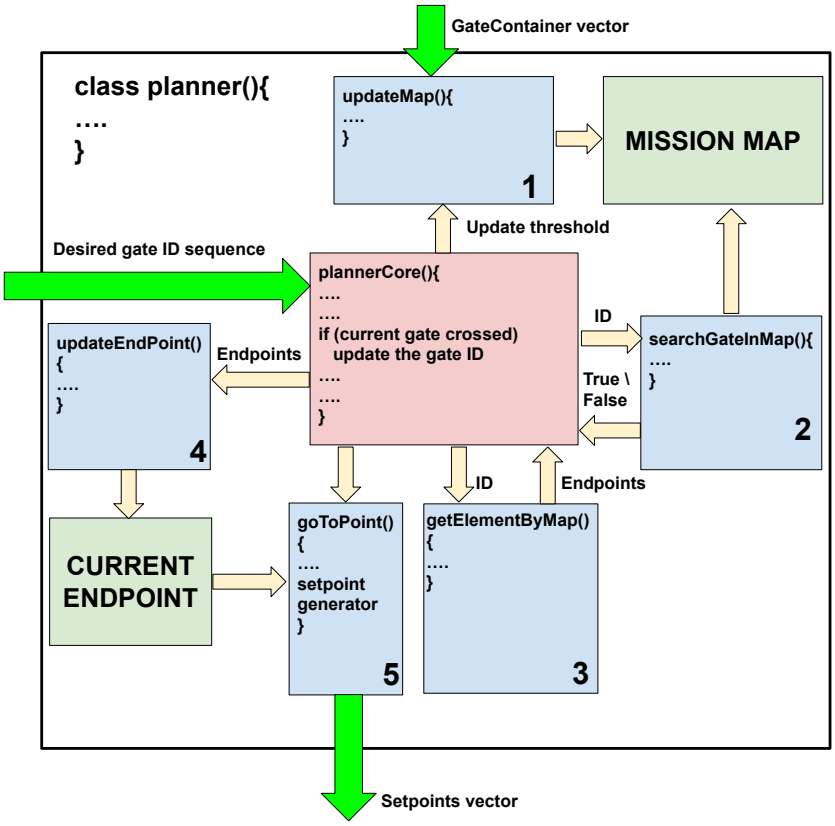

Figure 5.7: The figure shows the order of function calls made by the supervisor to complete the mission. These calls are iterative and repeat as each gate passes. Note that the supervisor sends the *searchGateInMap()* function the current gate ID [66, 67], which returns a Boolean. If the value assumed by the boolean is true, the supervisor proceeds to make subsequent calls. Otherwise it puts the drone into hovering mode, waiting for the current gate coordinates to be entered in the mission map.

function, which updates the current endpoints. Immediately afterwards it makes a call to the *goToPoint()* function, which has the task of generating the reference trajectories to be sent outside the *planner()* class.

5.7 Display thread

In addition to the thread for the data log, which has been present since the first software implementations, in this latest version a new thread called *display()* has been added. This thread is separate from the vision algorithm and during bench tests is able to show the video streaming preprocessed by the vision algorithm (within the vision thread). In addition, it can superimpose telemetry with the most interesting data on the frames.

In some cases, in addition to the numerical data saved in a text file through the logging thread, it may be useful to also have the video of the mission (which can be used for data postprocessing). For this purpose, if required, the *display()* thread is able to save the video stream coming from the vision thread in real-time and overlay the telemetry data. This approach was very useful during experimental tests, as it made it easier to resolve some anomalous behaviors. Figure 5.8 shows a frame relating to a mission consisting of the identification of 3 gates, where only the markers have been used for simplicity.

The frame was captured through the use of the *display()* thread, in addition to the telemetry it shows the two current endpoints to be reached to pass the first gate. At the top left is visible the current gate ID, which in this case corresponds to code 0. Below the target ID there are the coordinates in the fixed frame of the drone, while the error vector is shown at the top right: about 2.5cm for position and 1.5 degrees for orientation. At the bottom right it is possible to see the attitude vector of the drone and on the left the flight mode.

5.8 Experimental tests

The first experiment done with the new DART-2.0 platform was to repeat the straight trajectory shown in chapter 4 (with the same speed) in order to make a comparison between the two technologies. The following table shows the standard deviations on the individual degrees of freedom.

Axis	Standard deviation
horizontal (along x)	2.334 cm
altitude (along y)	1.129 cm
distance (along z)	3.694 cm

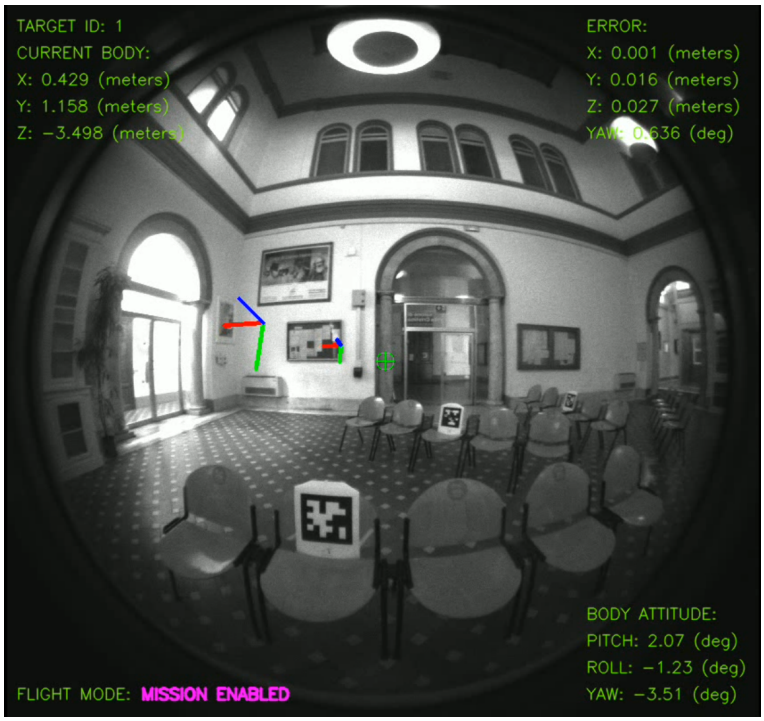

Figure 5.8: The figure shows a frame captured by the thread *display()* showing the view of the drone during the autonomous mission.

The results of these tests show how the performances obtained with the DART-2.0 platform (under the same conditions and with the same control technique used in the experimental tests of section 4.6) are better than those obtained with the DART-1.2 platform. Therefore, the reduction in latency on the new platform positively impacted the performance of the UAV. Above all, in the precision of the altitude control, where the problem of delay is most felt. Subsequently, the attitude estimate provided by the Intel stereoscopic camera was compared with the estimate produced through the use of the IMU used in the old DART platforms. This comparison was made by keeping the drone hovering for about 1 minute. Figure 5.9 shows the trend of the estimates produced by the two different sensors.

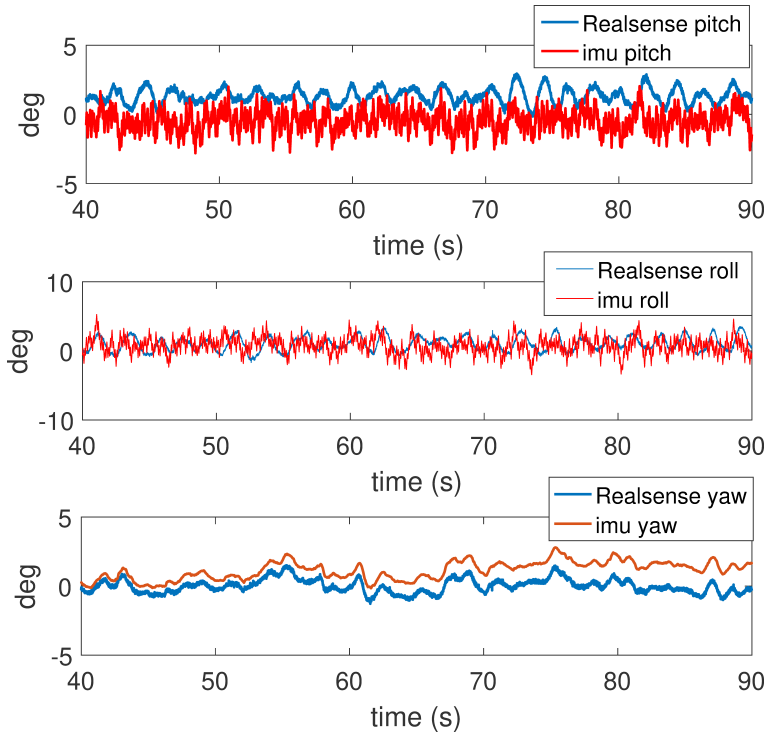

Figure 5.9: The graph shows a comparison of the attitude estimate produced by the Intel camera, with the estimate obtained by the old system based on the IMU Pololu-2739 AltIMU-10 v5. The new sensor in addition to being less noisy, significantly decreases the drift of Yaw's estimate. This is because the new estimate not only uses the gyroscope, but also uses the vision process to reduce the problem of drift.

Both the pitch and roll angles coming from the estimate made with the old IMU (Pololu-2739 AltIMU-10 v5) have a higher noise content than those coming from the Intel camera (which can clearly distinguish even lower attitude variations to the degree). This is mainly due to the higher quality of Intel hardware, which also has an order of magnitude higher cost. As for the yaw estimate, it is possible to observe that after a few tens of seconds there is a divergence between the two measures. Since static measurements were made with the Intel camera, where no yaw drift was found, it can reason-

ably be said that the drift during hovering is attributable to the old IMU. In fact, the estimate of yaw produced by the Intel camera does not use only the gyroscope (which as known is naturally subject to drift), but also uses the visual information (coming from the two onboard cameras) to cancel the drift of the gyroscope. Furthermore, once the additional supervisor code was written, flight tests were carried out to validate the developed algorithms, demonstrating that the drone is indeed able to complete the complex mission introduced in section 5.2. With reference to figure 5.8, graph 5.10 shows the 3D trajectory made by the drone during the complex mission and the relative setpoint.

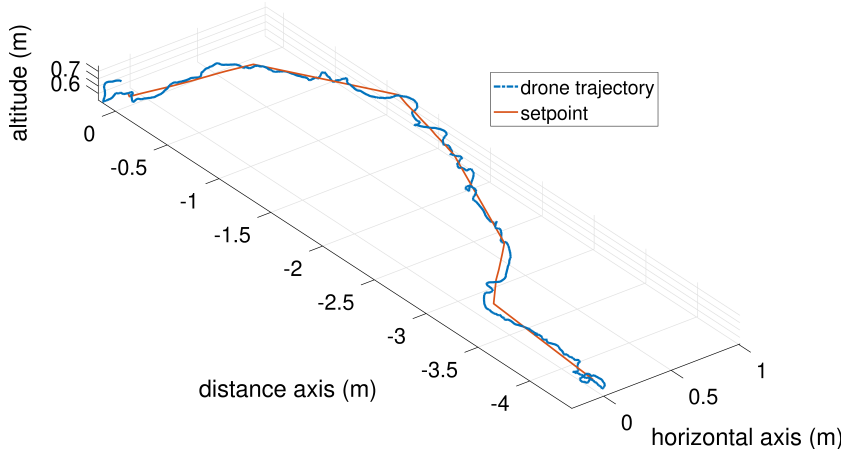

Figure 5.10: The graph shows a 3D view of the autonomous mission made by the drone. In red it is possible to see the desired trajectory generated in real-time and which depends on the detection of the gates. Instead in blue it is possible to observe the trajectory tracking.

As can be seen, the supervisor was able to correctly manage all phases of the mission. By revealing the position of all gates and generating the trajectories necessary for the termination of the mission. Given the short length of the mission, the experiment was conducted at a speed of 0.1 $\frac{m}{s}$. To better appreciate the tracking of trajectories, figure 5.11 also shows the trend of the individual axes. The trend on the individual degrees of freedom highlights the high performance achievable by this technology based on computer vision. In fact, the trajectory tracking turns out to be very faithful.

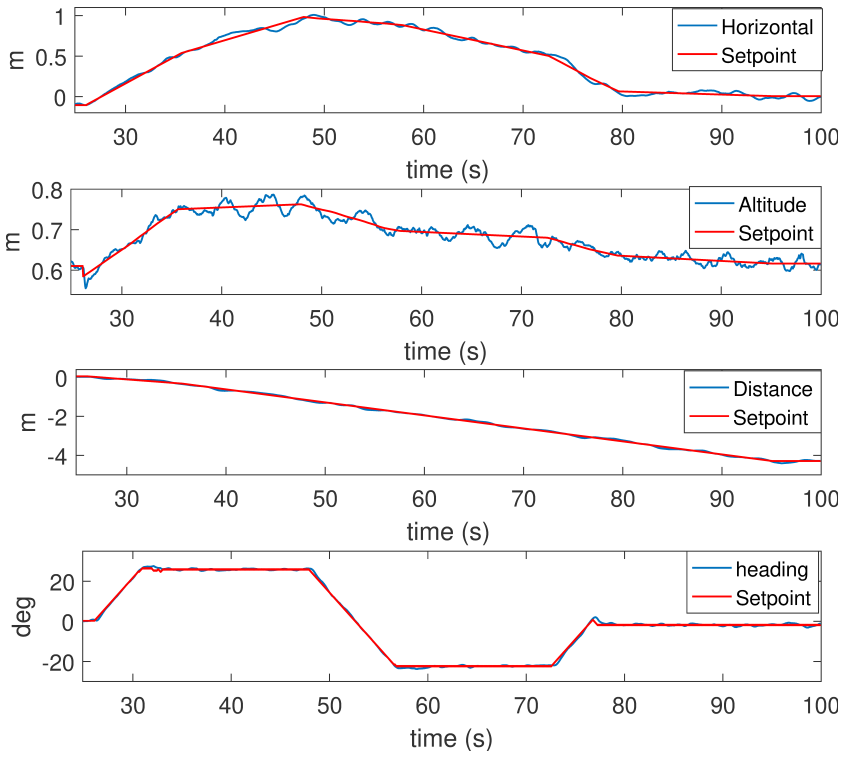

Figure 5.11: The figure shows the progress of the path following on the individual axes. The last graph below shows the trend of the yaw angle (in degrees), which is very faithful to the setpoint.

Axis	Standard deviation
horizontal (along x)	2.996 cm
altitude (along y)	1.253 cm
distance (along z)	4.364 cm
yaw	0.6427 deg

The table below shows the average standard deviations obtained on multiple runs of the same mission. As can be observed, the standard deviations of the error vector are slightly higher than those obtained during the following

tests of the straight trajectory. This is mainly due to two factors: (i) in this mission all the degrees of freedom of the drone are involved, ie the heading angle of the drone varies over time according to the orientation of the gates; (ii) in the current implementation, the trajectories that are generated do not take into account the acceleration and this can lead to discontinuous transients, which can then generate small overshoots on the true trajectory traveled by the drone. The last row of the table shows the standard deviation of the yaw error, which is just over 0.5 degrees. In fact, as can be seen from graph 5.11, the trend of the yaw angle is perfectly superimposed on the setpoint.

Finally, the mission was repeated at a higher speed of 0.8 $\frac{m}{s}$. Indeed, this speed represents a limiting case considering the short path under consideration, the trajectories to be followed and the little margin of error available. From this experiment, repeated on several tests, the mean standard deviations shown in the following table were then obtained.

Axis	Standard deviation
horizontal (along x)	4.144 cm
altitude (along y)	1.944 cm
distance (along z)	4.319 cm
yaw	2.217 deg

From these results it is evident that the degree of freedom that undergoes the greatest decline in performance is the horizontal axis. This behavior is logical, since the greatest variations on the setpoints occur on this degree of freedom. The average standard deviation of the error on the angle of yaw is quadrupled compared to the tests performed at a lower speed. However, this behavior is not due to the high level controller, but to the low level controller, which (as it is programmed) does not allow yaw variations higher than a certain speed.

5.9 Hardware in the loop

One of the most important problems for those who research in the avionics sector with real autonomous UAVs is the difficulty in testing the software developed for these systems. In fact, UAVs are naturally unstable systems and in the case of software bugs they can easily be damaged or pose a

danger to things or people. For this purpose, a graphic flight simulator has been developed [91, 92], which through a hardware in the loop technique is able to test all the high-level software that runs on the Nvidia Jetson Nano board. The flight simulator runs on a remote workstation, where

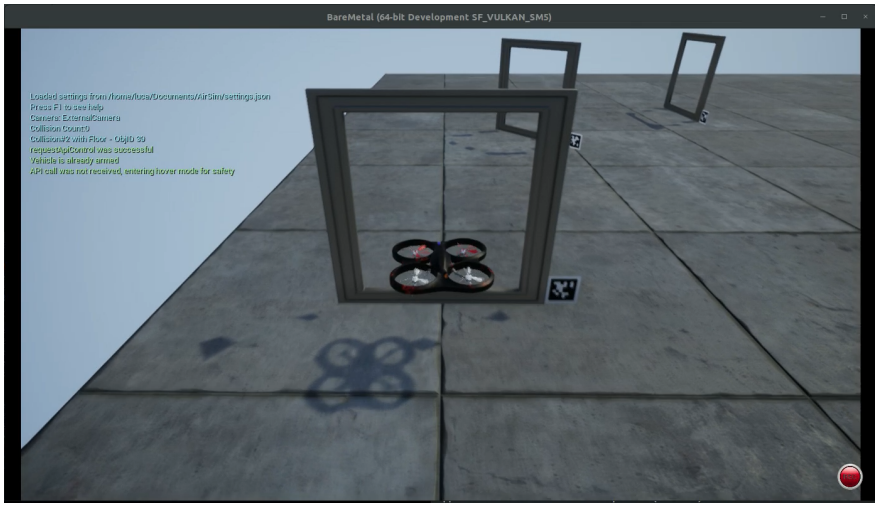

Figure 5.12: A frame of the synthetic environment developed through Unreal Engine, where the simulated drone is remotely controlled by the high-level software running on the Jetson Nano board.

a synthetic environment has been recreated through the Unreal graphics engine. Figure 5.12 shows a frame of the synthetic environment developed through Unreal engine, where you can see the simulated UAV controlled by the high-level software running on the real UAV remotely. The simulator is divided into two parts: (i) synthetic environment (graphic part using Unreal Engine); (ii) dynamics simulator, which contains the mathematical model part. To ensure that the dynamics simulator is not tied to the frame refresh rate of the synthetic environment, it runs in a separate program at a much higher rate. The dynamics simulator currently implemented makes use of the *AirSim* API [93,94] (open source project developed by *Microsoft*), which is able to communicate the state of the drone to the synthetic environment. To close the hardware in the loop ring, a special library called *dart sim utils* has been written, which uses the UDP protocol (the TCP protocol was

discarded because it is less suitable for real time applications). The library is installed both on the workstation and on the DART-2.0/1 platform, this allows the dynamics simulator to send (through an Ethernet cable connected to the Jetson Nano board) the simulated frames and state of the UAV (a simulated frame is shown in figure 5.13). These data are processed by the

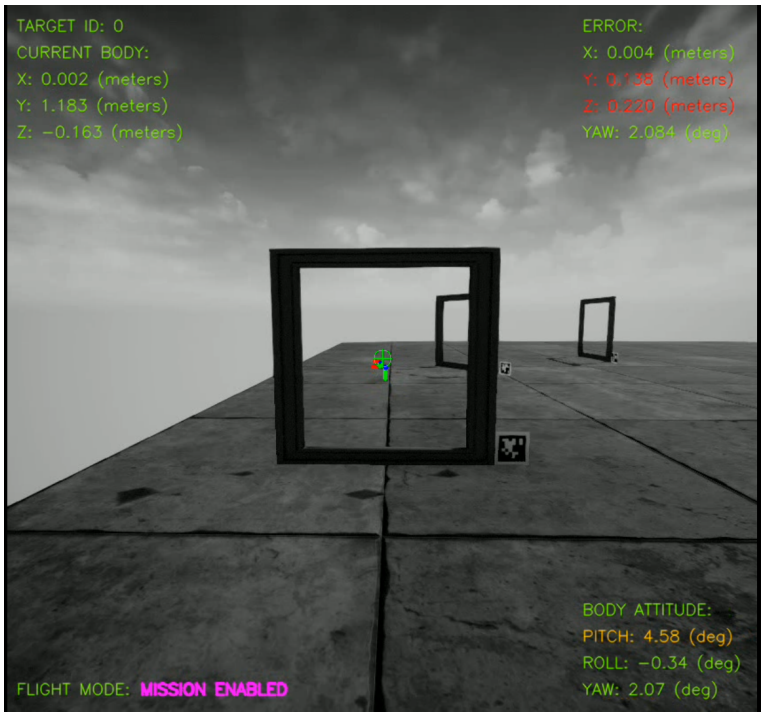

Figure 5.13: A simulated frame received by the DART-2.0/1 platform and processed by the vision algorithm (in fact, the telemetry showing the mission data is distinguished). In the frame it is possible to observe the synthetic environment and the gates identified by the AprilTag markers.

high-level software that runs on board the drone, which (through the dart sim utils client libraries) is able to send the autonomous commands to the remote workstation (instead of sending them to the low-level controller). This approach allows you to simulate autonomous missions by testing all the software developed, allowing you to reveal the presence of any bugs in

safety. By optimizing the software through the hardware acceleration of the Jetson nano board, a latency of the simulated frames of about 5 ms was obtained. This value being comparable with the latency declared by Intel for their stereoscopic camera, makes the simulation very realistic. Figure 5.14 shows the block diagram of the software layers present on the workstation and on the DART-2.0/1 platform necessary to close the hardware in the loop ring.

5.9.1 Simulated experimental tests

Once the simulator was finished and interfaced with the DART-2.0 / 1 platform, the same mission carried out in the real world was repeated in the synthetic environment, in order to demonstrate the realism of the simulation.

In fact, the process of receiving (trough the UDP channel) and unpacking the simulated video stream is not obvious, as the main difficulty is to keep the time delay equivalent to that of the real video stream. If this condition is not respected the resulting simulation will be not very faithful to the real behavior of the software algorithm. In fact, in the literature the papers that investigate the development of hardware in the loop techniques for UAVs do not contemplate sending a simulated video stream(due to the great implementation difficulty of these systems, depending on the technology currently available). Figure 5.15 shows the complete trajectory made by the drone while crossing the gates in the synthetic environment. As can be seen, all the algorithms mentioned in this chapter worked correctly also in the synthetic environment, demonstrating the quality of the simulation.

The trajectory in blue is generated by the supervisor (as in the real case), which aims to cross all the gates present in the desired sequence set a priori. The speed set for the simulated mission is 0.8 $\frac{m}{s}$ and is equivalent to that set for the real tests. The standard deviations of the error vector are shown in the following table.

Axis	Simulated standard deviation
horizontal (along x)	5.155 cm
altitude (along y)	6.857 cm
distance (along z)	6.811 cm
yaw	4.501 deg

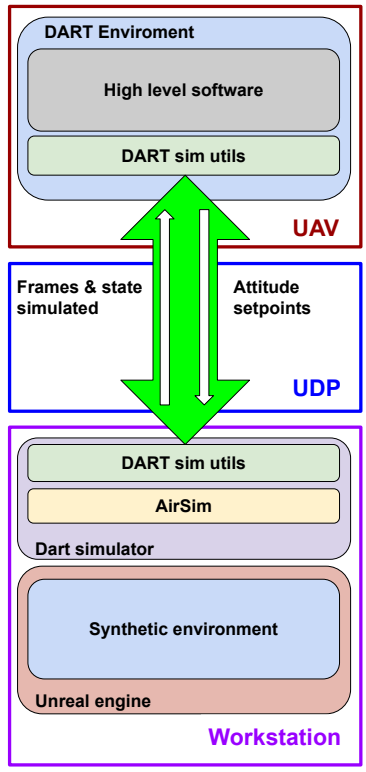

Figure 5.14: The block diagram shows the software layers present on the DART-2.0/1 platform and the remote workstation, required to perform a hardware in the loop simulation. The communication between the dynamics simulator and the UAV uses the dart sim utils API, through which it is possible to send state and frames of the simulated drone. The UAV through the same API is able to communicate autonomous commands to the dynamics simulator by closing the loop.

The decrease in performance observed in the table is solely due to the mathematical model used to simulate the dynamics of the drone. As currently the mathematical model used in the flight simulator is a generic model preexisting in the AirSim libraries [95]. In the next implementation of the flight

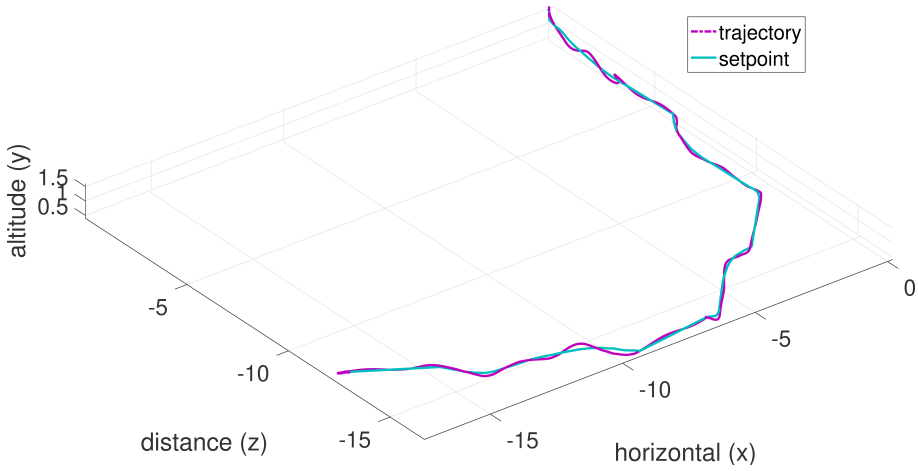

Figure 5.15: The graph shows the simulated mission, where in blue it is possible to see the desired trajectory generated by the supervisor and in purple the trajectory of the UAV. The scale is in meters and the total length of the mission is about 20 meters.

simulator, the generic model will be replaced with the mathematical model of the DART-2.0/1 platform. In this way, the calibration of the control module obtained in the real world will guarantee equivalent performance even in the synthetic environment.

5.10 Complete software architecture

In this section we show the overall architecture of the high-level software and its operating logic. The software features are dynamic and can be managed from the command line by inserting appropriate strings from the terminal. The following list shows the commands and related functions currently implemented.

- $--$***displaySampleTime*** : Shows active threads, expected and real sample rates.
- $--$***windows*** : Shows the images processed by the vision algorithm inside a window.

- $--$***plot*** : Makes the real time plot of the variables of interest inside a window (it also needs the *–windows* command).
- $--$***log*** : Save all shared variables in a *.txt* file, usable for data post processing.
- $--$***rec*** : Records the video stream processed by the vision algorithm and superimposes the telemetry obtained in real time.
- $--$***tagSize*** : Defines a marker size other than the default one.
- $--$***sim*** : Enable hardware-in-the-loop mode.
- $--$***ssh*** : If enabled, it eliminates the *OpenGL* optimization of the windows which are rendered on the remote PC.

Features management through appropriate input commands avoids recompiling the software at each experimental test. in addition it guarantees the dynamic management of the threads that must go into execution. For instance, if the $--$*windows* and $--$*rec* commands are not executed, the display thread is not initialized as it is not necessary. This approach optimizes software by avoiding unnecessary computational waste. Figure 5.16 shows the overall software architecture, where it is interesting to observe how the real and simulated flight modes are managed.

If the $--$*sim* option is not passed, the part of the code that manages the data stream coming from the stereoscopic camera is activated in the slam thread (thread 1). Furthermore, in the communication thread (thread 5) the sender part is activated, which is responsible for the communication with the mixer board (mid-level hardware). Vice versa, if the $--$*sim* option is inserted in the slam thread, the part of the code that manages the data flow from the Realsense camera is deactivated. In its place another piece of code is activated that simulates the management of Realsense data (responsible for receiving the video stream and the simulated state sent by the simulator, which runs on the remote workstation).

Furthermore, in the communication thread there is a substitution between the sender part that manages the communication with the mixer (necessary for the real flight mode) and the sender part that manages the sending of command signals to the workstation (necessary for closing the hardware loop in the loop during simulated flight mode). Threads 6, 7 and 8 are optional and can only be activated if required. These threads are useful during

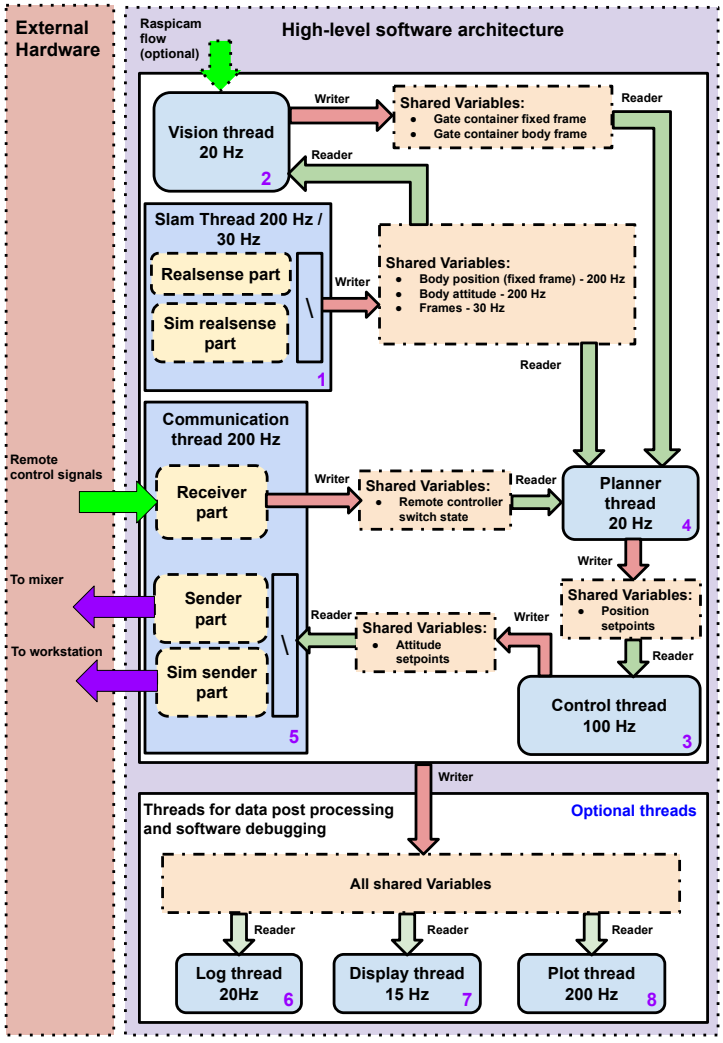

Figure 5.16: The figure shows the complete block diagram of the software architecture that runs on the Nvidia Jetson Nano card. It is important to observe how the yellow dotted blocks within threads 1 and 5 are mutually exclusive and depend on the activation of the real or simulated flight mode.

the experimental phases, as they allow to acquire data for the post processing phases and for the debugging phases. This group, made up of 3 threads, is able to receive all the shared variables within the software environment. Very useful feature during the debugging phase, which may require in-depth analysis on the evolution of the state of the variables. It should be noted that thread 1 has a double sampling frequency, equal to $200Hz$ and $30Hz$.

This derives from the fact that it has a double routine, the first at higher frequency manages the Slam data containing the state of the drone, and a second slower routine that manages the video stream of the Realsense camera lenses. The vision thread, if required, is also able to receive the video stream coming from the Raspicam camera, useful in case you want to increase the depth of field (as it has a longer focal length than the lenses used by the Realsense camera). Finally, as for thread sample rates, they are specified within individual blocks of figure 5.16.

5.11 Conclusion

In this chapter we have shown the approaches and software entities necessary to make a UAV capable of performing autonomous missions with unknown trajectories. An important aspect, which will play a fundamental role in the near future, will be that of making these systems increasingly capable of making choices according to different scenarios. The performances that these autonomous UAVs can achieve are of the order of centimeter, this will make them able to carry out increasingly complex activities currently unimaginable. The only brake on the development of technology in the UAV sector is represented by the difficulty of testing the algorithms that are developed in the academic field and by the stringent regulations that limit their flight in open environments. For this purpose, the proposed hardware in the loop technique has shown a solution to the problem, demonstrating that it is possible to test all algorithms safely and faithfully to reality.

Chapter 6

Conclusion

In the previous chapters all the technological difficulties to obtain autonomous UAVs that use on-board VBN techniques have been highlighted. One of the biggest challenges was the drop in performance due to the inevitable presence of vision noise, given that the cameras (unlike off-board VBN techniques) were mounted on board the UAV itself. In fact, the problem of vision noise would not have existed if off-board VBN techniques had been adopted. Another significant difficulty was the latency due to the image processing times (depending of the computer vision module) and the intrinsic delay of the frames sent by the camera.

As shown in the previous chapters, this problem is closely linked to the type of technology used and is difficult to solve by using sole sensor fusion techniques. It is important to reiterate the enormous advantages that the techniques developed in this PhD work can guarantee in terms of flexibility and usability. Furthermore, by optimizing the artificial vision and control process, laboratory tests have shown that the accuracies obtainable from these UAV prototypes are of the order of a centimeter (very close to the performance of the systems based on off-board techniques). An equally important aspect was the study and design of multi-board communication protocols, which allowed the separation of tasks in different hardware levels. Thanks to the study of the state of the art, it was possible to identify the best electronics for the project.

All this work has permitted the creation of a development platform for autonomous UAVs, called DART, completely open and modular, which allows the replacement of individual components in a simple and efficient way.

This made possible to use a smaller autopilot board (compared to the PixHawk), allowing the creation of very small drones that can be easily tested in indoor environments.

One of the greatest advantages of the DART platform is that it is low cost, especially when compared with platforms often used for research purposes that use mocap systems. However, the real killer application of the DART platform is the hardware-in-the-loop simulation capabilities that have allowed the high-level software to be tested in safety. In fact, the hardware-in-the-loop technique proposed in section 5.9 is currently very innovative, as it is possible to simulate the on-board VBN techniques themselves with high fidelity. At the following link there is a video showing some of the development phases and experiments that have been carried out during this PhD activity: `https://www.youtube.com/watch?v=6OgawvmNleM` .

6.1 Directions for future work

In the latest versions of the DART platform (2.0 / 2.1) the hardware has improved significantly, making it much more flexible than its previous versions. This aspect will lead, in the near future, to continue development by adding new software modules and substituting others. The modular nature of the software will allow to test new control techniques, different from the classical ones based on PID, for example geometric control techniques such as those proposed by Mellinger [96]. Another interesting research branches could focus on: (i) the replacement of the computer vision module with others that do not use environmental markers; (ii) the introduction of the exploration algorithms based on artificial intelligence [97, 98] and (iii) the improvement of the path planning module [99]. It is worth underlining that the project aims to become a Spin-off of the University of Florence, thanks to the peculiar characteristics of the DART platform.

Appendix A

Publications

This research activity has led to scientific publications and additional results which are summarized below.

International Journals

1. Luca Bigazzi, Stefano Gherardini, Giacomo Innocenti and Michele Basso "Development of non expensive technologies for precise maneuvering of completely autonomous unmanned aerial vehicles", *Sensors*, vol. 21, no. 2: 391, 2021.

International Conferences

1. Michele Basso, Luca Bigazzi, Giacomo Innocenti. "DART Project: A High Precision UAV Prototype Exploiting On-board Visual Sensing", in *The Fifteenth International Conference on Autonomic and Autonomous Systems (ICAS 2019)*, 5, June 2-6, 2019, Athens, Greece, ISBN 978-1-61208-712-2.
2. **Accepted for publication:** Luca Bigazzi, Michele Basso, Stefano Gherardini and Giacomo Innocenti. "Mitigating latency problems in vision-based autonomous UAVs", in *The 29th Mediterranean Conference on Control and Automation (MED 2021)*, June 22-25, Bari, Puglia.

Awards and acknowledgments

1. **Winner of Impresa Campus 2020**, "With a project aimed at precision agriculture called DronePorto", *during the training course promoted by the Incubatore Universitario Fiorentino (IUF)*, Dec 10, 2020, Florence.

2. Luca Bigazzi, Luigi Pannocchi, Stefano Gherardini, Enrico Boni and Michele Basso, "DART Project", participation in the pre-incubation process promoted by the *Incubatore Universitario Fiorentino (IUF)*, Dec, 2020, Florence.

Bibliography

[1] K. W. Chan, U. Nirmal, and W. G. Cheaw, "Progress on drone technology and their applications: A comprehensive review", AIP Conference Proceedings, vol. 2030 (020308), 2018.

[2] Shakhatreh, Hazim and Sawalmeh, Ahmad H and Al-Fuqaha, Ala and Dou, Zuochao and Almaita, Eyad and Khalil, Issa and Othman, Noor Shamsiah and Khreishah, Abdallah and Guizani, Mohsen, "Unmanned aerial vehicles (UAVs): A survey on civil applications and key research challenge", IEEE Access, vol. 7, 2019, pp. 48572–48634.

[3] R. Shakeri, M. A. Al-Garadi, A. Badawy, A. Mohamed, T. Khattab, A. K. Al-Ali, K. A. Harras, and M. Guizani, "Design challenges of multi-UAV systems in cyber-physical applications: A comprehensive survey and future directions", IEEE Communications Surveys & Tutorials, vol. 21, no. 4, 2019, pp. 3340–3385.

[4] X. Wang, Z. Huang, G. Sui et al., "Analysis on the development trend of future UAV equipment technology", Academic Journal of Engineering and Technology Science, vol. 2, no. 1, 2019.

[5] H. Ghazzai, H. Menouar, A. Kadri, and Y. Massoud, "Future UAV-based ITS: A comprehensive scheduling framework", IEEE Access, vol. 7, 2019, 75678–75695.

[6] B.H. Younus Alsalam, K. Morton, D. Campbell, and F. Gonzalez, "Autonomous UAV with vision based on-board decision making for remote sensing and precision agriculture", in 2017 IEEE Aerospace Conference, 4–11 March 2017, Big Sky, MT, USA.

[7] P. Tokekar, J.V. Hook, D. Mulla, and V. Isler, "Sensor planning for a symbiotic UAV and UGV system for precision agriculture", IEEE Transactions on Robotics, vol. 32, no. 6, Dec 2016, pp. 1498-1511.

[8] J. Kim, S. Kim, C. Ju, and H. I. Son, "Unmanned aerial vehicles in agriculture: A review of perspective of platform, control, and applications", IEEE Access, vol. 7, 2019, pp. 105100–105115.

[9] P. Henkel, U. Mittmann, and M. Iafrancesco, "Real-time kinematic positioning with GPS and GLONASS", in 2016 24th European Signal Processing Conference (EUSIPCO), Aug 2016, pp. 1063-1067.

[10] S. Jordan, J. Moore, S. Hovet, J. Box, J. Perry, K. Kirsche, D. Lewis, and Z. T. H. Tse, "State-of-the-art technologies for UAV inspections", IET Radar, Sonar & Navigation, vol. 12, 2018, pp. 151–164.

[11] M. A. Dinesh, S. Santhosh Kumar, J. Sanath, K. N. Akarsh, and K. M. Manoj Gowda, "Development of an Autonomous Drone for Surveillance Application", Proceedings of International Research Journal of Engineering and Technology (IRJET), vol. 5, 2018.

[12] H. Zhou, H. Kong, L. Wei, D. Creighton, and S. Nahavandi, "Efficient road detection and tracking for unmanned aerial vehicle", IEEE Transactions on Intelligent Transportation Systems, vol. 16, no. 1, Feb 2015, pp. 297-309.

[13] V.P. Srini, "A vision for supporting autonomous navigation in urban environments", Computer, vol. 39, no. 12, Dec. 2006, pp. 68-77.

[14] K. Nonami, "Research and Development of Drone and Roadmap to Evolution", Journal of Robotics and Mechatronics, vol. 30, no. 3, 2018, pp. 322–336.

[15] M. Elloumi, R. Dhaou, B. Escrig, H. Idoudi, and L. A. Saidane, "Monitoring road traffic with a UAV-based system", 2018 IEEE Wireless Communications and Networking Conference (WCNC), 2018, pp. 1–6.

[16] Todorovic, B., and Orlic, V., "Direct sequence spread spectrum scheme for an unmanned aerial vehicle PPM control signal protection", IEEE Communications Letters, vol. 13, 2009, pp. 727–729.

[17] Zhenming Li, Mingjie Lao, Swee King Phang, Mohamed Redhwan Abdul Hamid, Kok Zuea Tang, and Feng Lin, "Development and Design Methodology of anAnti-Vibration System on Micro-UAVs", 9th INTERNATIONAL MICRO AIR VEHICLES (IMAV), Sep 18-21 2017, Toulouse, France.

[18] Vu, N. A., Dang, D. K., and Le Dinh, T., "Electric Propulsion System Sizing Methodology for an Agriculture Multicopter", Aerospace Science and Technology, vol. 90, 2019, pp. 314-326.

[19] Bryson, M., and Sukkarieh, S., "Building a Robust Implementation of Bearing-only Inertial SLAM for a UAV", Journal of Field Robotics, vol. 24, 2007, pp. 113–143.

[20] Bu, S., Zhao, Y., Wan, G., and Liu, Z, "Map2DFusion: Real-time incremental UAV image mosaicing based on monocular SLAM" 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct 9-14, 2016, Daejeon, South Korea.

[21] Agarwal, A., Crouse, J. R., and Johnson, E. N., "Evaluation of a Commercially Available Autonomous Visual Inertial Odometry Solution for Indoor Navigation", 2020 International Conference on Unmanned Aircraft Systems (ICUAS), Sept 1-4, 2020, Athens, Greece.

[22] Kutay Yilmaz, Omer Faruk Kaya, Erkan Uslu, "Indoor UAV Localization and 3D Mapping Using Visual Odometry", 2020 Innovations in Intelligent Systems and Applications Conference (ASYU), Oct 15-17, 2020, Istanbul, Turkey, Turkey.

[23] Basulto-Lantsova, A., Padilla-Medina, J. A., Perez-Pinal, F. J., and Barranco-Gutierrez, A. I., "Performance comparative of OpenCV Template Matching method on Jetson TX2 and Jetson Nano developer kits", 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), Jan 6-8, 2020, Las Vegas, NV, USA.

[24] Sieberth, T., Wackrow, R., and Chandler, J. H., "Automatic detection of blurred images in UAV image sets", ISPRS Journal of Photogrammetry and Remote Sensing, vol. 122, 2016, pp. 1–16.

[25] Antonio Bacciaglia, Dian Guo, Pier Marzocca, Cees Bi, Alessandro Ceruti, "BIMODAL UNMANNED VEHICLE: PROPULSION SYSTEMINTEGRATION AND WATER/AIR INTERFACE TESTING", 31st Congress of the International Council of the Aeronautical Sciences (ICAS 2018), Sep 9-14, 2018, Belo Horizonte, Brazil.

[26] Segarra, D., Caballeros, J., Aguilar, W. G., SamÃ , A., and RodrÃguez-MartÃn, D., "Orientation Estimation Using Filter-Based Inertial Data Fusion for Posture Recognition" Lecture Notes in Computer Science, 2019, pp. 220-233.

[27] Pierleoni, P., Belli, A., Palma, L., Pernini, L., and Valenti, S., "An accurate device for real-time altitude estimation using data fusion algorithms", 2014 IEEE/ASME 10th International Conference on Mechatronic and Embedded Systems and Applications (MESA), Sept 10-12, 2014, Senigallia, Italy.

[28] L. Cheng, L. Zhong, S. Tian, and J. Xing, "Task Assignment Algorithm for Road Patrol by Multiple UAVs With Multiple Bases and Rechargeable Endurance", IEEE Access, vol. 7, 2019, pp. 144381–144397.

[29] M. Erdelj, M. Król, and E. Natalizio, "Wireless sensor networks and multi-UAV systems for natural disaster management", Computer Networks, vol. 124, 2017, pp. 72–86.

[30] H. Ren, Y. Zhao, W. Xiao, Wu and Z. Hu, "A review of UAV monitoring in mining areas: Current status and future perspectives", International Journal of Coal Science & Technology, 2019, pp. 1–14.

[31] M. Erdelj, E. Natalizio, K. R. Chowdhury, R. Kaushik, and I. .F. Akyildiz, "Help from the sky: Leveraging UAVs for disaster management", IEEE Pervasive Computing, vol. 16, no. 1, 2017, pp. 24–32.

[32] R. Avanzato, and F. Beritelli, "A Smart UAV-Femtocell Data Sensing System for Post-Earthquake Localization of People", IEEE Access, vol. 8, 2020, pp. 30262–30270.

[33] L. Bigazzi, S. Gherardini, G. Innocenti and M. Basso, "Development of Non Expensive Technologies for Precise Maneuvering of Completely Autonomous Unmanned Aerial Vehicles" Sensors, vol. 21, no. 2: 391, 2021.

[34] Aljehani, Maher and Inoue, Masahiro, "Performance evaluation of multi-UAV system in post-disaster application: Validated by HITL simulator", IEEE Access, vol. 7, 2019, pp. 64386–64400.

[35] Alotaibi, Ebtehal Turki and Alqefari, Shahad Saleh and Koubaa, Anis, "Lsar: Multi-uav collaboration for search and rescue missions", IEEE Access, vol. 7, 2019, pp. 55817–55832.

[36] S.M. Khansari-Zadeh, and F. Saghafi, "Vision-Based Navigation in Autonomous Close Proximity Operations using Neural Networks", IEEE Transactions on Aerospace and Electronic Systems, vol. 47, no. 2, April 2011, pp. 864–883.

[37] J. Zhang, W. Liu, and Y. Wu, "Novel Technique for Vision-Based UAV Navigation", IEEE Transactions on Aerospace and Electronic Systems, vol. 47, no. 4, October 2011, pp. 2731-2741.

[38] J. Zhang, Y. Wu, W. Liu, and X. Chen, "Novel Approach to Position and Orientation Estimation in Vision-Based UAV Navigation", IEEE Transactions on Aerospace and Electronic Systems, vol. 46, no. 2, April 2010, pp. 687–700.

[39] W.G. Aguilar, V.S. Salcedo, D.S. Sandoval, and B. Cobena, "Developing of a Video-Based Model for UAV Autonomous Navigation", in 2017 Latin American Workshop on Computational Neuroscience (LAWCN): Computational Neuroscience, pp. 94–105.

[40] B.M. Miller, K.V. Stepanyan, A.K. Popov, and A.B. Miller, "UAV navigation based on videosequences captured by the onboard video camera", Automation and Remote Control, vol. 78, December 2017, pp. 2211–2221.

[41] Y. Lu, Z. Xue, G.-S. Xia, and L. Zhang, "A survey on vision-based UAV navigation", Geo-spatial Information Science, vol. 21, no. 1, January 2018, pp. 21-32: Special Issue: Information Processing for Unmanned Aerial Vehicles (UAVs) in Surveying, Mapping, and Navigation. Guest Editors: Gui-Song Xia, Mihai Datcu, Wen Yang, Xiang Bai.

[42] A. Cesetti, E. Frontoni, A. Mancini, P. Zingaretti, and S. Longhi, "A Vision-Based Guidance System for UAV Navigation and Safe Landing using Natural Landmarks", Journal of Intelligent and Robotic Systems, vol. 57, no. 233, 2010.

[43] L.R. Garcia Carrillo, A.E. Dzul Lopez, R. Lozano, and C. Pegard, "Combining Stereo Vision and Inertial Navigation System for a Quad-Rotor UAV", Journal of Intelligent and Robotic Systems, vol. 65, 2012, pp. 373–387.

[44] I. Mademlis, A. Torres-González, J. Capitán et al., "A multiple-uav software architecture for autonomous media production", Workshop on Signal Processing Computer vision and Deep Learning for Autonomous Systems (EUSIPCO2019), 2019.

[45] I. Mademlis, N. Nikolaidis, A. Tefas, I. Pitas, T. Wagner, and A. Messina, "Autonomous UAV cinematography: a tutorial and a formalized shot-type taxonomy", ACM Computing Surveys (CSUR), vol. 52, no. 5, 2019, pp. 1–33.

[46] GPS Accuracy, Official U.S. government information about the Global Positioning System (GPS) and related topics. https://www.gps.gov/systems/gps/performance/accuracy/

[47] S. Zhao, Y. Chen, and J. A. Farrell, "High-precision vehicle navigation in urban environments using an MEM's IMU and single-frequency GPS receiver", IEEE transactions on intelligent transportation systems, vol. 17, no. 10, 2016, pp. 2854–2867.

[48] Q. Lu, Y. Zhang, J. Lin, and M. Wu, "Dynamic Electromagnetic Positioning System for Accurate Close-Range Navigation of Multirotor UAVs", IEEE Sensors Journal, vol. 20, no. 8, April 2020, pp. 4459–4468.

[49] Liu, Haowen and Yang, Bingen, "Quadrotor Singularity Free Modeling and Acrobatic Maneuvering", International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, vol. 59230, 2019, pp. V05AT07A055.

[50] Q. Yang, J. Zhang, G. Shi, J. Hu, and Y. Wu, "Maneuver decision of UAV in short-range air combat based on deep reinforcement learning", IEEE Access, vol. 8, 2019, pp. 363–378.

[51] R.P. Padhy, F. Xia, S.K. Choudhury, P.K. Sa, and S. Bakshi, "Monocular Vision Aided Autonomous UAV Navigation in Indoor Corridor Environments", IEEE Transactions on Sustainable Computing, vol. 4, no. 1, Jan.-March 2019, pp. 96–108.

[52] S. Grzonka, G. Grisetti, and W. Burgard, "A Fully Autonomous Indoor Quadrotor", IEEE Transactions on Robotics, vol. 28, no. 1, Feb. 2012, pp. 90–100.

[53] J.P. How, B. Behihke, A. Frank, D. Dale, and J. Vian, "Real-time indoor autonomous vehicle test environment", IEEE Control Systems Magazine, vol. 28, no. 2, April 2008, pp. 51–64.

[54] S. Yang, S.A. Scherer, X. Yi, and A. Zell, "Multi-camera visual SLAM for autonomous navigation of micro aerial vehicles", Robotics and Autonomous Systems, vol. 93, July 2017, pp. 116–134.

[55] T.T. Mac, C. Copot, R. De Keyser, and C.M. Ionescu, "The development of an autonomous navigation system with optimal control of an UAV in partly unknown indoor environment", Mechatronics, vol. 49, February 2018, pp. 187–196.

[56] Y.M. Mustafah, A.W. Azman, and F. Akbar, "Indoor UAV Positioning Using Stereo Vision Sensor", Procedia Engineering, vol. 41, 2012, pp. 575–579.

[57] J: Zhang, X. Wang, Z. Yu, Y. Lyu, S. Mao, S. C. G. Periaswamy, J. Patton, and X. Wang, "Robust rfid based 6-dof localization for unmanned aerial vehicles", IEEE Access, vol. 7, 2019, pp. 77348–77361.

[58] L. I. Balderas, A. Reyna, M. A. Panduro, C. Del Rio, and A. R. Gutiérrez, "Low-profile conformal UWB antenna for UAV applications", IEEE Access, vol. 7, 2019, pp. 127486–127494.

[59] W. You, F. Li, L. Liao, and M. Huang, "Data Fusion of UWB and IMU Based on Unscented Kalman Filter for Indoor Localization of Quadrotor UAV", IEEE Access, vol. 8, 2020, pp. 64971–64981.

[60] A. Gonzalez-Sieira, D. Cores, M. Mucientes, and A. Bugarin, "Autonomous navigation for UAVs managing motion and sensing uncertainty", Robotics and Autonomous Systems, vol. 126, April 2020, pp. 103455.

[61] N. Imanberdiyev, C. Fu, E. Kayacan, and I-M. Chen, "Autonomous navigation of UAV by using real-time model-based reinforcement learning", 2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV), 2016, pp. 1–6.

[62] J. Jimenez Lugo, and A. Zell, "Framework for Autonomous On-board Navigation with the AR.Drone", Journal of Intelligent and Robotic Systems, vol. 73, 2014, pp. 401–412.

[63] M. Basso, L. Bigazzi, G. Innocenti, "DART Project: A High Precision UAV Prototype Exploiting On-board Visual Sensing", in the 15th International Conference on Autonomic and Autonomous Systems (ICAS), 02-06 June 2019, Athens, Greece.

[64] J.-H. Kim, S. Sukkarieh, and S. Wishart, "Real-Time Navigation, Guidance, and Control of a UAV Using Low-Cost Sensors", Field and Service Robotics, vol. 24, 2006, pp. 299–309.

[65] N. Gageik, P. Benz, and S. Montenegro, "Obstacle detection and collision avoidance for a UAV with complementary low-cost sensors", IEEE Access, vol. 3, 2015, pp. 599–609.

[66] Olson, E., "AprilTag: A robust and flexible visual fiducial system", 2011 IEEE International Conference on Robotics and Automation, May 9-13, 2011, Shanghai, China.

[67] J. Wang, and E. Olson, "AprilTag2: Efficient and robust fiducial detection", in 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct 9-14 2016, Daejeon, Korea.

[68] S.O.H. Madgwick, "An efficient orientation filter for inertial and inertial/-magnetic sensor arrays", April 2010.

[69] R. Mahony, T. Hamel, and J.-M. Pflimlin, "Nonlinear Complementary Filters on the Special Orthogonal Group", IEEE Transactions on Automatic Control, vol. 53, no. 5, 26 Aug 26 2008, pp. 1203–1218.

[70] M. Euston, P. Coote, R. Mahony, Jonghyuk Kim, and T. Hamel, "A complementary filter for attitude estimation of a fixed-wing UAV", 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, 22-26 Sept. 2008, Nice, France.

[71] Martin, P., and Salaun, E., "The true role of accelerometer feedback in quadrotor control", 2010 IEEE International Conference on Robotics and Automation (ICRA), 3-7 May 2010, Anchorage, AK, USA.

[72] D. Gebre-Egziabher, R.C. Hayward, and J.D. Powell, "Design of multi-sensor attitude determination systems", IEEE Transactions on Aerospace and Electronic Systems, vol. 40, no. 2, Apr 2004, pp. 627-649.

[73] A.M. Sabatini, "Quaternion-based extended Kalman filter for determining orientation by inertial and magnetic sensing", IEEE Trans Biomed Eng., vol. 53, no. 7, Jul 2006, pp. 1346–56.

[74] Vandersteegen, M., Van Beeck, K., and Goedeme, T., "Super accurate low latency object detection on a surveillance UAV", 2019 16th International Conference on Machine Vision Applications (MVA), May 27-31 2019, Tokyo, Japan.

[75] Bayer, J., and Faigl, J., "On Autonomous Spatial Exploration with Small Hexapod Walking Robot using Tracking Camera Intel RealSense T265", 2019 European Conference on Mobile Robots (ECMR), Sept 4-6, 2019, Prague, Czech Republic.

[76] Jwu-Sheng Hu, Nelson Yen-Chung Chang, Jwu-Jiun Yang, Jyun-Ji Wang, Rodolfo Gondim Lossio, Ming-Chih Chien, Yung-Jung Chang, Chen-Yu Kai, and Shyh-Haur Su, "FPGA-based Embedded Visual Servoing Platform

for Quick Response Visual Servoing", 2011 8th Asian Control Conference (ASCC), 15-18 May, 2011, Kaohsiung, Taiwan.

[77] Honegger, D., Oleynikova, H., and Pollefeys, M., "Real-time and low latency embedded computer vision hardware based on a combination of FPGA and mobile CPU", 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sept 14-18, 2014, Chicago, IL, USA.

[78] Mur-Artal, R., Montiel, J. M. M., and Tardos, J. D., "ORB-SLAM: A Versatile and Accurate Monocular SLAM System", IEEE Transactions on Robotics, vol. 31, 2015, pp. 1147–1163.

[79] Priandana, K., Hazim, M., Wulandari, Kusumoputro, B., "Development of Autonomous UAV Quadcopters using Pixhawk Controller and Its Flight Data Acquisition", 2020 International Conference on Computer Science and Its Application in Agriculture (ICOSICA), Sept 16-17 2020, Bogor, Indonesia, Indonesia.

[80] Rushdi, M. S. A., Tennakoon, T. M. A., Perera, K. A. A., Wathsalya, A. M. H., and Munasinghe, S. R., "Development of a small-scale autonomous UAV for research and development", 2016 IEEE International Conference on Information and Automation for Sustainability (ICIAfS), Dec 16-19 2016, Galle, Sri Lanka.

[81] Ceron, A., Mondragon, I., and Prieto, F., "Onboard visual-based navigation system for power line following with UAV", International Journal of Advanced Robotic Systems, March 14 2018, 2–12.

[82] Marchand, E., Spindler, F., and Chaumette, F., "ViSP for visual servoing: a generic software platform with a wide class of robot control skills", IEEE Robotics and Automation Magazine, vol. 12, 2005, pp. 40–52.

[83] Al Habsi, S., Shehada, M., Abdoon, M., Mashood, A., and Noura, H, "Integration of a Vicon camera system for indoor flight of a Parrot AR Drone", 2015 10th International Symposium on Mechatronics and Its Applications (ISMA), Dec 8-10 2015, Sharjah, United Arab Emirates.

[84] Vidal, A. R., Rebecq, H., Horstschaefer, T., and Scaramuzza, D., "Ultimate SLAM? Combining Events, Images, and IMU for Robust Visual SLAM in HDR and High-Speed Scenarios", IEEE Robotics and Automation Letters, vol. 3, Jan 15 2018, pp. 994–1001.

[85] Dimitrova, R. S., Gehrig, M., Brescianini, D., and Scaramuzza, D., "Towards Low-Latency High-Bandwidth Control of Quadrotors using Event Cameras", 2020 IEEE International Conference on Robotics and Automation (ICRA), Sept 15 2020, Paris, France.

[86] Tiemann, J., Schweikowski, F., and Wietfeld, C., "Design of an UWB indoor-positioning system for UAV navigation in GNSS-denied environments",

2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN), Oct 13-16 2015, Banff, AB, Canada.

[87] Cisek, K., Zolich, A., Klausen, K., and Johansen, T. A., "Ultra-wide band Real time Location Systems: Practical implementation and UAV performance evaluation", 2017 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS) Oct 3-5 2017, Linkoping, Sweden.

[88] Gargioni, G.,Peterson, M., Persons, J. B., Schroeder, K., Black, J., "A Full Distributed Multipurpose Autonomous Flight System Using 3D Position Tracking and ROS", 2019 International Conference on Unmanned Aircraft Systems (ICUAS), June 11-14 2019, Atlanta, GA, USA, USA.

[89] Jung, S., Hwang, S., Shin, H., and Shim, D. H., "Perception, Guidance, and Navigation for Indoor Autonomous Drone Racing Using Deep Learning", IEEE Robotics and Automation Letters, vol. 3, July 2018, pp. 2539–2544.

[90] Roberge, V., and Tarbouchi, M., "Fast path planning for unmanned aerial vehicle using embedded GPU System", 2017 14th International Multi-Conference on Systems, Signals and Devices (SSD), March 28-31 2017, Marrakech, Morocco.

[91] Guowei Cai, Chen, B. M., Lee, T. H., and Miaobo Dong., "Design and implementation of a hardware-in-the-loop simulation system for small-scale UAV helicopters", 2008 IEEE International Conference on Automation and Logistics, Sep 1-3, 2008, Qingdao, China.

[92] Odelga, M., Stegagno, P., Bulthoff, H. H., and Ahmad, A., "A Setup for multi-UAV hardware-in-the-loop simulations", 2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS), Nov 23-25, 2015, Cancun, Mexico.

[93] Shah, S., Dey, D., Lovett, C., and Kapoor, A., "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles", Springer Proceedings in Advanced Robotics, 2017, pp. 621–635.

[94] Ratnesh Madaan, Nicholas Gyde, Sai Vemprala, Matthew Brown, Keiko Nagami, Tim Taubner, Eric Cristofalo, Davide Scaramuzza, Mac Schwager, Ashish Kapoor, "AirSim Drone Racing Lab", Proceedings of the NeurIPS 2019 Competition and Demonstration Track, vol. 123, 2020, pp. 177–191, Vancouver CANADA.

[95] Ma, C., Zhou, Y., and Li, Z., "A New Simulation Environment Based on Airsim, ROS, and PX4 for Quadcopter Aircrafts", 2020 6th International Conference on Control, Automation and Robotics (ICCAR), April 20-23, 2020, Singapore.

[96] Michael, N., Mellinger, D., Lindsey, Q., and Kumar, V., "The GRASP Multiple Micro-UAV Testbed" IEEE Robotics and Automation Magazine, vol. 17, 2010, pp 56–65.

[97] Rho, E., and Jo, S., "OctoMap-based semi-autonomous quadcopter navigation with biosignal classification", 2018 6th International Conference on Brain-Computer Interface (BCI), Jan 15-17, 2018, GangWon, South Korea.

[98] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W., "OctoMap: an efficient probabilistic 3D mapping framework based on octrees", Autonomous Robots, vol. 34, 2013, pp. 189–206.

[99] Mellinger, D., and Kumar, V., "Minimum snap trajectory generation and control for quadrotors", 2011 IEEE International Conference on Robotics and Automation, May 9-13, 2011, Shanghai, China.