# A Low-Power Architecture for High Frequency Sensor Acquisition in Many-DOF UAVs

Renato Mancuso*, Or D. Dantsker*, Marco Caccamo*, Michael S. Selig*,
*University of Illinois at Urbana-Champaign, USA, {rmancus2, dantske1, mcaccamo, m-selig}@illinois.edu

***Abstract*—Unmanned Aerial Vehicles (UAVs) are becoming increasingly popular thanks to an increase in the accessibility of components with high reliability and reduced cost, making them suitable for civil, military and research purposes. Vehicles classified as UAVs can have largely different properties in terms of physical design, size, power, capabilities, as well as associated production and operational cost. In this work, we target low-power, low-cost UAVs that feature a high number of degrees of freedom (DOF) and that are instrumented with a large number of sensors. For such platforms, we propose an architecture to perform data acquisition from on-board instrumentation at a frequency (100 Hz) that is twice as fast as existing products. Our architecture is capable of performing acquisition with strict timing constraints, thus, the produced data stream is suitable for performing real-time sensor fusion. Furthermore, our architecture can be implemented using embedded, commercial hardware, resulting in a low-cost solution. Finally, the resulting data acquisition unit features a low-power consumption, allowing to remain operative from two to three hours if powered with a miniature battery.**

## I. Introduction

Unmanned Aerial Vehicles (UAVs) are becoming increasingly popular thanks to an increase in the accessibility of components with high reliability and reduced cost. This includes: high-performance airframes; high energy-density batteries; efficient and lightweight motors; low-power and long-range radio frequency (RF) devices; high-performance, low-profile sensors, storage and processing units. Thanks to advances in technology and accessibility, UAVs are becoming more appealing for practical use in civil, military and research domains. For example, UAVs are used for surveillance, remote sensing and monitoring, as well as transport. The autopilot systems in UAVs represent the on-board logic that performs monitoring of state variables and produces actuation commands. Consequently, a robust and accurate autopilot system is indispensable for UAVs to successfully perform the assigned tasks.

In this work, we target UAVs that feature a high number of input degrees of freedom (DOF), (number of actuation inputs) and that are instrumented with a large number of sensors to have an accurate overall view of the system state. Specifically, we focus on small to mid-sized, low-cost (less than 100,000 USD) UAVs that are challenging for two main reasons: first, their reduced size does not imply a simplification in physical behavior of the model, nor the complexity of its actuation; second, particular attention is needed to produce an avionic infrastructure that is lightweight, features low power consumption, and requires a proportional budget to be set in place.

For such systems, actuators require a controller running at 50 Hz to maintain stability. The frequency requirement stems from the fact that commercial actuators found on small and mid-sized UAVs (e.g. servos, electronic speed controllers) are designed to operate at that frequency. Therefore, optimal and accurate control requires having a sensor sampling frequency of 100 Hz. For the aforementioned reason, in this paper, we focus on an architecture that performs data sampling from on-board instrumentation at the target frequency of 100 Hz. As detailed, the proposed solution has a set of desirable features that make it practically applicable to a wide-range of UAVs. In particular, we propose a data logging platform that:

1) Is able to perform data sampling from a complete set of on-board sensors at a target **frequency of 100 Hz**, resulting twice as fast as existing products;
2) Can be assembled using **low-cost commercial embedded devices**;
3) Relies on novel software solutions rather than hardware over-provisioning to meet the **real-time constraints** of the incoming data flow;
4) Thanks to the low-profile components involved, we propose a solution that features **low power consumption and weight**, making it suitable to be carried on small-sized UAVs and able to operate for hours with a small battery;
5) Being mostly software-defined, rather than hardware constrained, is largely **re-organizable, expandable and on-the-fly re-configurable**.

We perform an analysis of the proposed high-frequency sensor sampling architecture with respect to commercial data

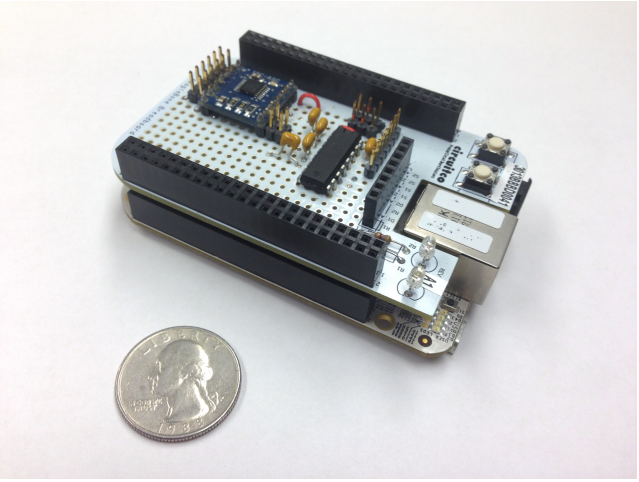


Fig. 1. Sensor data acquisition unit.

acquisition products for UAVs of comparable size and characteristics. We found that, to the best of our knowledge, our architecture is able to achieve the highest sampling frequency for a large number of data sources and that it can be assembled at the lowest cost. Specifically, our platform is equipped with a 6-DOF inertial measurement unit (IMU), GPS, altimeter, airspeed sensor, magnetometer, and a large number of analog and digital I/O channels with the ability to perform on-board storage and real-time transmission of flight data at 100 Hz.

The rest of the paper is organized as follows. Section II reviews the related work on modelling and sensor data acquisition solutions for UAVs. Next, in Section III we draw a comparison of our platform with existing commercial solutions. Section IV contains a detailed description of our architecture from both the hardware and software point of view. We perform an evaluation of our solution using real flight-data in Section V. Finally, the paper concludes in Section VI.

## II. Related Work

UAVs have largely been studied from both a theoretical perspective and from an applicative point of view. Theoretical work has been done in understanding the dynamics of multirotor vehicles, such as quadrotors [1, 2, 3], hexarotors [4, 5] and octorotors [6, 7]. Similarly, the problem of controlling variable scaled-size helicopters has been investigated in [8], while in [9] the problem of accounting for servo actuation delay is studied. Moreover, the problem of modeling [10, 11] and controlling [12] fixed-wing UAVs (airplanes) with different configurations have been investigated.

A common problem in deploying a control strategy to UAV flight control systems is having a detailed and accurate knowledge of the overall state of the system, which includes actuator position as well as the physical state of the aircraft. Such state variables have to be monitored on-line at a *fast enough* speed (as previously mentioned, a frequency of 50 Hz is used on commercial UAVs) to perform proper automatic control.

The problem of collecting state data has been approached in two main ways. A first solution involves instrumenting the flying site with motion-capture systems that are able to track the flying object and feed it live state data (position, velocity, orientation, proximity to other UAVs/objects). For instance, in [13] Kushleyev et al. use an infrastructure based on the Vicon motion capture system [14] to provide accurate measurements (sampling at 100 Hz) to a group of small quadrotors. Similar setups of motion capture system have been used in [**?** **?** ]. In [15, 16, 17], different architectures for supporting the development of indoor flight testbeds are proposed that aim at optimizing resource usage or providing fault-tolerance guarantees. The main disadvantages of indoor instrumentation are that: (1) the cost of the equipment can easily be orders of magnitude greater than the UAVs that are being controlled and (2) such an environment is suitable for UAVs that do not require large open spaces to be operated.

A second approach consists of embedding sensing devices into the UAVs, so that sensing (and control) is performed on-board. This implies that the targeted UAVs must carry enough payload for the avionics components. This solution is applied to UAVs []. The problem of retrieving a coherent view of on-board imprecise sensors has been extensively studied in [18] and Kalman Filters [19, 20] are used to correct drifts and measurement errors. In [21], a solution is proposed for a low-cost flying-wing operating at sampling rate of 50 Hz. Similar works have been proposed in [22, 23].

Flight data coming from on-board sensors can be used to perform live control or build a complete time history of the state of the monitored vehicle, to be analyzed offline. The ability to perform on-board sensor data acquisition and processing is a prerequisite for autonomous flight. Electronic devices that can aggregate data from sensors and perform on-board control take the name of autopilots [24, 25, 26]. However, for the sake of maintaining stability, having a full set of on-board sensor data is not strictly necessary for the entire duration of the flight.

Conversely, data recording solutions, which typically do not need to perform data processing, must achieve a large enough sampling frequency from all data sources in order to collect enough data to allow for accurate post-flight analysis. Accuracy in data collection, although not necessarily safety-critical, is fundamental for a meaningful reconstruction of flight conditions. Commercial data recording solutions for small UAVs are currently available [27, 28]. However, to the best of our knowledge, none of the proposed solutions are able to achieve a stable sampling frequency of 100 Hz while maintaining a small volume and weight factor, high energy efficiency, and the capability to record traces from a large number of sources for extended periods of flight, as discussed in Section III.

## III. Product Comparison

In order to properly evaluate our solution, existing commercial products that are able to perform sensor data acquisition were studied. In this section, we draw a comparison to highlight the performance gap between what is proposed in this work and the leading commercial products. The examined units are the Cloud Cap Piccolo II autopilot [25], the MicroPilot MP2128g autopilot [26], the RCAT Systems Industrial UAV data-logger [28] and the Eagle Tree Systems Flight Data Recorder Pro data-logger [27]. All of these units are intended to be used on small to mid-sized UAVs and thus are comparable to our system.

The comparison is summarized in table I, where the proposed solution is reported on the last column. The table includes information about the types of sensors supported by the unit under analysis, its sampling frequency, as well as the storage capabilities and estimated cost. As can be seen, our solution is able to collect data from the widest range of on-board sensors and at the highest frequency. The platform can be assembled using commercial components, resulting in a remarkably low deployment cost.

## IV. Architecture Description

In this section, we detail our implementation on a commercial embedded board and discuss the aspects of our design that required particular attention in order to achieve the desired high-end performance given the constrained hardware. The

| Product | Cloud Cap Piccolo II | MicroPilot MP2128g | RCAT Systems Industrial UAV | Eagle Tree Systems Flight Data Recorder Pro | Our sensor data acquisition unit |
|---|---|---|---|---|---|
| **Sensors** | | | | | |
| **Inertial sensors** | 3-axis, ±10 g accelerometer 3-axis, ±300 deg/s gyroscope | 3-axis, ±5 g accelerometer 3-axis gyroscope | 1-axis, ±8 g accelerometer | 2-axis, ±38 g accelerometer | 3-axis, ±18 g accelerometer 3-axis, ±300 deg/s gyroscope |
| **Magnetometers** | Add-on supported | Add-on supported | - | - | 3-axis ±750 mG and 3-axis ±11 G |
| **Altimeter (barometric)** | 1 ft resolution | 1 ft resolution | 8 ft resolution | 1 ft resolution | 1 ft resolution |
| **Airspeed (pitot probe)** | up to 180 mph | up to 300 mph | 10–290 mph | 9–350 mph | 5–180 mph |
| **GPS** | 4 Hz | 4 Hz | 1 Hz | 10 Hz | 120 Hz (IMU assisted) |
| **Digital I/O** | 16 | 8 | - | - | 20 |
| **Analog inputs** | 4x 10 bit | 32x 24 bit at 5 Hz | 2x | 2x | 7x 10 bit, 16x 12 bit, 1x 14 bit |
| **Other inputs** | CANbus | - | 2 Thermocouples, current and voltage measurement, optical RPM measurement | 2 Thermocouples, current and voltage measurement, optical RPM measurement, 4 CH PWM measurement | 8 CH PWM measurement, 1x serial port, CANbus |
| **Data Handling** | | | | | |
| **Sampling rate** | 20 Hz | 5–30 Hz | 20 Hz | 40 Hz | 100 Hz |
| **Local output** | LPT | Serial | - | - | Serial or Ethernet |
| **Storage** | - | 1.5 MB on-board | up to 512 MB SD | 10 kB on-board | up to 64 GB microSD |
| **RF link** | 25 mi | 3 mi | 15 mi | 14 mi | 25 mi |
| **Estimated cost** | $20,000+ | $6,000+ | $2,500+ | $650–1,500+ | $300 + $80 for IMU[1] |

TABLE I
PRODUCT COMPARISON

complete set of system schematics and source codes of the developed architecture are available upon request.

### A. Hardware Description

A system diagram depicting the main components of the hardware platform is shown in Figure 2. In the figure, the main blocks that compose the functional units are highlighted. For each of them, some details are provided about the device type as well as the communication interface. The specifications for all the components used in the sensor data acquisition system are given in Table II.

| | |
|---|---|
| **Processing unit** | BeagleBone running 32-bit Ubuntu Linux |
| **Sensors** | |
| **IMU** | XSens Mti-g 6-DOF IMU with Wi-Sys WS3910 GPS Antenna |
| **Airspeed probe** | EagleTree Systems pitot-static probe |
| **Airspeed sensor** | All Sensors 20cmH2O-D1-4V-MINI differential pressure sensor |
| **Magnetometer** | PNI Corp MicroMag 3 |
| **Analog-to-digital converters** | 2x Gravitech 12 bit - 8 Channel ADC |
| **Power** | |
| **Regulators** | Castle Creations CCBEC |
| **Batteries** | Thunder Power ProLite 2S 450 mAh |
| **Transceiver** | Digi 9X Tend 900-MHz card |
| **Data Storage** | 8GB microSD card |
| **Data Rate** | 100 Hz |

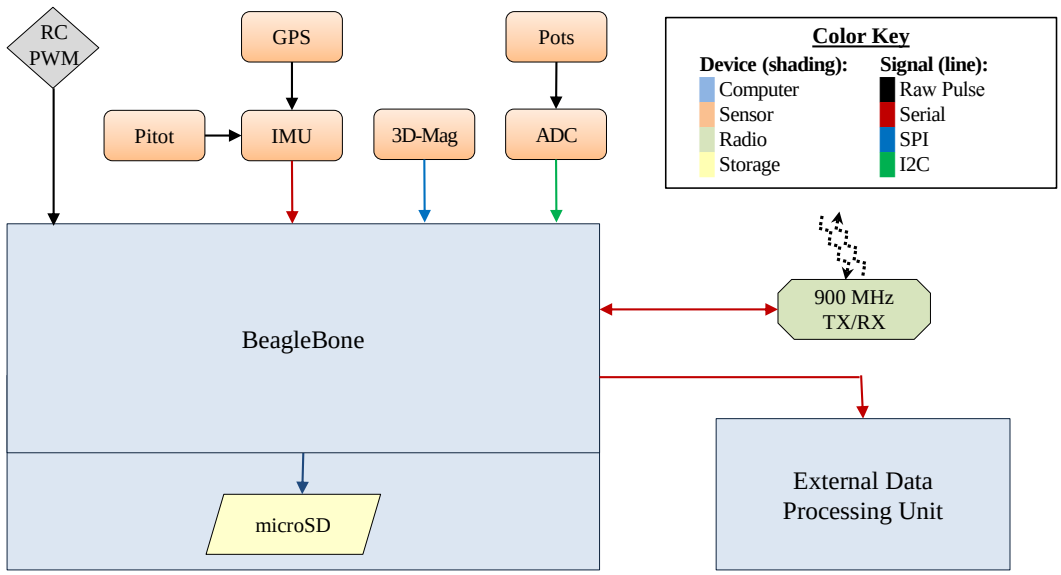
TABLE II
SENSOR DATA ACQUISITION UNIT SPECIFICATIONS

*1) Inertial Measurement Unit:* The (Inertial Measurement Unit) IMU unit depicted in the figure is equipped with a 3-axis gyroscope, a 3-axis accelerometer and a GPS receiver. This provides the unit a total of 6 degrees of freedom. A 2-channel, 14-bits analog to digital converter is also included in the same block, of which one channel is used for airspeed measurement. Finally, even though the IMU unit also includes a 3-axis magnetometer, a separate, external magnetometer unit, mounted on the tail of the aircraft, has been used to acquire measurements for the magnetic field, as detailed in Section IV-A2.

The reason for this design choice is that measurements taken with the built-in magnetometer are heavily influenced by the activity of the airplane motor. A way to reduce this effect consists in placing the magnetometer unit as far as possible from the motor. Thus, the ideal position for the IMU would be the tail of the model. However, the disadvantage of placing the IMU unit in the tail is twofold. First, the weight of the unit itself is not negligible and placing it in the tail would unbalance the overall weight distribution. Second and more important, the IMU is also providing measurement of acceleration, which, in order to be representative of the behavior of the whole body, needs to be taken as close as possible to the center of mass of the model. For the typical airplane, the center of mass is located between a 1/4 to 1/3 the distance between the leading and trailing edges of the wings, which is right behind the motor.

The unit has to be initialized and programmed with the sampling frequency and desired accuracy. The unit considered in our implementation is the XSens MTi-g [29]. It features a serial interface and is connected to one of the UART interfaces of the data collection unit, which is programmed to work as a RS-232 serial port. The IMU unit works in an asynchronous way: data are internally produced and exposed on the serial interface at the programmed sampling frequency. Thereby, a polling strategy has to be set in place in order to capture the transmitted sample when it is offered by the unit. If a transmitted frame is missed, the sample is lost. A 16-bit counter updated by the IMU itself and included in each frame

[1]This cost considers using a DIYDrones ArduIMU+ V3 which features the same sensors specifications as the XSens Mti-g IMU used in this implementation for availability reasons.




!t

Fig. 2. Architectural block diagram of the system

can be used to detect the loss of a sample.

*2) Magnetic Measurement Unit:* As mentioned earlier, the magnetometer included in the IMU unit cannot be used due to the strong interference in the magnetic field coming from the motor. For this reason, in our design, we included a separate stand-alone module to perform measurement of the magnetic field in order to compute the heading of the aircraft. The magnetometer module, for the reason explained above, is placed at the tail of the model. Specifically, the sensor used in our implementation is a MicroMag3 3-axis magnetometer, which can sample up to a frequency of 2 KHz, with a maximum resolution of $1.5^{10}$ $G^2$.

The MicroMag3 magnetometer is accessible trough a Serial Peripheral Interface (SPI). The communication protocol required by this component requires that a full sampling command is sent for each of the 3 axis. The command also includes the precision at which the returned sample should be acquired. However, higher sampling frequencies result in sharply decreasing precision.

*3) Analog to Digital Converters:* The design also includes an analog-to-digital converter (ADC) unit that can expose up to 16 12-bits channels (the actual configuration tested has 8 channels). The ADC channels can be used for a wide range of measurements that are significant in the setup of a non-trivial UAV testbed like the one considered in this paper. Specifically, ADC channels can be used to collect data from potentiometers, airspeed sensors, temperature, voltage and current sensors. In the current implementation, 8 ADC channels have been wired and acquired, but, keeping the same hardware layout, up to 16 channels can be obtained. The communication with the ADC module is performed over a $I^2C$ bus.

Potentiometers are components that are able to measure the value of a rotation angle by outputting a representation of the angle with an analog value of voltage. Thus, the accuracy of the measurement only depends on the quantization bits used in the digital representation. In an airplane model, potentiometers can be employed to monitor the actual status of each control surface (flaps, ailerons, elevators, rudder). For an accurate measurement of all the relevant variables involved in the state of the considered testbed, such values are in fact necessary. This is because control surfaces are controlled using servomotors, which set their rotation angle according to a pulse-width modulated (PWM) signal. A common misconception is that knowing the PWM signal input to the servo is enough to know its rotation angle. This is not true for three main reasons. First, commercial servos have a number of steps that is in the range 512 to 2048, thus, in low-cost configurations, no more than 512 values of angles can be determined. Second, a failing servo cannot be detected without having a feedback from its physical status. Third, a servomotor applies a certain amount of torque to a control surface, but the resulting position of the surface is also determined by opposite force of airstream pushing on the surface in flight conditions[3]. Note that the latter varies together with the flight conditions (flight speed, orientation, wind speed and so on), as well as the angle of the surface itself.

ADC channels can be used to acquire data from additional airspeed sensors, which can provide accurate measurement in case of slow flight, windy conditions and landing/take-off at a higher rate than GPS. Thermocouples sensors, which are useful to monitor the temperature of testbed's internal components (such as motor, batteries, avionic instrumentation), are measured trough ADC channels as well. Finally, measurements from voltage/current sensors can be acquired in the same way and are useful to monitor both power consumption and activity of the components in both the avionic instrumentation and the propulsion subsystem.

[2]G stands for Gauss which is a unit of measurement for the intensity of a magnetic field. One G is equal to 100 $\mu T$ (micro-tesla).

[3]This effect is known as blow-back.

*4) Pulse Width Modulation Inputs:* As previously mentioned, servos actuating the control surfaces as well as the motor are commanded using PWM signals. Thereby, typically, the outputs of the radio receiver are PWM signals, as well as the actuation commands coming from the autopilot. Being able to record such signals permits a full reconstruction of the history of the actuation outputs sent to the UAV, allowing, for instance, to study the behavioral response of the model to a given sequence of commands.

In the PWM signal, the information is carried by a square wave and encoded as the time between a rising edge and a falling edge. Typical PWM signals for flight control components operate at a frequency of about 50 Hz and the time difference between the rising and falling edges varies from 0.8 to 2 ms. Even though the PWM channels are coming from a single device, which is either a radio control receiver or an autopilot, the generated PWM waves cannot be assumed to be in phase.

Although the PWM module is depicted as a separated component, in our design the PWM demodulation is performed directly on the data collection unit. For this purpose, any programmable micro-controller could be used, but the benefit of having the demodulation done on a dedicated module is counterbalanced by the need to add additional wiring, thereby increasing the weight and power consumption. Instead, we demonstrate that such task can be performed directly on the monitoring unit without introducing excessive overhead as to compromise the collection of data coming from the aforementioned units. As a result, 8 PWM channels are connected to a correspondent number of GPIO pins on which the sampling is performed.

*5) Data Collection Unit:* The data recorded at the individual functional blocks described above are collected for acquisition in a single aggregation point. The hardware component used as the data collection unit is a commercial embedded board equipped with a breadboard on which the IMU, magnetometer and ADC units are located on or wired to.

Specifically, we have used a BeagleBone [30] board. This board featured a Cortex-A8 ARM CPU operating at a frequency of 720 MHz. The DRAM size is 256 MB, while the boot scripts as well as the root filesystem is stored on a microSD card. This resource-constrained setup has the obvious advantage to have an extremely limited power consumption. This is highly desirable for a flying testbed since it directly allows to reduce the weight of the carried batteries. According to our measurements on the power consumption of the board in a fully operational state, the data collection unit absorbs about 0.45 W. This means that a small 500 mA battery can operate it for a time window of two to three hours. Conversely, commercial data recording solutions have a consumption of about 1 W, with an accuracy that is 50% lower.

The choice of this embedded platform is also driven by the need to have a set of interconnected devices that communicate over different protocols and interfaces, such as SPI, $I^2C$, GPIO and UART. Moreover, since different sets of measurements could be relevant over time, future extendability has to be taken into account. In particular, our setup can be easily extended including 16 additional ADC channels. This platform also allows for wired and wireless data-link connections.

Considering an output format of simple plain data (filtering and compression can be performed at a later stage) containing the output of the described units, we have evaluated a data generation rate of about 25 KB/s. Indicatively, it means that each 15 minutes of flight leads to the production of about 50 MB of data[4]. Thereby, the other advantage of having a fully customizable solution for data recording is that the system can be easily configured to monitor the incoming data without recording and enable/disable recording upon the recognition of a provided set of events. Possible events can be the GPS position being within a provided distance from a specified waypoint; the altitude and/or the speed being above a given threshold; the position of a configured switch on the transmitter and in general anything that triggers a change of state in the monitored values.

### *B. Software Layer*

In order for the data acquisition unit to minimize the data loss and to keep the desired sampling and recording rate given the constrained hardware, a careful, real-time oriented design of the software layer which handles communication protocols and sampling intervals has to be performed. Moreover, side effects of the software layer design choices must be taken into account. For instance, among different designs that are able to achieve the desired sampling rate, the one that minimizes CPU usage should be chosen because processor utilization directly reflects into power consumption, which, as already discussed, is a crucial parameter to be taken into account.

In this section we describe our design, which represents a good compromise between modularity, sampling speed and power consumption.

*1) Overall Logic:* We have discussed in the previous section how Figure 2 describes the hardware organization in our platform. Indicatively, each module of the system represents an area of concern characterized by an interface, a certain amount of data bandwidth and sampling speed. Intuitively, the same separation should be reflected in the design of the software layer. The advantage of this is twofold: (1) a modular design allows long-term maintainability of the layer, which is a crucial issue since the same platform should be easily adaptable to different UAV models and sensor configurations; (2) in order to allow fully on-the-fly reconfigurable platform, it is necessary to design the system in a way that units can be individually powered off (for example in case of detected failure), paused or brought back on-line (for instance upon detection of an event).

On the other hand, the requirement for modularity implies that control flows among different units need to be separated. In a traditional Linux system like the one employed in our platform, the way such separation can be achieved is trough the definition of different executable units. Thus, a perfect modularity would imply defining a different process per each functional unit. However, due to the complete address-space separation of traditional Linux processes, we have evaluated that the time and CPU resources involved in context switches

[4]Most sensors require a calibration to be performed on the ground, which lasts for about 15 minutes.

have an heavy and negative impact on the timing properties of the single units, not allowing globally a sampling frequency higher than about 50 Hz.

To draw a trade-off between modularity and optimization in the usage of CPU resources, the design of our software layer employs lightweight threads, each handling the communication and the data collection from an assigned functional unit.

The overall logic, thus can be described as follows: at system start-up, a customized initialization procedure is performed per each unit. No data acquisition is started before all the functional units are reported as on-line. The main operations performed at initialization time are:

1) Allocation of memory buffers for incoming data and initialization of the final output file;
2) Initialization procedure for UART, $I^2C$ and SPI interfaces. Parameter setup in also performed on the connected devices;
3) Memory-mapping of hardware devices (such as timers and GPIO registers) that are used bypassing the OS interface;
4) Creation of worker threads that will be responsible for each functional unit.

Once the initialization has been completed and the data collection is started, each worker thread is responsible for (A) periodically interrogating the assigned functional unit and (B) place collected data in an assigned buffer. The periodicity of each thread reflects the sampling speed of the assigned unit and, in a scenario with perfect time synchrony between threads and peripherals, it would be enough to let each thread perform the next data collection at the expected arrival time of the next data frame. Unfortunately, this assumption does not hold considered the low-cost nature of the employed sensors. Thus, a strategy to handle data skew, eliminating the presence of stale data becomes necessary.

We have engineered each thread to reduce its processing latency and to always release the CPU in case data are not available at the time of wakeup. This is necessary because on a single core scenario with a constrained clock speed, each uncontrolled busy period can easily introduce unacceptable latency in a different component of the system, leading to data frame loss. Often, the communication interface has been reimplemented in user-space and simplified to reduce the produced latency and hardware resources are directly accessed to avoid the slowdown resulting from frequent system calls. A detailed description of the different solutions adopted in order to achieve the desired performance per each functional unit is provided in the following sections.

Finally, a separated thread is dedicated to collect sampled data from the local buffers associated to each functional unit and produce an unique, timed view of the aggregated output. Since this is the only thread with a direct notion of wall-clock time, it is also responsible for stale data elimination.

*2) Timing:* When working with physical devices in which the notion of time is important to characterize both functional behavior (e.g. timing of the communication channel) and interpretation of data, fast access to time samples is crucial.

The Linux kernel makes precise time samples, suitable for real-time purposes, available through the `gettimeofday` and `clock_gettime` system calls. However, the fundamental problem is that a system call is required whenever a new sample is needed. Since our objective is sampling all the channels 100 times per second, the system call approach cannot scale due to the significant overhead of performing a single call.

For this reason, the approach followed system-wide in our design is to assign a hardware timer to the data acquisition process. Specifically, the board used as a data collection unit provides a set of 5 32-bit wide timers that can be used for general purpose and that are clocked at 32 KHz (i.e. with a maximum resolution of 31.25 $\mu$s). In our implementation, we select one of the unused timers at initialization phase and configure it to work in auto-reload mode with the maximum resolution. The registers containing the incremented timer value can be accessed by mapping a portion of physical memory (`/dev/mem`) in user-space. In this way, a time sample can be acquired by any worker thread by performing a single memory operation. This eliminates the need (and connected overhead) for system calls to acquire time samples. Note that reading from the hardware timer is enough to calculate the length of time intervals, but it does not provide any notion of absolute (wall-clock) time.

*3) IMU Handling Thread:* The IMU unit provides sensor readings through a serial interface. Once initialized, the IMU starts the sampling process internally, so to offer a new sample of all the embedded sensors every 10 ms (100 Hz). In order to prevent frame loss, the data collection unit has to poll the device when data may be ready to be transferred. If the collection unit does not perform polling with an appropriate timing, the sample in the IMU queue gets overwritten with new measurements and the frame is dropped.

In first approximation, an event-prediction based approach can be used by (1) considering the time-stamp of the captured sample, (2) accounting for the transfer and processing time and (3) calculating the arrival time of the next event. However, there are two reasons why the approach as it is leads to poor performance and an high frame drop rate. First, waking up the thread responsible for sampling the IMU at exactly the same frequency is not always possible: another thread could be performing computation (or waking up to process events) at the same time. Second, due to timing skew, the IMU may have data ready to be transferred slightly ahead of time or past the nominal arrival time.

For the explained reasons, we followed a *reverse-exponential backoff* approach. Specifically, the wake-up time is calculated in the way a pure event-based approach requires, but the thread is activated slightly in advance (about $1/2$ ms), in line with the observed skewness of the IMU unit. If no data are ready at that time, it means that a sample is going to be available at the serial interface within about 1 ms. Thus, we divide the possible slack time in 2 halves and put the thread to sleep for the resulting amount of time (in the first step, $1/2$ ms). If no sample is captured after this additional amount of time, the procedure is repeated reducing the sleep-time in half. Hence the reverse-exponential backoff.

An optimization is used here: if the calculated sleep time

due to the backoff procedure falls below the resolution of the used kernel, the call performed to release the CPU for the desired amount of time is converted to a `sched_yield`. In this way, we introduce no explicit sleep time, making the polling loop faster but without keeping the CPU busy.

*4) MAG Handling Thread:* The additional magnetometer installed on the tail of the model is interfaced using an SPI interface. The board used as a data collection unit provides direct support for SPI. However, for our evaluation, we found that the overall stack provided to perform SPI communication introduced a too high and unnecessary timing overhead.

As previously mentioned, the communication protocol for the particular magnetometer unit used in our design requires that an explicit sampling command is sent to the unit to request a sample. This has to be done independently for each of the three axis and with the desired speed (100 MHz). For this reason and observing that the timing overhead of the standard SPI kernel interface is not negligible, we have reimplemented the SPI communication interface to work entirely in user-space.

Specifically, through the platform's internal MUX switches, the SPI pins are routed to GPIO pins, whose bank's physical address can be mapped inside the process space at initialization time. Thus, basic SPI read/write operations can be reimplemented performing simple memory operations instead of read/write operations over file descriptors.

Reimplementing the SPI interface in user-space has two main advantages. First, a consistent number of system calls can be avoided, reducing the overall communication latency and CPU usage. Second, the inter-bit time of the SPI protocol can be tuned to exactly match the specific device in use. Using this set of optimizations, we were able to achieve a 2 KHz maximum sampling speed. Unfortunately, due to hardware constraints, the device itself produces progressively inaccurate samples as the requested sampling frequency increases. A good compromise for this device has been observed to be at a sampling frequency of 50 Hz. Thus, in the final design, such operating frequency has been selected, even though we were able to consistently acquire data form the considered device (together with the output of the other units) at 100 Hz.

For the magnetometer, the sampling strategy is fairly simple: the handling thread is activated with a 50 Hz frequency and the command to read a sample is sent. After the command is issued, the device asserts a ready signal when data are ready to be received from the master. As such, a polling mechanism is required to conditionally wait for data arrival. To prevent the thread from keeping the CPU busy during the polling cycle, the `sched_yield` system call is used.

*5) ADC Handling Thread:* The main device used to acquire measurements from analog devices to be converted in digital samples (ADC conversion) is connected to the data collection unit through an $I^2C$ interface. In this case, the observed overhead introduced by the provided $I^2C$ abstraction provided by the kernel has been found to be acceptable for the desired sampling frequency (100 Hz). This is mainly because the device allows to be programmed with the number of channels that have to be sampled and returns a unique block of data containing all the measurements.

Consequently, at every wake-up time, a write operation is performed (to request the samples of a given set of channels), followed by a read operation. The acquired block of data is simply split into separate locations of the buffer associated with the working thread.

*6) PWM Handling Thread:* The PWM demodulation is the most computational intense operation performed by the platform and, intuitively, also the most time-sensitive. The signal coming from the PWM channels is a set of square waves in an arbitrary phase in which the value carried by the signal is represented in terms of timing between a rising edge of the wave and the corresponding falling edge. Although the employed hardware includes PWM modulators in hardware, no support is provided for demodulation. Thereby, it must be done in software.

In UAV models, PWM signals are those coming from the transmitter and as a feedback from the sensor that monitors the revolving speed of the motor. In such signals, the distance between a rising edge and a falling edge varies within a range of 0.8 and 2 ms. It means that having a latency of just 100 $\mu$s in the edge detection can introduce an error higher than 10% for the resulting demodulation.

A first solution would be mapping the corresponding GPIO registers in which the PWM channels are connected and continuously sampling the observed values calculating the timing between two detected edges of the signal. Since multiple GPIO pins can be read with a single memory operation, in our evaluations, we have experienced that this leads to a perfect PWM demodulation. The obvious drawback is that this solution keeps the CPU busy and thus: (1) makes the other threads suffer an unacceptable latency; (2) whenever the spinning thread that continuously monitors the PWM channels is descheduled, an edge in the signal can be missed, leading to an incorrect signal demodulation.

The adopted solution involved setting up the GPIO lines as sources of interrupts at both rising and falling edge. In this way, other threads can be scheduled between any edge, while higher priority is given to the interrupt handler whenever a change in any of the PWM signals occurs. The Linux kernel provides a user-space interface to GPIO-triggered interrupts through the virtual `sys` filesystem. Individual GPIO lines are exported as file descriptors and `poll` system calls can be used to detect interrupt arrival without spinning the CPU. However, we have evaluated that the observed latency of this solution is still unacceptable, mainly because of the large number of system calls involved (more than one per ms).

For this reason, an custom kernel module has been designed to attach fast and simple interrupt handlers to each observed GPIO pin. Whenever an interrupt occurs, the kernel handlers calculate the timing of the signal and perform the demodulation, writing the resulting value in a memory page. The physical address of the memory page used as an output buffer is exported to the user-space using the `proc` virtual filesystem. Finally, the user-space thread responsible for PWM data collection performs a mapping of the exported physical address and fetches the values of the demodulated signal performing simple memory operations. In this way, no system call is involved in PWM demodulation and the achieved result has the maximum precision permitted by the latency involved

in the interrupt handling of a standard Linux kernel (see Section V).

*7) Dumper Thread:* Finally, all the samples accumulated in the buffers of each one of the worker threads, are collected, aggregated and transferred to disk by a dumper thread. The dumper thread is activated exactly at a time frequency of 100 Hz.

Each time the dumper reads into the buffer of a given worker thread, it also resets the buffer. This is useful to make sure that the final data do not contain stale samples, i.e. samples containing an old value of the data that has not been updated by the corresponding worker thread. This can happen, for example, if the underlying sampled device is effectively operating at a frequency lower than 100 Hz.

Since the dumper thread performs both read and write operations on the buffers, data races have to be taken into account. On the other hand, due to the enforced timing on the threads, the probability of having data races is limited. To exploit the latter property while still ensuring data correctness, we have replaced standard pthread mutex (`pthread_mutex`) with fast user-space mutex (futex). This allows the threads to perform write operations on the buffers in a serialized way (preserving data correctness) while still avoiding expensive system calls whenever only one entity at a time is accessing the buffer.

Futex constructs are used between user-space sampling processes and dumper thread. One exception, however, is the PWM sampling process. In this case, serialization on a given block of memory has to be achieved between a user-space thread and a set of interrupt handlers. To the best of our knowledge, no primitive exists to simply achieve this type of synchronization. For this reason, the user-thread, upon acquisition of a sample, uses atomic swap instructions (`SWP`) to reset the buffer shared with the kernel-space PWM interrupt handlers. Note that, on uni-processor platforms, ensuring that the reset operation is performed with exactly one instruction would be enough to ensure correctness. However, using the `SWP` instruction makes the overall logic easy to port on a multi-core platform.

## V. Evaluation

A fixed-wing trainer-type radio control aircraft was built for testing the sensor data acquisition unit. The aircraft built was a Great Planes Avistar Elite [31], which has a 62.5 in wingspan and weighs 7 lbs. It is powered by a AXI 4120/14 600 W motor[32], a Castle Creation Phoenix ICE 75 Amp electronic speed controller[33], and a Thunder Power 14.8 V, 5 AHr lithium polymer battery[34]; and is controlled using Futaba servos and a 2.4 GHz spread spectrum receiver[35]. The aircraft has the following control surfaces: 2 ailerons (roll), 2 flaps, 1 elevator (pitch), and 1 rudder (yaw). The completed flight-ready aircraft is shown in Figure 3.

The sensor data acquisition unit and sensors were installed into the aircraft. In order, the following installations were made: the IMU was hard-mounted to the floor of the fuselage (body of the aircraft) at the center of gravity location; the sensor data acquisition unit was installed in the center of the fuselage, near and above the IMU; the magnetometer was installed in the tail of the aircraft; the pitot probe was mounted on the left wingtip, with tubing connecting its pressure taps to the differential pressure sensor, which was mounted in the left wing panel. Some modifications were done to the aircraft structure to allow these installations. The center section of the fuselage is shown in Figure 4.

The completed, instrumented aircraft was taken to Eli Field in Monticello, IL for flight testing. The aircraft was flown completely manually so the data logging unit could be tested. The data logging unit was started and the aircraft was left to sit for 15 min to allow the sensors to calibrate. The aircraft was then throttled up, took off, was flown through a simple traffic pattern, and landed, totaling 98 seconds from throttle-up for takeoff to full stop after landing. The aircraft's flight path is shown in Figure 5 and the data recorded is shown in Figure 6.

In the first 7 seconds of the recording, the aircraft remained stationary on the runway, which allowed for steady-state measurements to be taken. The plots in Figure 6 show that there was little to no change in the measurements coming from all of the sensors during this time period. Therefore, it can be assumed that there is minimal interference being induced between any of the subsystems.

Then, at 7 seconds, the aircraft was throttled-up for takeoff and at 105 seconds it completely stopped after landing. During this time period, the sensors tell the story of the flight.

In Figure 6 (a), the position of the aircraft can be seen with the start position assigned the location (0,0,0)[m]; it should be noted that the end location is not the same as the start location because the rolled to a stop 80 m South-East from the start position. The time history of aircraft location matches the flight path plot.

Figure 6 (b) shows the attitude of the aircraft, were $\phi$ is the roll angle, $\theta$ is the pitch angle, and $\psi$ is the heading. The time history of the attitude shows when the aircraft pitching up to takeoff, down to loose altitude before landing, up to flare right before touch-down, and while maneuvering. The changes in heading correlate with the turns in the traffic pattern seen in (a). It is important to note that when there is a vertical line in the plot, where the heading changes from -180 deg to 180 deg, the airplane is turning smoothly from heading South-West to South-East, through South, which is represented as both $-180$ deg and 180 deg. The roll angles visible in the plot correspond to the roll required for the banked turn and therefore an increase in roll occurs as the turn is initiated and a decrease when the turn is ended.

Figures 6 (c) and (d) show the linear accelerations and rotation rates experienced by the aircraft. The noise seen in these figures are cause by motor vibration and can be removed by using a low pass filter. The noise stops between 66 seconds and 88 seconds when the motor is turned off before landing in order to lose speed and altitude. There are a few spikes in IMU Z-axis measurements between 91 and 93 seconds that correlate to the aircraft bumping on the runway several times during landing. Since the system records at 100 Hz, we are able to tell that the aircraft had 2 large bumps with about 1 second between them followed by a smaller third bump 0.5 seconds later, with the airplane finally rolling down the runway

Fig. 3. Completed flight-ready Avistar aircraft.

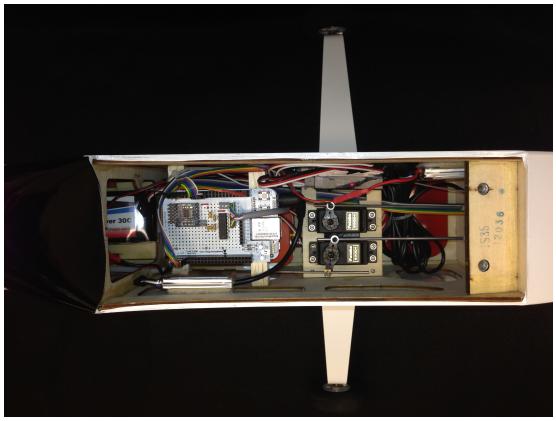


Fig. 4. Photo of the center section of the fuselage, with the nose pointed to the left.

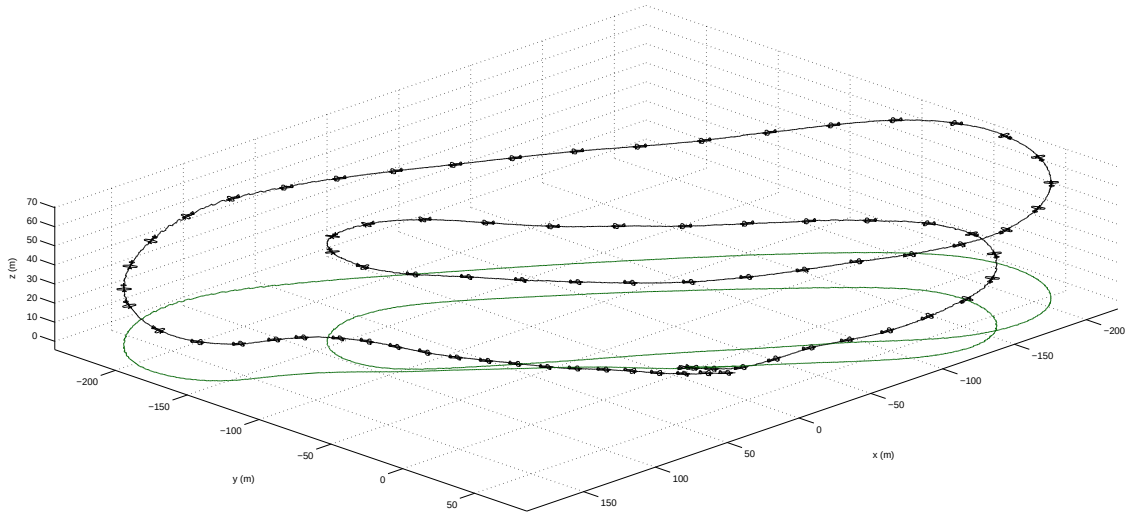



Fig. 5. Flight path. The aircraft is drawn three times bigger than the actual size and once every second tangent to the flight path.

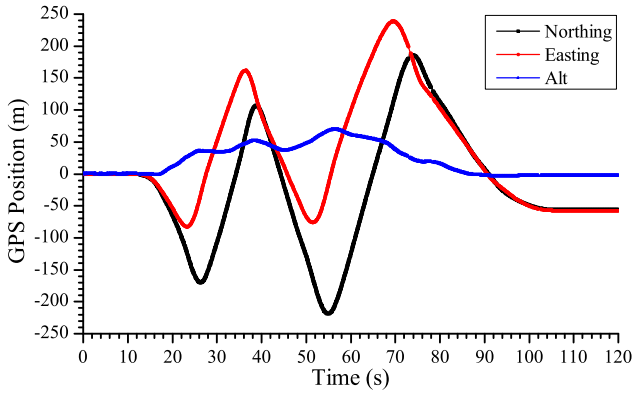
after that; this was confirmed with ground video taken of the flight.

The horizontal and vertical ground speeds of the aircraft are shown in Figure 6 (e) and in Figure 6 (f), the total ground speed and airspeed are plotted. The offset between these speeds are easily accounted for by factoring in the wind. The little bit of wind during the flight changed direction to the aircraft as the aircraft changed heading.

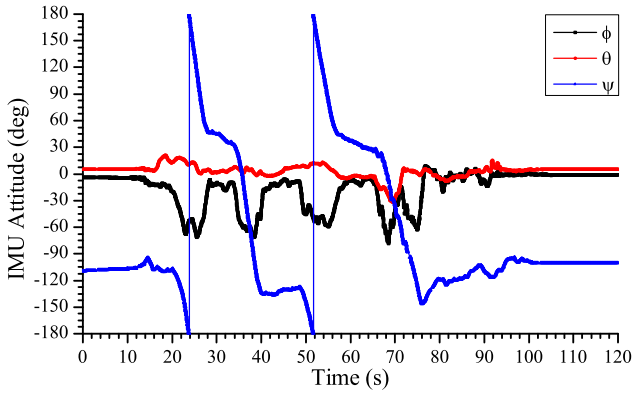
Figure 6 (g) provides a time history of the control inputs given by the pilot. The control inputs for all of the maneuvers described above can be be seen.

Finally, Figure 6 (h) shows the magnetic field strength as recorded by the IMU and the stand-alone magnetometer. There is a larger offset in magnetic field strength between the magnetometer and IMU when the motor is off as opposed to on. The stand-alone magnetometer, which is located in the tail, receives a smaller fraction of the magnetic field induced by the propulsion system than the IMU receives. This difference is magnified when the motor is off.

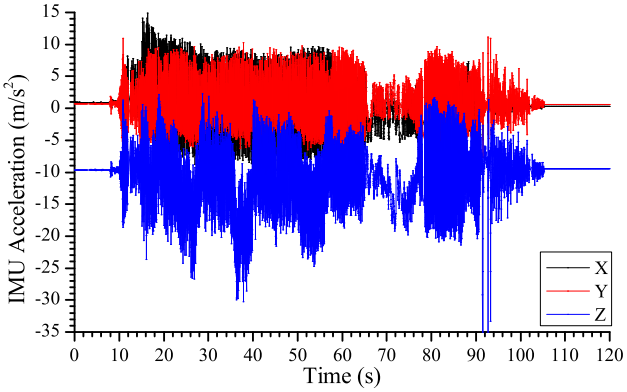
The sensor data acquisition unit provided continuous high-frequency flight state data for the entirety of the flight. There were no faults evident.



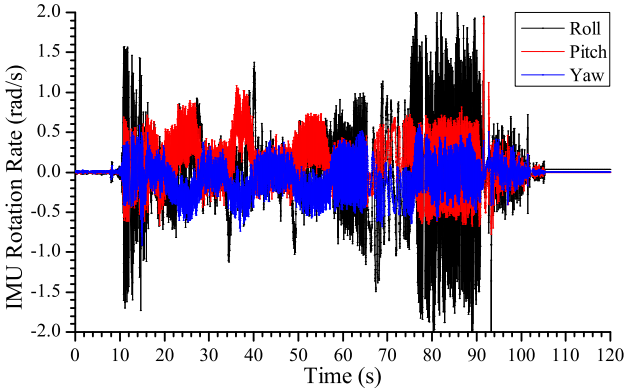

(a) Aircraft position



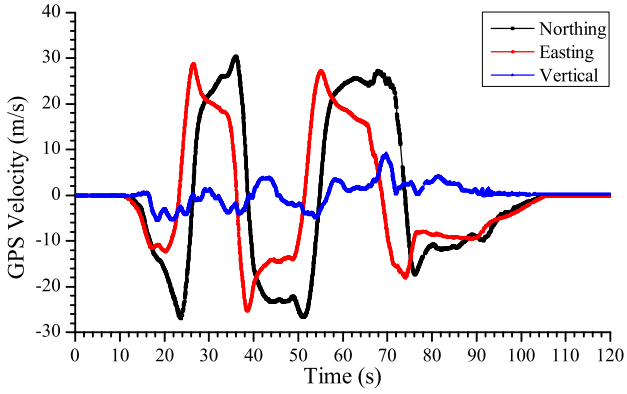

(b) Aircraft attitude



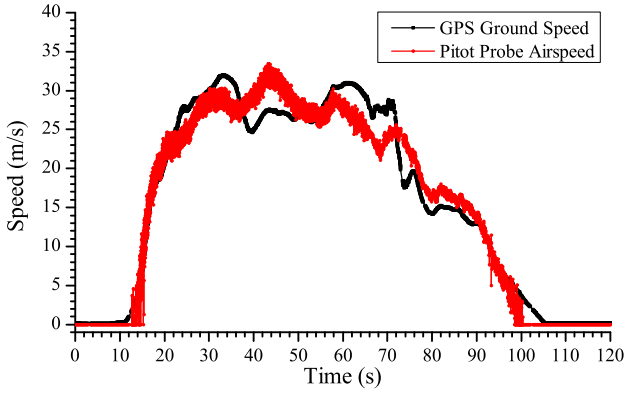

(c) Aircraft body frame accelerations



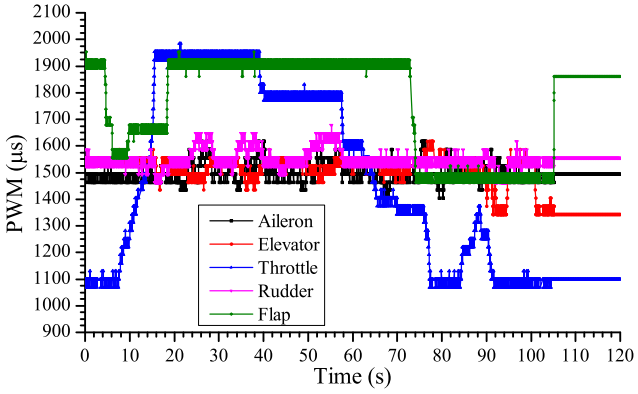

(d) Aircraft body frame rotation rates



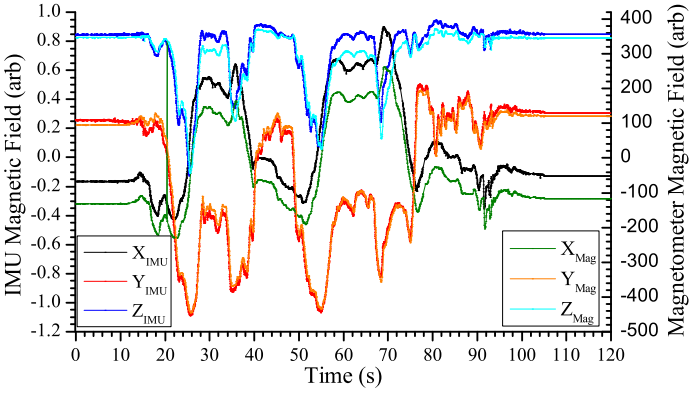

(e) Aircraft GPS velocity




(f) Aircraft speed




(g) Aircraft control surface command inputs




(h) Magnetic field strength in aircraft body frame

Fig. 6. Sensor measurements

## VI. Conclusion

Tracking the overall state of a UAV in flight conditions requires collecting data from different sources: the sensors of the on-board instrumentation. Furthermore, we have discussed that a desirable sampling speed is 100 Hz. In fact, achieving such a data acquisition frequency, not only allows performing a precise stability and control actuation at 50 Hz, but also permits an accurate reconstruction of the aircraft conditions both live and in a post-flight phase.

Existing commercial sensor data acquisition solutions allow the aggregation of a limited number of sources, achieving a sampling speed which is sensibly lower than 100 Hz and featuring a relatively high cost. In this work, we propose an architecture and an implementation for a sensor data acquisition unit that is suitable for small and mid-sized UAVs. Thanks to its ability to perform fast aggregation of data originated at a large number of sources, it is particularly useful to be equipped on models featuring a considerable number of degrees of freedom. Furthermore, the proposed unit is entirely composed of commercial, easily accessible embedded components, overall featuring (A) low production costs, (B) low power consumption and (C) reduced size/weight factor.

We have implemented the proposed architecture and instrumented a real testbed (an airplane) to perform the evaluation of our solution. In this way, we were able to asses the correctness of our implementation, as well as the achieved performances in terms of accuracy of the collected data. As a part of our future work, we plan to feed an autopilot with the data-stream produced by the presented architecture, in order to investigate: complete sensor data fusion; faulty-sensor data correction using measurements from different/redundant units; collaborative flight of UAVs exchanging locally computed state variables.

Finally, we have made the complete set of schematics, as well as the source code of the described platform, available upon request.

## References


[1] R. Mahony, V. Kumar, and P. Corke. Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *Robotics Automation Magazine, IEEE*, PP(99):1, 2012.

[2] L. Taeyoung, M. Leoky, and N. H. McClamroch. Geometric tracking control of a quadrotor UAV on se(3). In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 5420–5425, 2010.

[3] P. Pounds, R. Mahony, and P. Corke. Modelling and control of a large quadrotor robot. *Control Engineering Practice*, 18(7):691–699, February 2010.

[4] S. Omari, M.-D. Hua, G. Ducard, and T. Hamel. Hardware and software architecture for nonlinear control of multirotor helicopters. *Mechatronics, IEEE/ASME Transactions on*, PP(99):1–13, 2013.

[5] G. Ducard and M.-D. Hua. Discussion and practical aspects on control allocation for a multi-rotor helicopter. In *UAV-g 2011, Conference on Unmanned Aerial Vehicle in Geomatics*, pages 0001–0006, 2011.

[6] V. G. Adir, A. M. Stoica, and J. F. Whidborne. Modelling, stabilization and single motor failure recovery of a 4Y octorotor. In *Proc. 13th IASTED International Conference on Intelligent Systems and Control (ISC 2011)*, pages 82–87, Cambridge, U.K., July 2011.

[7] V. G. Adir, A. M. Stoica, and J. F. Whidborne. Modelling and control of a star-shaped octorotor. In *Proc. 2nd International Conference on Mechanical Engineering, Robotics and Aerospace (ICMERA 2011)*, pages 195–199, Bucarest, Romania, October 2011.

[8] R. Mahony and T. Hamel. Robust trajectory tracking for a scale model autonomous helicopter. *International Journal of Robust and Nonlinear Control*, 14(12):1035–1059, 2004.

[9] H. Teimoori, H. R. Pota, M. Garratt, and M. K. Samal. Planar trajectory tracking controller for a small-sized helicopter considering servos and delay constraints. In *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, pages 681–686, 2011.

[10] Michael S. Selig. Modeling Full-Envelope Aerodynamics of Small UAVs in Realtime. AIAA Paper 2010–7635, August 2010.

[11] Michael S. Selig. Modeling Propeller Aerodynamics and Slipstream Effects on Small UAVs in Realtime. AIAA Paper 2010–7938, August 2010.

[12] L. R. Ribeiro and N. M. F. Oliveira. UAV autopilot controllers test platform using matlab/simulink and x-plane. In *Frontiers in Education Conference (FIE), 2010 IEEE*, pages S2H–1–S2H–6, 2010.

[13] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar. Towards a swarm of agile micro quadrotors. *Autonomous Robots*, 35(4):287–300, 2013.

[14] Vicon Motion Systems, Inc. http://www.vicon.com, Accessed Oct. 2013.

[15] S. Lange and P. Protzel. Cost-efficient mono-camera tracking system for a multirotor UAV aimed for hardware-in-the-loop experiments. In *Systems, Signals and Devices (SSD), 2012 9th International Multi-Conference on*, pages 1–6, 2012.

[16] M. Valenti, B. Bethke, D. Dale, A. Frank, J. McGrew, S. Ahrens, J.P. How, and J. Vian. The MIT indoor multi-vehicle flight testbed. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2758–2759, 2007.

[17] H. Oh, D. Y. Won, S. S. Huh, D. H. Shim, M. J. Tahk, and A. Tsourdos. Indoor UAV control using multi-camera visual feedback. *J. Intell. Robotics Syst.*, 61(1-4):57–84, January 2011. ISSN 0921-0296.

[18] P. D. Groves. *Principles of GNSS, inertial, and multi-sensor integrated navigation systems*. GNSS technology and applications series. Artech House, 2008. ISBN 9781580532556.

[19] R. E. Kalman. A new approach to linear filtering and prediction problems. 1960.

[20] Victor Manuel Becerra, P. D. Roberts, and G. W. Griffiths. Applying the extended kalman filter to systems described by nonlinear differential-algebraic equations. *Control Engineering Practice*, 9(3):267–281, 2001.

[21] L. Di, T. Fromm, and Y. Chen. A data fusion system for attitude estimation of low-cost miniature UAVs. *J. Intell. Robotics Syst.*, 65 (1-4):621–635, January 2012.

[22] A. M. Jensen, Y. Q. Chen, M. McKee, T. Hardy, and S. L. Barfuss. AggieAir - a low-cost autonomous multispectral remote sensing platform: New developments and applications. In *Geoscience and Remote Sensing Symposium,2009 IEEE International,IGARSS 2009*, volume 4, pages IV–995–IV–998, 2009.

[23] D. Kingston, R. Beard, A. Beard, T. McLain, M. Larsen, and W. Ren. Autonomous vehicle technologies for small fixed wing UAVs. In *AIAA Journal of Aerospace Computing, Information, and Communication*, pages 2003–6559, 2003.

[24] H. Y. Chao, Y. C. Cao, and Y. Q. Chen. Autopilots for small unmanned aerial vehicles: A survey. *International Journal of Control, Automation and Systems*, 8(1):36–44, 2010.

[25] Cloud Cap Technology. Cloud cap technology – piccolo ii highly integrated uas autopilot. https://www.cloudcaptech.com/piccolo_II.shtm, Accessed Oct. 2013.

[26] MicroPilot. Micropilot - products - mp2128g. http://www.micropilot.com/products-mp2128g.htm, Accessed Oct. 2013.

[27] Eagle Tree Systems, LLC. Eagle tree r/c telemetry. http://www.eagletreesystems.com/, Accessed Oct. 2013.

[28] RCAT Systems. Rcat systems - UAV & unmanned vehicle products, UAV telemetry system, UAV electronics, pitot probes, alpha beta probes. http://rcatsystems.com/uav.php, Accessed Oct. 2013.

[29] Xsens Technologies B.V. Xsens, mti-g. http://www.xsens.com/en/general/mti-g/, Accessed Nov. 2012.

[30] BeagleBoard.org Foundation. Beagleboard.org - beaglebone. http://beagleboard.org/Products/BeagleBone, Accessed Oct. 2013.

[31] Hobbico, Inc. Great planes avistar elite .46 advanced trainer rtf. http://www.greatplanes.com/airplanes/gpma1605.html, Accessed Oct. 2013.

[32] Model motors s.r.o. Axi 4120/14 gold line. http://www.modelmotors.cz/index.php?page=61&product=4120&serie=14&line=GOLD, Accessed Oct. 2013.

[33] Castle Creations, Inc. http://castlecreations.com/products/products_fly.html. http://castlecreations.com/products/products_fly.html, Accessed Oct.

2013.
[34] Advanced Energy Tech. Thunder power rc. http://thunderpowerrc.com/, Accessed Nov. 2012.
[35] Hobbico, Inc. Futaba radio control systems and accessories. http://futaba-rc.com/, Accessed Oct. 2013.