HAL
open science

Adaptive Closed-loop Speed Control of BLDC Motors with Applications to Multi-rotor Aerial Vehicles

Antonio Franchi, Anthony Mallet

To cite this version:

Antonio Franchi, Anthony Mallet. Adaptive Closed-loop Speed Control of BLDC Motors with Applications to Multi-rotor Aerial Vehicles. IEEE International Conference on Robotics and Automation, May 2017, Singapour, Singapore. hal-01476812

HAL Id: hal-01476812

https://laas.hal.science/hal-01476812v1

Submitted on 2 Mar 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Preprint version, final version at http://ieeexplore.ieee.org/ 2017 IEEE Int. Conf. on Robotics & Automation, Singapore

Adaptive Closed-loop Speed Control of BLDC Motors with Applications to Multi-rotor Aerial Vehicles

Antonio Franchi and Anthony Mallet

***Abstract*— This paper introduces the adaptive bias and adaptive gain (ABAG) algorithm for closed-loop electronic speed control (ESC) of the brushless direct current (BLDC) motors typically used to spin the propellers in multi-rotor aerial robots. The ABAG algorithm is adaptive and robust in the sense that it does not require the knowledge of any mechanical/electrical parameter of the motor/propeller group and that neither a pre-calibration nor the knowledge of the feedforward/nominal input is needed. The ABAG algorithm is amenable to an extremely low complexity implementation. We experimentally prove that it can run in 27.5 µs on a 8 MHz microcontroller with no floating point unit and limited arithmetic capabilities allowing only 8-bit additions, subtractions and multiplications. Besides the controller implementation we present a self-contained open source software architecture that handles the entire speed control process, including clock synchronization, and over-current and blockage safeties. The excellent performance and robustness of ABAG are shown by experimental tests and aerial physical interaction experiments.**

I. Introduction

Multi-rotor aerial vehicles represent at date a fundamental asset of aerial robotics. They rose to a prominent role with respect to other aerial platforms mainly because of their mechanical/electrical resilience and relative simplicity, their capability of hovering, and the usability indoor and close to natural or artificial structures. Research in the stabilization of such platforms went very far allowing, e.g., the tracking of impressive acrobatic maneuvers. Nowadays off-the-shelf solutions are available on the market that relief roboticists of the burden of controlling canonical aerial platforms.

However, there are still several open points in the control of aerial robots. Two prominent examples are the *physically interactive aerial robots* and *omnidirectional aerial platforms*. As explained in the following, these new fields of research call for a more accurate and faster control of the rotor speed, which is directly related to the wrench (i.e., thrust force and drag moment) produced by the propeller.

In the case of physically interactive aerial robots, such as cable-connected [1] or tool-operating [2] aerial robots, and aerial manipulators [3], precise and quick control of the propeller speed would allow to effectively implement interaction control algorithms, such as, e.g., impedance control [4] and interaction-force observers [5]. Ultimately, a precise control of the propeller speeds is at the basis of a *full-body* torque control (i.e., base included) of an aerial manipulator.

The propellers of omnidirectional aerial platforms, such as the tilted-propeller hexarotor [6], [7] or the omnidirectional octorotor [8], are all fixed in different orientations. The direction of the total force then depends on the speed of each propeller. The accurate control of each speed is of fundamental importance to precisely orient the total force. This issue did not arise in underactuated multi-rotors with collinear propellers, where the direction of the total force is constant in body frame regardless of the propeller speeds.

Also in the case of common multi-rotors that do not belong to the previous two categories a fast and precise control of the propeller speed is highly beneficial, since it allows to, e.g., *i)* set higher gains for the attitude control thus increasing the reactivity of the whole platform in fast maneuvers, and *ii)* precisely estimate and compensate external disturbances.

In this paper we focus on *electronic speed controllers* (ESC) for *brushless direct current* (BLDC) motors, which are the mostly used in multi-rotor robots.

The majority of ESC softwares at date provide the possibility to adjust the speed in open loop by setting the PWM duty cycle. A look-up table can be used to map the duty cycle to the speed. However this method is clearly not fast, imprecise and non-robust (due to its open loop nature), and requires a non-negligible pre-calibration effort. To solve this problem some ESC softwares, like the SimonK, BLHeli, and Autoquad ESC32 implement a real closed-loop speed control, typically a proportional integral (PI) plus feedforward (FF). This method allows to reach a good control of the speed but at the price, again, of a pre-calibration phase for the FF term that has to be performed for each motor/propeller pair. The 'I' term provides also some level of robustness which is however kept limited due to the need of avoiding excessive overshoot and wind-up problems.

In this paper we present an alternative algorithm for controlling the propeller speed. Differently from the others *i)* it does not require any pre-calibration phase, *ii)* it is extremely robust and applicable to a wide set of motor/propellers without the need of gain tuning, *iii)* can achieve performances that are independent of the terminal voltage supplied by the battery, the mechanical wearing, the temperature, and so on, *iv)* it is amenable to an extremely low complexity implementation even when compared with a 'supposedly simple' PI+feedforward controller.

Furthermore, we present an open source implementation of the algorithm (available at https://git.openrobots.org/projects/tk3-mikrokopter) in a self contained software architecture that provides also additional features such as, e.g., clock synchronization, mechanical and electrical safeties, and speed reversibility. Then we provide an experimental campaign that validates the great adaptiveness and robustness of the algorithm, able to achieve very good performances with no

[1]LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France. antonio.franchi@laas.fr, anthony.mallet@laas.fr

Work funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 644271 AEROARMS

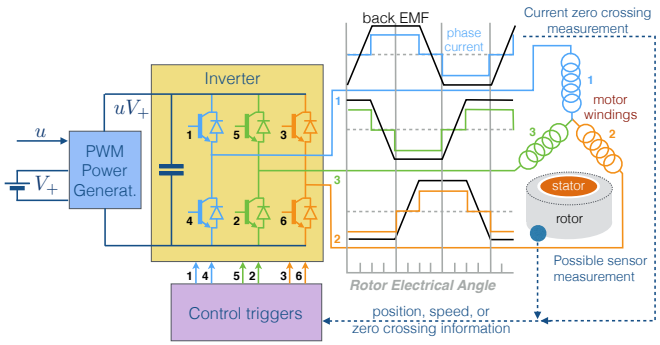

Fig. 1: A simplified scheme of a typical BLDC motor and BC circuit. Different wirings and and number of windings can be used depending on the model, without affecting the working principles.

specific gain tuning despite the change of motor and propeller types, and other external disturbances. Finally we show the major benefit of employing the proposed propeller speed control in physically interactive aerial robotic tasks.

An alternative approach to force control is the one based on aerodynamical power [9] rather than the propeller speed. The possibly better force tracking requires however more complex hardware and long-lasting calibration.

This paper is organized as it follows. The model and problem setting is given in Sec. II. Section III presents the algorithm and Sec. IV shows its implementation in a self-contained software architecture. An extensive experimental validation is given in Sec.V and Sec.VI concludes the paper discussing future perspectives.

II. Modeling and Problem Setting

We consider a standard BLDC motor connected to an ESC unit (see Fig. 1 for a high-level simplified scheme of a typical system). The motor is composed of a static part (stator) that includes the electromagnetic coils (or windings) and a rotating part (rotor) that carries no electric parts but only permanent magnets. In order to be precise, in the following we shall call *rotor* the rotating part of the motor and we denote the rotating wing fixed to the the rotor only with the word *propeller* (or just *load* if we want to stay general).

The ESC unit contains an inverter (see Fig. 1) whose output provides the sequential current commutation on the coils, which in turns generates the rotating magnetic field that starts and sustains the motion of the rotor. Commutations are triggered by a rotor position feedback that comes either from an additional sensor (e.g., an encoder) or, in case of *sensorless* BLDC, from the detection of the instants of zero crossing of the voltage generated in the unpowered windings.

In the implementation and tests (Secs. IV and V) of the speed control algorithm proposed in Sec. III we shall use the latter approach, which is more common in aerial robotics applications, since they do not require to drive a propeller at very low speeds (for which the sensorless approach is not viable). Furthemore, the sensorless approach allows to obtain a smaller weight, reduced hardware complexity, and lower costs. Nevertheless, the speed control algorithm proposed in Sec. III can be seamlessly used also for sensorized motors.

The equivalent voltage applied to the inverter of the BC is $u\,V_+$, where $u \in [0,1]$ is the *duty cycle* of the actual pulse width modulated (PWM) voltage signal, and $V_+ \in \mathbb{R}^+$ is the power supply voltage (e.g., the terminal voltage of a battery).

Thanks to the very low inductance design of typical BLDC motors[1] we can typically neglect the fast current dynamics within the sampling frequency of the control loop. However, in order to remain general and to account also for the errors due to this approximation, we consider a generic model for the dynamics of x represented by a standard nonlinear differential equation

$$\dot{x} = \bar{f}(x,t) + \bar{h}(x,V_+,t)u, \tag{1}$$

where the rotor *frequency* (or speed) of rotation (naturally expressed in Hz), and $\bar{f}$, and $\bar{h}$ are unknown nonlinear functions of x and the time t. The dependency on t in (1) makes the system very general because it incorporates the presence of both unknown and approximated parameters and dynamics. The function $\bar{h}$ plays, roughly speaking, the role of the inverse of the inertia 'seen' by the input u. The function $\bar{f}$ represents instead all the other possibly nonlinear dynamical effects (including the external moment applied to the rotor). Without loss of generality we assume $\bar{h}(x,V_+,t) = \bar{h}(x,t) > 0$ for any x and t in the domain of interest. In sensorless BLDC motors the most directly available measure is the rotor *period* of rotation $y = 1/x$, we can then write the period dynamics from (1) as

$$\dot{y} = -y^2\bar{f}(1/y,t) - y^2\bar{h}(1/y,t)u = f(y,t) - h(y,t)u. \tag{2}$$

Since $\bar{h}(x,t) > 0$ we also have that $h(y,t) > 0\ \forall y > 0$ and $\forall t$.

The addressed control problem is then the following.

Problem 1 (*Robust Control of the propeller period of rotation*). Assume that it is possible to set u, and let be given the desired and measured motor periods of rotation: denoted with y^d and y, respectively. Design a feedback control law $u(y,y^d)$, that steers y to y^d and satisfies the following requirements:

- [*Complexity*] The algorithm has extremely low complexity, i.e., it must be possible to implement it with simple arithmetic operations (additions, subtractions, shift, comparisons and simple multiplications) at low resolution (e.g., 8 bits).
- [*Adaptiveness and Robustness*] The functions $f(\cdot)$ and $h(\cdot)$ in (2) are plausible for a real BLDC motor used in aerial robotics. However their exact structure is not known, a part for the fact that $h(\cdot) > 0$.

Since the algorithm is, eventually, implemented in discrete time, we shall denote with $\star_k$ the discrete-time counterpart of a variable $\star$, where $k \in \mathbb{Z}$ denotes the time step index.

III. Algorithm Description and Discussion

In this section we describe a control law for the input u that practically solves Problem 1.

A. The ABAG Algorithm

We named our algorithm the *Adaptive–Bias/Adaptive–Gain* (ABAG) Algorithm. A pseudocode description is given in Algorithm 1. The algorithm basic loop is composed by four simple steps:

[1]For example, the inductance of the Roxxy2827-35, one of the motors used in the experiments of Sec. V, is 2×10^{-4} H.

Algorithm 1: *ABAG Propeller Speed Control Algorithm*

Inputs	: $y_k \in \mathbb{R}^+$	%Measured rotation period, [s]
	$y_k^d \in \mathbb{R}^+$	%Desired rotation period, [s]
Output	: $u_k \in [0,1]$	%PWM duty cycle, [adim.]
Parameters	: $\alpha \in (0,1)$	%error sign filtering factor, [adim.]
	$\bar{e}_b \in (0,1)$	%threshold for bias adaptation, [s]
	$\delta_b \in (0,1)$	%bias adaptation step, [adim.]
	$\bar{e}_g \in (0,1)$	%threshold for gain adaptation, [s]
	$\delta_g \in (0,1)$	% gain adaptation step, [adim.]
Variables	: $\bar{e}_k \in [-1,1]$	% low-pass filtered error sign, [adim.]
	$b_k \in [0,1]$	% adaptive bias, [adim.]
	$g_k \in [0,1]$	% adaptive gain, [adim.]

1 $k = 0,\ u_0 = \bar{e}_0 = g_0 = b_0 = 0$
2 **while** $++k$ **do**
3 $\quad \bar{e}_k = \alpha \bar{e}_{k-1} + (1-\alpha)\mathrm{sgn}(y_k - y_k^d)$ %error sign filtering
4 $\quad b_k = \mathrm{sat}_{[0,1]}\left(b_{k-1} + \delta_b \,\mathrm{hside}(|\bar{e}_k| - \bar{e}_b)\,\mathrm{sgn}(\bar{e}_k - \bar{e}_b)\right)$ %bias update
5 $\quad g_k = \mathrm{sat}_{[0,1]}\left(g_{k-1} + \delta_g \,\mathrm{sgn}(|\bar{e}_k| - \bar{e}_g)\right)$ %gain update
6 $\quad u_k = \mathrm{sat}_{[0,1]}\left(b_k + g_k \,\mathrm{sgn}(y_k - y_k^d)\right)$ %control input computation

1) Error sign low pass filtering (Line 3): this step updates the variable $\bar{e}_k \in [-1,1]$, called the *low-passed error sign*, which is the output of a low pass filter applied to the sign of $(y_k - y_k^d)$. The parameter $\alpha \in (0,1)$ is the *filtering factor*: the closer α is to 1 the more the error sign is low-pass filtered.

2) Adaptive bias update (Line 4): this step updates the variable $b_k \in [0,1]$, which is called the *adaptive bias*. The update is done in the following way: if $\bar{e}_k > \bar{e}_b \in (0,1)$ (a parameter called the *bias adaptation threshold*) then b_k is incremented by $\delta_b \in (0,1)$ (a parameter called the *bias adaptation step*); if $\bar{e}_k < -\bar{e}_b$ then b_k is decremented by δ_b; otherwise b_k is left unchanged. In any case b_k is kept limited in the set $[0,1]$. This step is compactly written in Line 4 where 'hside' denotes the Heaviside (or unit step) function.

3) Adaptive gain update (Line 5): this step updates the variable $g_k \in [0,1]$, called the *adaptive gain*. The update law is inspired by [10] and is done in the following way: if $|\bar{e}_k| > \bar{e}_g \in (0,1)$ (a parameter called the *gain adaptation threshold*) then g_k is incremented by $\delta_g \in (0,1)$ (a parameter called the *gain adaptation step*); otherwise $\bar{g}_k$ is decremented by δ_g; In any case $\bar{e}_k$ is kept limited in the set $[0,1]$.

4) Control input computation (Line 6): this step computes the algorithm output u_k (i.e., the control input of (2)) on the basis of the current values of the adaptive bias b_k, adaptive gain g_k and the sign of $y_k - y_k^d$. In particular, if $y_k > y_k^d$ then $u_k = b_k + g_k$, if $y_k < y_k^d$ then $u_k = b_k - g_k$, otherwise $u_k = b_k$. In any case u_k is kept limited in the set $[0,1]$.

B. Rationale of the ABAG Algorithm

The rationale behind the proposed algorithm stems from basic discontinuous control analysis. First of all it is important to highlight the role of the signal $\bar{e}_k$ (Line 3 of Algorithm 1) and of the associated decision maps[2]. Developing the recursion we obtain $\bar{e}_k \simeq (1-\alpha)\sum_{l=k-m}^{k} \alpha^l e_{k-l}$, where

[2]In this context a decision map is a function $\xi : [0,1] \to \{-1,0,1\}$ which 'decides' whether a variable has to be decreased, kept constant, or increased depending on whether its value is -1, 0, or 1, respectively.

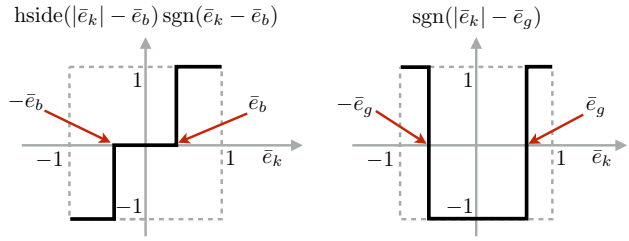

Fig. 2: Decision maps for the two adaptation steps. Left: ξ_b decision map for the bias: b_k is increased if $\bar{e}_k < -\bar{e}_b$, kept constant if $-\bar{e}_b < \bar{e}_k < \bar{e}_k >$, and decreased if $\bar{e}_k > \bar{e}_b$. Right: ξ_g decision map for the gain: g_k is increased if $|\bar{e}_k| > \bar{e}_g$ and decreased otherwise.

$e_k = y_k - y_k^d$, and m is the 'time window' after which the infinite sum terms are indistinguishable from the quantization error. Clearly, the larger α the larger m.

When the tracking quality is good, i.e., $e_k \simeq 0$, the number of times in which $e_{k-l} > 0$ and in which $e_{k-l} < 0$, for $l = 0,\ldots,m$, are close, thus generating $\bar{e}_k \simeq 0$ because of its low-pass nature. If instead the tracking is not good, there will be a large predominance of positive (or negative) values which will bring $\bar{e}_k$ in the neighborhood of 1 (or -1). By suitably thresholding the continuous function $\bar{e}_k$ it is possible to turn $\bar{e}_k$ into a decision map tailored on the specific needs.

The ABAG algorithm uses two distinct thresholding. The first generates the following decision map (see Fig. 2-left)

$$\xi_b = \mathrm{hside}(|\bar{e}_k| - \bar{e}_b)\mathrm{sgn}(\bar{e}_k - \bar{e}_b), \tag{3}$$

which is used in Line 4 of Algorithm 1. If in last m samples e_k has mainly been positive then $\xi_b = 1$, if the error has been isotropically oscillating around 0 then $\xi_b = 0$, and if e_k has mainly been negative then $\xi_b = -1$. The second thresholding generates another decision map (see Fig. 2-right)

$$\xi_g = \mathrm{sgn}(|\bar{e}_k| - \bar{e}_g), \tag{4}$$

which is used in Line 5 of Algorithm 1. The decision map ξ_g returns 1 if the tracking is not acceptable (e_k is not oscillating around 0 in the last m samples) and -1 otherwise.

The use of the two decision maps in the gain and bias adaptations laws is justified in the following. Writing Line 6 in continuous time (see previous remark) we have $u = b + g\,\mathrm{sgn}(e)$. Using (2) the dynamics of e is then

$$\dot{e} = \dot{y} - \dot{y}^d = \underbrace{f(y,t) - h(y,t)b - \dot{y}^d}_{l(y,t,\dot{y}^d)} - h(y,t)\,g\,\mathrm{sgn}(e). \tag{5}$$

Let us consider $\frac{1}{2}e^2$ as the candidate Lyapunov function, whose time derivative along the closed loop trajectories is

$$\tfrac{d}{dt}\left(\tfrac{1}{2}e^2\right) = \dot{e}e = l(y,t,\dot{y}^d)e - h(y,t)g|e|. \tag{6}$$

Recalling that $h(y,t) > 0$, the rhs of (6) is made negative if

$$g > l(y,t,\dot{y}^d)/h(y,t). \tag{7}$$

If, e.g., g is set to a large enough constant, robust stability would be guaranteed. However if g is chosen too large then an undesirable chattering phenomenon might appear when the error is practically zero, due to large jumps in the control command. Even though generating chattering for a very

Operation	Instructions	Clock cycles	Time at 8 MHz
32bits integer division	≥ 500	≥ 564	$\geq 70.5\,\mu s$
16bits integer division	≥ 153	≥ 190	$\geq 23.75\,\mu s$
32bits integer multiplication	40	74	$9.25\,\mu s$
16bits integer multiplication	7	10	$1.25\,\mu s$
14 poles motor at 150 Hz			$158\,\mu s$ period
ABAG algo in Table II	≤ 140	≤ 220	$\leq 27.5\,\mu s$

TABLE I: Complexity of costly operations on 8bit AVR microcontrollers compared to the required period of a motor controller spinning a 14 poles motor at 150 Hz and to the implementation of the whole ABAG algorithm in Table II.

short moment is acceptable, an everlasting chattering must be avoided to preserve the system mechanical integrity.

The idea behind the adaptation of g is to increase g only if the tracking quality degrades and to decrease it otherwise. Then it appears natural to let the decision map ξ_g decide the direction of the adaptation of g, while the step of the adaptation is set by another parameter δ_g. This brings naturally to the Line 5 of Algorithm 1.

Even if the gain adaptation keeps g as small as possible at each time t, it cannot decrease it below the value of the rhs of (7) at t. Therefore it is important also to decrease in parallel that value, to ensure an effective chattering suppression by the gain adaptation. This need is at the basis of the adaptation of the bias b. In fact the better b approximates $(f(y,t)-\dot{y}^d)/h(y,t)$ the smaller the rhs of (7). Similarly to a kind of gradient descent search, the adaptation of b is then driven by the decision map ξ_b which points towards the direction needed to decrease the error taking also into account the negativity of $-h(y,t)$ in (2). The step of the search is set by the parameter δ_b.

IV. Software Implementation and Complexity

A software implementation of the ABAG algorithm has been developed for the MikroKopter hardware. In particular, we used the BL-Ctrl-2.0 motor controller for which the electronic schematics are available[3]. This ESC uses an ATmega168A microcontroller running at 8 MHz. The I^2C bus of the microcontroller is connected to another central microcontroller, called the *flight controller* (not described here), that communicates with all motors as well as with the outside world. An RS232 line is also available for direct control of individual motors.

The ATmega168A has no floating point unit and has limited arithmetic capabilities, allowing only 8bits additions, subtractions and multiplications. Other operations must be performed by expensive software routines. Table I shows a few examples of such common arithmetic operations. In order to obtain good performance in the proposed software, all such operations have been avoided completely.

A. Overview, Interface, and Speed Measurement

The BL-Ctrl-2.0 hardware allows the microcontroller to control the six MOSFETs of the phase bridge through six independent GPIO. A hardware generated PWM signal can be applied to each of the high side transistors, while the lower side transistor are only switched on or off. The analog

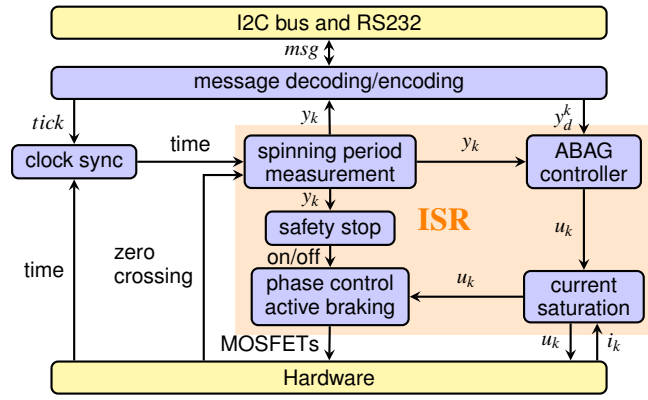

Fig. 3: ESC software architecture overview.

comparator unit of the microcontroller is hardwired to allow direct detection of the zero crossing of any phase. The associated interrupt service routine (ISR) is used to precisely trigger the software driving the phase bridge. Figure 3 shows an overview of the control software architecture. The individual blocks are described in the next subsections.

1) Interface and Protocol: Communication with the ESC is done through a message based protocol[4], either on the I^2C bus or on the RS232 line. To save bandwidth, the protocol is not designed to be robust to message corruption: the RS232 line (that may sometimes corrupt data) is used only for testing. During normal operation, the reliable I^2C bus is used. However, messages are marked with start and stop delimiters to allow proper synchronization if one peer has to drop part of a message because of a communication buffer overflow or any other unexpected reason.

The two communication channels can achieve up to 500 kbit/s baudrate. However, the I^2C bandwidth is shared among all motors and the bus is half duplex, so the available bandwidth for a motor in a typical quadrotor setup is about 60 kbit/s (e.g., 600 Hz for 10 bytes messages). In order to guarantee that during normal operation the desired spinning velocity for each motor is sent at the desired frequency (typically 1 kHz), the flight controller is configured as a bus master and always has the sending priority. It polls the motors for any data they have to send only when the bus is free. Also, in order to guarantee that all motors receive their desired spinning velocity in a synchronized way, such messages are broadcasted on the bus as an array of velocities in a single message. All motors receive the array at the same time, and pick up the velocity corresponding to their own id.

2) Speed Measurement and Clock Synchronization: The main difficulty was to get a precise time measure on the microcontroller that has no quartz oscillator. To this aim, we carefully implemented a clock synchronization algorithm.

The spinning period is measured as the time elapsed between two phase switches, i.e., $1/6^{\text{th}}$ of an electrical rotation (or $1/42^{\text{th}}$ of a physical rotation with a 14 poles motor). The measurement noise is filtered with an exponential moving average. Practically, a weighting factor of $1/4$ for new measurements leads to a good compromise between

[3] http://wiki.mikrokopter.de/en/BL-Ctrl_2.0

[4] https://git.openrobots.org/projects/tk3-mikrokopter/gollum/communication

```
1  int16_t u; /* output ∈[0;1023] */
2  int16_t bias, gain; /* bias and gain ∈[0;1023] */
3  int32_t e_bar; /* avg err sign, 16:16 fixed point */
4
5  void abag_ctrl(uint16_t y, uint16_t y_d) {
6    /* error sign low pass filtering */
7    if (y > y_d) e_bar = (3 * e_bar + 65536)/4;
8    else         e_bar = (3 * e_bar - 65536)/4;
9    /* bias adaptation and saturation */
10   if (e_bar > 49152 && bias < 1023) bias += 1;
11   if (e_bar < -49152 && bias > 1)   bias -= 1;
12   /* gain adaptation */
13   if (e_bar < -32768 || e_bar > 32768) {
14     if (gain < u/2) gain += 2;
15   } else {
16     gain -= 2; if (gain < 1) gain = 1;
17   }
18   /* command computation and saturation */
19   if (y > y_d) {
20     u = bias + gain; if (u > 1023) u = 1023;
21   } else {
22     u = bias - gain; if (u < 0) u = 0;
23   }
24 }
```

TABLE II: Implementation of ABAG control algorithm in C language. The longest code path is about 140 instructions that require about 220 clock cycles, i.e., 27.5 µs at 8 MHz. The parameters used are given in Table III.

Parameter	Value in the code	Parameter	Value in the code
$\bar{e}_g \in (0,1)$	$(\frac{32768}{65536} = 0.5)$	$\delta_g \in (0,1)$	$(\frac{2}{1024} = 0.001953)$
$\bar{e}_b \in (0,1)$	$(\frac{49152}{65536} = 0.75)$	$\delta_b \in (0,1)$	$(\frac{1}{1024} = 0.000976)$
$\alpha \in (0,1)$	$(\frac{3}{4} = 0.75)$		

TABLE III: Values of the parameters used in the experiments.

noise filtering capabilities and filtering delay. The period is measured with 1 µs resolution and it can be experimentally observed that the standard deviation of the measurement noise is about 2 µs (corresponding to ±0.6 Hz at 50 Hz or ±2 Hz at 90 Hz with a 14 poles motor).

B. ABAG Algorithm Implementation and Complexity

For maximum efficiency, the ABAG algorithm is implemented using only additions, comparisons and shifts. The actual C code is shown in table II. The longest code path runs in about 27.5 µs at 8 MHz (note that all other code paths are about the same length). This relatively light complexity allows to run the controller directly from the ISR routine that measures the current spinning velocity and controls the phase bridge. This is an important property since the controller, by its nature, provides a very discontinuous PWM duty cycle input u, the faster it runs the less chattering will actually be visible from the motor coils. This also guarantees that the controller runs each time a velocity measurement is available. Typically, for a 14 poles motor spinning from 20 Hz to 100 Hz this means that the controller runs at frequencies ranging from 840 Hz to 4.2 kHz.

In order to obtain $\bar{e}_k$ (called `e_bar` in the code), 16:16 fixed-point arithmetic is used. With this representation, the ±65 536 numbers in the code (line 7 and 8) represent ±1, i.e., the current error sign. Note that α is hardcoded in order to let the compiler optimize this computation efficiently as a simple shift and additions. Using a parameter that could be changed online would require a real 32 bits division.

The gain amplitude is limited to half of the commanded duty cycle (line 14). This limitation is useful only at low duty cycles (i.e., low spinning frequency), and was required

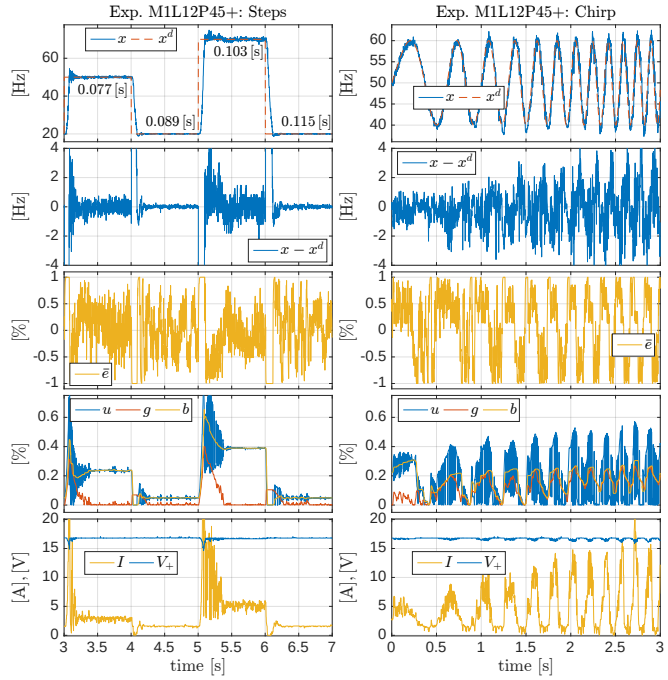

Fig. 4: A detail of the step and chirp responses for one particular experiment. The time instants in the top left plot indicate the rise/descend times of the step responses.

to limit the current required to go from low to high spinning frequency, in particular the startup current. In practice, the aerial platforms are hovering at about half of the maximum duty cycle or more, so the gain limitation is seldom triggered.

The values of $\bar{e}_g$ and $\bar{e}_b$ have been chosen empirically to provide a good compromise between low chattering and good response time.

C. Active Braking, Current Safety, and Blockage Safety

Active (or dynamic) braking is a way to improve the deceleration time. This is implemented by controlling the phase bridge so that it produces a counter torque instead of the zero torque provided by the freewheeling mode. Additionally, two safety blocks prevent accidental damage to the hardware: *i)* a current limitation strategy, and a *ii)* short-circuit prevention in case of propeller blockage.

V. Experiments

1) Pure Speed Control Tests: In order to validate the stability, robustness and adaptiveness of ABAG and its implementation, we conducted an experimental campaign using two motors and three propellers with different weights and sizes. The propellers are all shaped for clockwise (cw) spinning. Nevertheless, to test the robustness, we also spun them counterclockwise (ccw), letting them generate much more drag and turbulence. The *same controller parameters* have been used for all the tests (see Table III). *No knowledge* of the model (e.g., feedforward (FF) input or any sort of pre-calibration) has been exploited.

We used a step sequence and a chirp since they allow to see the transient/steady state, and the dynamic tracking performances, respectively. Figure 4 reports two plots of *all* the controller variables for a portion of one particular

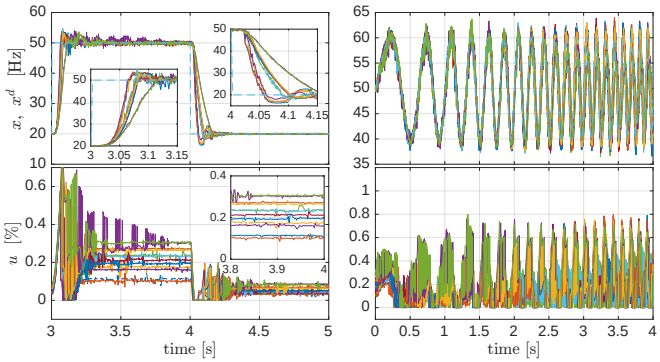

Fig. 5: Outputs x^d, x (top) / input u (bottom) time-zoomed signals for the step (left) / chirp (right) responses for all the experiments.

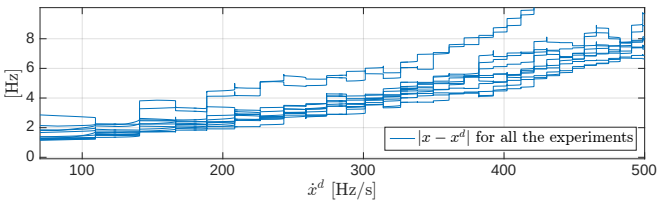

Fig. 6: Cumulative maximum tracking error norms in all the chirp responses w.r.t the desired acceleration $\dot{x}^d$ (Hz/s).

experiment. Time-zoomed plots showing the output tracking and the corresponding inputs for all the experiments are given in Fig. 5. Figure 6 shows the cumulative distribution of the absolute error while tracking the chirp trajectory w.r.t. its acceleration $\dot{x}^d$. As it can be clearly seen, the closed–loop system is asymptotically stable, the performances are very similar across the different experiments regardless the different nature of the systems in each experiment.

In the step sequence experiments, the (finite time) responses are very fast despite the lack of any FF. The step rising time is typically of a few tens of ms, even in presence of current saturations and voltage droppings. Small acceptable overshoots appear in some cases despite the fact that the FF is not known at all. The steady state average error is basically zero and the standard deviation is the same as the one of the measurement noise (see Sec. IV-A.2) (when the current saturation allows). In fact, the noise introduced by the controller is negligible w.r.t. the noise of the measurement. It can also be appreciated how the input u varies significantly across the experiments to automatically accomodate for the different systems.

The tracking of the time-varying chirp trajectories is also excellent. All the experiment show a zero mean error norm with standard deviation below 3 Hz for accelerations smaller than 200 Hz/s. The performances degrade very smoothly (in most cases linearly) for an increasing acceleration. The two outliers are due to current saturation incurring when a small motor is used for too big propeller.

2) Tests in Physically Interactive Aerial Tasks: The possibility to control and measure the propeller frequency has made possible the control of the interaction force in physically interactive aerial task. A first example is the one shown in the experiments of [1] in which a quadrotor takes off from a 50° sloped surface with the help of a tether anchored to the ground. The fine control of the propeller speed done by the ABAG algorithm (the error standard deviation is below 0.5 Hz) makes possible the fine control of the tether tension thus allowing this challengin take-off maneuver.

A second example is the one shown in [11] where a tilted-propeller hexarotor is in contact with a stiff environment and is controlled by means of an admittance force control. The admittance force control scheme requires a good tracking of the desired position and the estimation of the interaction force, which is done using a momentum based wrench estimator. Both requirements are obtained thanks to the underlying ABAG algorithm.

VI. Conclusions

In this paper we have presented a robust and low-complexity control algorithm for the propeller speed of multi-rotor aerial robots. We have demonstrated its implementability on a computationally limited ESC and experimentally validated its stability, robustness and applicability to aerial robotics tasks in which physical interaction with the environment is required.

In the future we plan to improve even more the responsiveness of the controller with e.g., some adaptive model based technique. We also plan to investigate the possibility to adopt a similar method for the control of the aerodynamic power.

References

[1] M. Tognon, A. Testa, E. Rossi, and A. Franchi, "Takeoff and landing on slopes via inclined hovering with a tethered aerial robot," in *2016 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Daejeon, South Korea, Oct. 2016, pp. 1702–1707.

[2] G. Gioioso, M. Ryll, D. Prattichizzo, H. H. Bülthoff, and A. Franchi, "Turning a near-hovering controlled quadrotor into a 3D force effector," in *2014 IEEE Int. Conf. on Robotics and Automation*, Hong Kong, China, May. 2014, pp. 6278–6284.

[3] G. Muscio, F. Pierri, M. A. Trujillo, E. Cataldi, G. Giglio, G. Antonelli, F. Caccavale, A. Viguria, S. Chiaverini, and A. Ollero, "Experiments on coordinated motion of aerial robotic manipulators," in *2016 IEEE Int. Conf. on Robotics and Automation*, May 2016, pp. 1224–1229.

[4] F. Forte, R. Naldi, A. Macchelli, and L. Marconi, "Impedance control of an aerial manipulator," in *2012 American Control Conference*, June 2012, pp. 3839–3844.

[5] B. Yüksel, C. Secchi, H. H. Bülthoff, and A. Franchi, "A nonlinear force observer for quadrotors and application to physical interactive tasks," in *2014 IEEE/ASME Int. Conf. on Advanced Intelligent Mechatronics*, Besançon, France, Jul. 2014, pp. 433–440.

[6] S. Rajappa, M. Ryll, H. H. Bülthoff, and A. Franchi, "Modeling, control and design optimization for a fully-actuated hexarotor aerial vehicle with tilted propellers," in *2015 IEEE Int. Conf. on Robotics and Automation*, Seattle, WA, May 2015, pp. 4006–4013.

[7] M. Ryll, D. Bicego, and A. Franchi, "Modeling and control of FAST-Hex: a fully-actuated by synchronized-tilting hexarotor," in *2016 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Daejeon, South Korea, Oct. 2016, pp. 1689–1694.

[8] D. Brescianini and R. D'Andea, "Design, modeling and control of an omni-directional aerial vehicle," in *2016 IEEE Int. Conf. on Robotics and Automation*, Stockholm, Sweden, May 2015.

[9] M. Bangura, H. Lim, H.-J. Kim, and R. Mahony, "Aerodynamic power control for multirotor aerial vehicles," in *2014 IEEE Int. Conf. on Robotics and Automation*, Hong Kong, China, May 2014, pp. 529–536.

[10] V. I. Utkin and A. S. Poznyak, "Adaptive sliding mode control with application to super-twist algorithm: Equivalent control method," *Automatica*, vol. 49, no. 1, pp. 39–47, 2013.

[11] M. Ryll, G. M. F. Pierri, E. Cataldi, G. Antonelli, F. Caccavale, and A. Franchi, "6D physical interaction with a fully actuated aerial robot," in *2017 IEEE Int. Conf. on Robotics and Automation*, Singapore, May 2017.