Autopilot framework with INDI RPM control, real-time actuator feedback, and stability control on companion computer through MATLAB generated functions

Alessandro Mancinelli,* Erik Van Der Horst,† Bart D.W. Remes‡ and Ewoud J.J. Smeur§
Delft University Of Technology, Kluyverweg 1, Delft, NL

ABSTRACT

This paper presents a modular autopilot framework for Unmanned Aerial Vehicles (UAVs) that addresses the limitations of modern flight controllers. The framework utilizes separate and external subsystems for actuator control and the resolution of the Control Allocation problem. The actuator subsystem, implemented on a Teensy 4.0 microcontroller, incorporates an Incremental Dynamic Inversion Control (INDI) RPM controller, enabling direct control of motor RPM and facilitating the implementation of dynamic inversion-based control laws. The primary flight computer, a Cube Orange, coordinates the system, while an OrangePi 5 single-board computer serves as a companion computer. Real flight results demonstrate the effectiveness of the framework, highlighting its potential for robust and efficient UAV control.

http://www.imavs.org/

1 INTRODUCTION

Unmanned Aerial Vehicles (UAVs), are aircraft designed to operate without a human pilot on board. UAVs have captivated a vast audience and found profound significance through their unmatched versatility, accessibility, and diverse range of applications across industries. From breathtaking aerial photography and aerial manipulation to efficient delivery and logistics.

In recent years, with growing interest from both industry and research communities, the field of Unmanned Aerial Vehicles (UAVs) has witnessed rapid advancements. The integration of a large number of actuators in UAVs has increasingly become prevalent, as exemplified by notable studies such as [1, 2, 3, 4]. The inclusion of an increasing number of actuators in UAVs serves a crucial role in achieving propulsion redundancy and enhancing overall performance. A prime example illustrating this is the dual-axis tilt rotor quad-plane depicted in Figure 1, where a total of 14 actuators, specifically 4 motors and 10 servomotors, are utilized to independently control the vehicle's 6 degrees of freedom.

Figure 1: A picture of the dual-axis tilting rotor quad-plane developed at the TUDelft MAVLab.

Managing a significant number of actuators presents a complex challenge for modern commercial flight controllers, such as The Cube[1] , Pixhawk 6c[2] , Pixracer[3] or the Lisa M[4]. These flight controllers offer limited Pulse-Width Modulation (PWM) actuator outputs, which are sometimes further restricted due to the shared PWM lines with auxiliary peripherals and sensors. Consequently, the integration of an external subsystem becomes indispensable to effectively handle the growing quantity of actuators in modern UAVs. Moreover, this external flight system must possess the capability to monitor the state of the actuators in real-time. The significance of precise real-time estimation of the actuator state becomes particularly evident in contemporary UAV design, where the Incremental Nonlinear Dynamic Inversion (INDI) control method is widely used[5, 6]. This advanced control method heavily relies on accurate actuator state estimation to ensure optimal performance. Traditionally, given the lack of actuator state feedback, the estimation of the actuator state is obtained by feeding the desired actuator input into an actuator model. The actuator model must be identified through laborious tests, which can introduce inaccuracies and consequently impact the quality of the estimation.

The ability to achieve reliable real-time measurements of the actuator state yields substantial advantages that extend beyond enhanced control performance. It assumes a pivotal role in promptly detecting actuator faults, facilitating the design

*a.mancinelli@tudelft.nl
†e.vanderhorst@tudelft.nl
‡b.d.w.remes@tudelft.nl
§e.j.j.smeur@tudelft.nl

[1] https://www.cubepilot.com/#/cube/features
[2] https://docs.px4.io/main/en/flight_controller/pixhawk6c.html
[3] https://docs.px4.io/main/en/flight_controller/pixracer.html
[4] https://wiki.paparazziuav.org/wiki/Lisa/M_v2.0

http://www.imavs.org/

and implementation of effective fault-tolerant control strategies. By possessing a dependable estimation of the actuator state, UAV systems can swiftly identify deviations or anomalies in actuator behavior, enabling timely interventions.

Furthermore, a notable constraint of commercial flight controllers lies in their limited computational power. Typically, these flight controllers utilize ARM Cortex M-family processors, which are specifically optimized for embedded applications. However, when compared to ARM A-family processors found in Single Board Computers (SBCs) like the Raspberry Pi [7], their computational power falls short. Embracing more powerful processors presents a promising opportunity to significantly enhance the computational capabilities of commercial flight controllers, enabling the execution of complex algorithms to support the UAV control and operations.

One potential way to improve the computational power of commercial flight controllers is through the adoption of an external companion computer, as demonstrated by previous works such as [8, 9, 10]. In particular, in our previous work [11], we utilized a Raspberry Pi 4B to solve the non-linear control allocation problem for the dual-axis tilting rotor quadplane. This approach has proven to be highly successful and reliable.

In this paper, we present a modular autopilot framework that utilizes separate subsystems to address the aforementioned limitations of modern flight control. The primary flight computer employed is a Cube Orange, which runs the popular Paparazzi UAV autopilot software [12]. The main task of the primary flight computer is to estimate the vehicle states, run the autopilot routines and coordinate all the other sub-modules. As a companion computer sub-module, an OrangePi 5[5] SBC was utilized, featuring a Rockchip RK3588S octa-core 64-bit processor. A notable feature of the OrangePi 5 SBC is its ability to run a Linux operating system, enabling the compilation and execution of functions developed in the MATLAB environment using the MATLAB C-coder toolbox[6]. For the actuator control and real-time feedback subsystem, a Teensy 4.0[7] micro-controller was employed. The primary flight computer communicates with all the subsystems through custom modules developed in Paparazzi UAV , utilizing UART communication.

The paper is structured as follows: Section 2 presents the Modular Autopilot framework, which encompasses the Paparazzi UAV autopilot running on the primary flight computer. It also explores the information flow among all the subsystems and the primary flight computer. Section 3 provides a detailed analysis of the actuator control and real-time feedback subsystem, providing insights into its functionality and performance. Within this section, an INDI RPM controller is also presented. Section 4 focuses on the companion computer sub-module and explores its setup for running MATLAB functions. Section 5 discusses and showcases real flight results obtained from the implemented framework. Finally, Section 6 draws conclusions based on the presented findings and discussions.

[5] http://www.orangepi.org/
[6] https://nl.mathworks.com/help/coder/index.html
[7] https://www.pjrc.com/store/teensy40.html

2 AUTOPILOT STRUCTURE

This section presents the Autopilot structure, covering both hardware and software aspects. Special attention is given to the working principles of Paparazzi UAV, the software running on the primary flight computer. The Paparazzi UAV flight software, running on the primary flight computer, is made available to the reader[8].

2.1 *Main Autopilot Hardware Structure*

The overall Autopilot structure is illustrated in Figure 2. It can be observed that the AP structure consists of several parts. The main part is the Primary flight computer, where the estimation, stability, guidance, and navigation routines are implemented and executed. To achieve accurate estimation of flight states and position, the primary flight computer interfaces with multiple sensors, located both internally within the Orange Cube module and externally.

The internal sensors of the Orange Cube module include an Invensense ICM42688, an Invensense ICM20948, an Invensense MS5611 I2C barometer, and an AK099916 magnetometer. All the internal sensors are mounted on a temperature-controlled, vibration-isolated board.

For state estimation purposes, the primary flight computer utilizes also the following external sensors:

- UBLOX ZED-F9P RTK GPS connected through a Serial connection.
- Angle of Attack sensor employing a Megatron MAB12A magnetic encoder connected through PWM input.
- Sideslip angle sensor employing a Megatron MAB12A magnetic encoder connected through PWM input.
- External Drotek RM3100 magnetometer connected through the I2C bus.
- 4525DO Airspeed sensor employing a differential MS5611 barometer connected through the I2C bus.
- TFMini lidar with a maximum range of 15 meters, connected through the I2C bus.

All the sensor readings, except for the Sideslip and Angle of Attack readings are fused together using an Extended Kalman Filter based on the Estimation and Control Library (ECL) from PX4[9], running on the Paparazzi UAV flight software.

[8] https://github.com/alessandro-mancinelli/Modular_AP_PPZ.git
[9] https://docs.px4.io/main/en/advanced_config/tuning_the_ecl_ekf.html

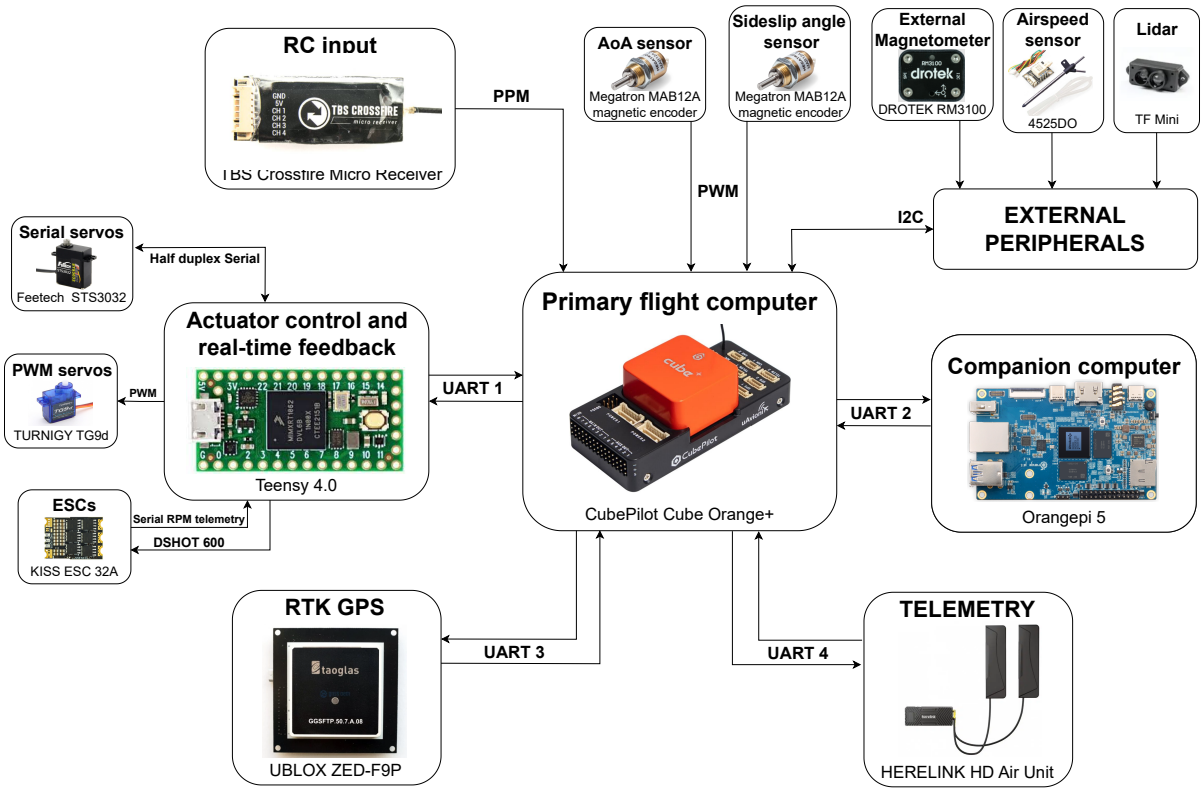

Figure 2: Hardware components and their intercommunication.

Regarding the Remote Control (RC) communication with the safety pilot of the UAV, an RC input source is established using a TBS Crossfire Micro receiver connected through a PPM signal. Additionally, a Herelink HD Air unit connected through UART is utilized to establish communication with the Ground Control Station.

A serial communication is utilized for the communication between the Primary Flight Computer and both the Actuator control and the real-time feedback subsystem, and the Companion computer subsystem. In our specific case, the companion computer is responsible for solving the control allocation problem in real-time. Since the solution needs to be communicated to all the actuators, this requires a fast and reliable communication channel. To minimize overhead, we implemented a UART communication protocol with one starting byte and one checksum byte. The baud rate chosen for the Serial communication is 1.5 megabaud.

2.2 *Paparazzi UAV flight software*

Paparazzi UAV is a versatile and highly customizable flight software that offers users a wide range of options for configuring their aircraft. At the core of the Paparazzi software is the airframe XML file, which serves as the central configuration hub for selecting and setting up all the necessary components and modules required for a successful flight. This modular approach empowers users to customize their aircraft according to specific requirements by choosing the appropriate sensors, actuators, and other peripherals. A comprehensive overview of the information flow in Paparazzi UAV is available on the wiki [10].

One of the key strengths of Paparazzi UAV is its support for custom modules, which facilitate communication and collaboration among different components. These modules can interact with each other using external global variables or, in a more structured manner, through an intermediary routine known as the Application Binary Interface (ABI) routine. This enables seamless integration and coordination between various functionalities, enhancing the overall capabilities and adaptability of the system.

2.2.1 The ABI publisher publish/subscribe middle-ware

ABI is a custom publish/subscribe middleware that facilitates the exchange of data between software components. Within the Paparazzi UAV software, two different modules can exchange information through ABI. Once the ABI message structure is defined, each module can utilize the ABI routine as either a publisher or a subscriber of information. The role of a publisher is to update the content of the ABI node with new information to be broadcasted in the autopilot. One or

[10] https://wiki.paparazziuav.org/wiki/DevGuide/DesignOverview

http://www.imavs.org/

http://www.imavs.org/

more modules can then subscribe to that ABI node. An interesting aspect is that the subscriber node is notified whenever the ABI message contains fresh data and automatically triggers a function to post-process the data. In the design of the Modular Autopilot framework, we extensively utilized the ABI middleware. Specifically, we associated an ABI message with both the Actuator Control and Real-time Feedback subsystem and the Companion Computer subsystem, enabling interaction between our control module and these subsystems.

2.2.2 Custom Teensy actuators Paparazzi module

The custom module called serial_act_t4, developed within the Paparazzi UAV software, serves as the software back-end running on the Primary Flight Computer. Its purpose is to enable communication between the Actuator Control and Real-time Feedback subsystem running on the Teensy 4.0 and the Primary Flight Computer. The messages exchanged through the Serial UART channel between the Teensy 4.0 and the Primary Flight Computer are defined in the ca_am7.h file, which is located in the "Modular_AP_PPZ/sw/airborne/modules/sensors" path. It should be noted that the "Modular_AP_PPZ" refers to the Paparazzi repository available at the TUDelft github page[8]. In the serial_act_t4.h file, two structures are defined: serial_act_t4_data_in and serial_act_t4_data_out, representing the inbound and outbound messages, respectively. In our case, we have decided to include a rolling message with lower priority in the main message struct. Whenever a message is sent, one element of a float array with a length of 255 is also transmitted. Consequently, the update rate of the float array is approximately 0.4% of the refresh rate of the main message.

This module is associated with a publishable ABI message named "SERIAL_ACT_T4_OUT". Users can publish to this message from any location within the Paparazzi software, allowing the outbound message structure to be sent over the hardware serial interface to the Teensy.

Similarly, an ABI broadcast message called "SERIAL_ACT_T4_IN" is triggered whenever a new inbound message structure is received from the Teensy. This broadcast message can also be accessed from any location within the Paparazzi software.

To gain a comprehensive understanding of how to subscribe to or publish an ABI message, readers are encouraged to consult the Paparazzi UAV ABI documentation[11]

2.2.3 Custom Orangepi PPZ module

Similarly to the implementation of the serial_act_t4 module, a custom module called ca_am7 module was developed within the Paparazzi UAV software as a back-end running on the Primary flight computer to enable communication with the OrangePi 5. The inbound and outbound message structures exchanged with the OrangePi are defined in the ca_am7.h file, located in the "Modular_AP_PPZ/sw/airborne/modules/sensors" path. Just like the serial_act_t4 module, we also included a rolling message with lower priority in the main message struct.

The ABI broadcast message containing the inbound data structure from the OrangePi is named "AM7_IN", while the publishable ABI message for the outbound data transmission is named "AM7_OUT".

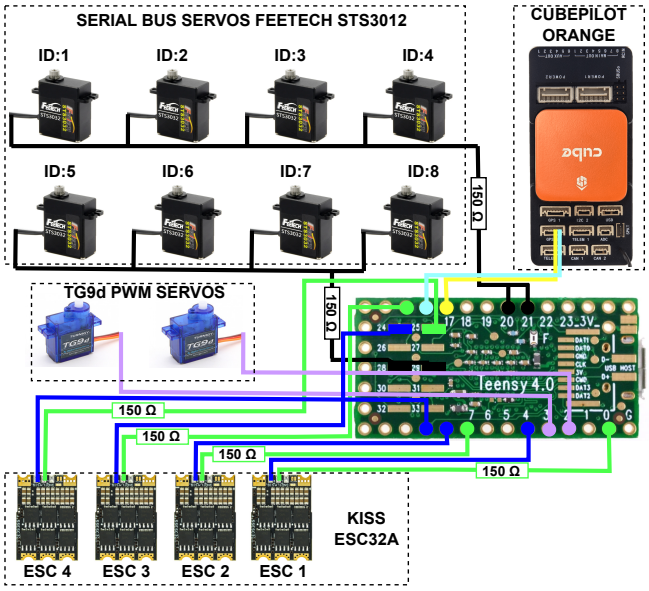

Figure 3: Electrical schematic of the actuator control and real-time feedback subsystem. The 150 Ω resistors are utilized to safeguard the bus from potential transmission conflicts.

3 ACTUATOR CONTROL AND REAL-TIME FEEDBACK SUBSYSTEM

In this section, we present the code of the Actuator control and real-time feedback subsystem running on the Teensy 4.0. We also provide a brief overview of the protocol used to control both the serial servos, PWM servos, and the ESCs. The source code for generating the binary file for the Teensy 4.0 is available in the Teensy_actuators_PPZ GitHub repository[12]. The software is developed using the Arduino IDE and the Teensyduino add-on[7]. The electrical scheme of the actuator control and real-time feedback subsystem is depicted in Figure 3.

3.1 Serial bus servos control

In the current configuration, the Teensy 4.0 microcontroller controls a total of 8 Feetech STS3012 serial servos.

[11] https://paparazzi-uav.readthedocs.io/en/stable/developer_guide/abi.html

[12] https://github.com/tudelft/Teensy_actuators_PPZ.git

Header	ID number	Data Length	Command	Parameter	Checksum
0x55 0x55	ID	Packet Length	Cmd code	Prm 1... Prm N	Checksum

Table 1: Command packet frame for the communication between the Teensy 4.0 and the Feetech STS3012 servos.

These 8 servos are controlled using two separate buses, with each bus connected to 4 servos. The protocol used to communicate with the serial servos is a single-wire half-duplex serial protocol. It involves standardized packet transmission between a master device (the Teensy 4.0) and slave devices (the STS3012 servos). An overview of the packet frame is provided in Table 1.

The master device sends an instruction packet on the bus, addressing a specific servo ID. All devices on the bus receive the packet, but the servos discard it if their ID does not match. Only the servo with the matched ID executes the instruction and then sends a confirmation packet.

The servo ID is a value ranging from 0 to 253, which means that a maximum theoretical number of 254 servos per bus is supported. However, since the bus is shared among all servos and the bus bandwidth is limited to 1 Mega baudrate, increasing the number of servos will limit the refresh rate of the instructions that can be sent to each servo.

Furthermore, it should be noted that if we want to both command a servo position and obtain servo position feedback, two instruction packets have to be sent to the servo. The first instruction contains the desired servo position, to which the servo will respond with an ack packet. Then, we need to send another instruction requesting the servo's current angular position, to which the servo will respond with a packet containing its current position. This can be extended to include additional information such as load, tension, or angular speed. For a detailed explanation of all the possible commands for the serial servos, the reader can refer to the Documentation/Serial_servo_protocol.pdf document in the Teensy_actuators_PPZ GitHub repository[12].

We observed that typically each servo takes 20 to 50 microseconds to respond to a Teensy request. Due to this response time and our requirement for high-speed position control and feedback from the servos, we decided to use two separate buses to control the 8 servos. This approach allowed us to achieve a refresh rate of 350 Hz for both the angular position control and angular position feedback of each servo, while providing enough time to avoid bus conflicts. As an additional safety feature, we also added a 150 Ω resistor to the serial line. This resistor helps limit the maximum current flow on the Teensy serial pin to 22 mA in the event of conflicts on the serial line. Without the resistor, the current flow on the Teensy serial pin in the case of a conflict would exceed the maximum limit of 25 mA tolerated by the microcontroller, potentially resulting in chip damage.

3.2 PWM servos control

On top of controlling Serial bus servos, the Teensy_actuators_PPZ software running on the Teensy 4.0 can also control conventional PWM servos. In our current setup, we have utilized two PWM TGYd micro servos, which are assigned to the ailerons of the airframe. These servos, due to their communication technology, do not feature any feedback regarding their angular position. Therefore, an estimation of the servo position is carried out on the Teensy board. The estimation is based on a first-order transfer function that incorporates the command and the user-provided first-order dynamics characteristics of the servo. These parameters can be identified through a step response test using external equipment, such as an IMU mounted on the servo arm, as demonstrated in [3].

The routine responsible for PWM servo control and estimation is implemented in the *writeEstimatePwmServos()* function, which can be found in the servo_esc_control_w_feedback_T4_PPZ.ino file in the Teensy_actuators_PPZ GitHub repository[12].

3.3 Control of the ESCs

The KISS ESC 32A in our system are controlled using Digital Shot (DShot) commands at a speed of 600 bit/s. The DShot protocol has become widely adopted as a digital communication protocol specifically designed for controlling ESCs. It offers several advantages over conventional UART communication.

One notable advantage of the Dshot protocol over the UART protocol lies in how zeros and ones are identified in the communication. In DShot, the differentiation between zeros and ones is based on the duration of the high state voltage, rather than relying on traditional binary representation through voltage levels. This utilization of timing information allows for more precise encoding of data.

The DShot packet frame consists of a total of 16 bits. The initial 11 bits are dedicated to identifying the throttle command. A value of all zeros indicates that the motor is disarmed, while values 1-47 are reserved for special purposes. Consequently, the throttle command can range from 0 to 2000. Following the 11-bit throttle command, one bit is allocated for telemetry request, and an additional 4 bits contain the cyclic redundancy check (CRC) for message integrity.

When the telemetry bit is enabled, the ESC outputs a telemetry packet through a dedicated serial line operating at a baud rate of 115200. The telemetry packet comprises 10 bytes of information, including temperature, voltage, current, and RPM data from the ESC. For a more detailed explanation

http://www.imavs.org/

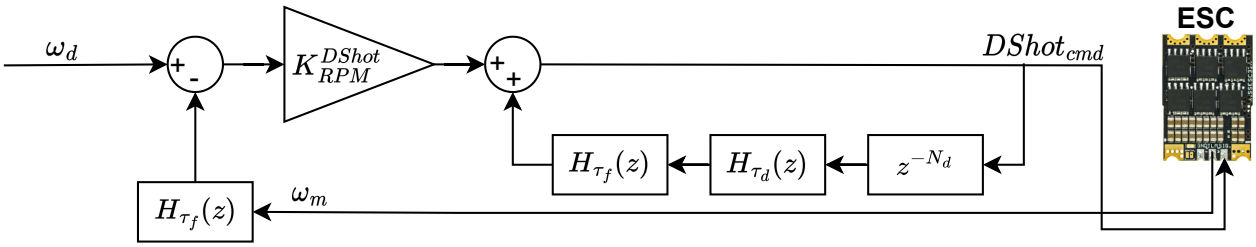

Figure 4: Scheme of the RPM INDI controller.

of the telemetry protocol, the reader can refer to the Documentation/KISS_ESC_telemetry_protocol.pdf file available in the GitHub repository[12].

Due to the limited baud rate of this serial line, we decided to employ a refresh rate of 500 Hz for both DShot throttle commands and telemetry transmission. Keeping within this limit ensures reliable and timely telemetry updates from the ESCs.

3.3.1 Motor RPM control

The high-speed command and telemetry capabilities of the ESCs have allowed for the design and implementation of an RPM controller for the motors. The ability to reliably control the motor's RPM is a crucial feature for modern controllers that rely on model inversion to compute the actuator solution. By directly setting the actuator RPM, the need to accurately model external factors such as battery discharge or propeller inflow is eliminated. This simplifies the control process and enhances the overall performance of the system.

The INDI RPM controller scheme used to control the motor RPM is illustrated in Figure 4, where ω_m is the RPM value measured by the ESC and ω_d is the desired RPM. The transfer function $H_\tau(z)$ represents a low-pass filter in the discrete-time domain with a time constant of τ:

$$H_\tau(z) = \frac{\tau}{z - (1 - \tau)}, \tag{1}$$

where z denotes the discrete-time step delay operator.

Through experimentation, we determined that this filter with a time constant of $\tau_f = 0.1131$ effectively filters out the noise present in the ESC readings. With the ESC telemetry frequency of 500 Hz, this corresponds to a first order low-pass filter with a cutoff frequency of 60 rad/s in the Laplace domain.

Another filter of the form of Equation 1 with parameter τ_d represents the first-order dynamics of the motor system, while N_d in Figure 4 represents the discrete-time steps of delay introduced by the ESC for processing the DShot command. To determine these parameters, we analyzed the step response for various DShot commands. The analysis of the system response, as shown in Figure 5, yielded an estimated value of

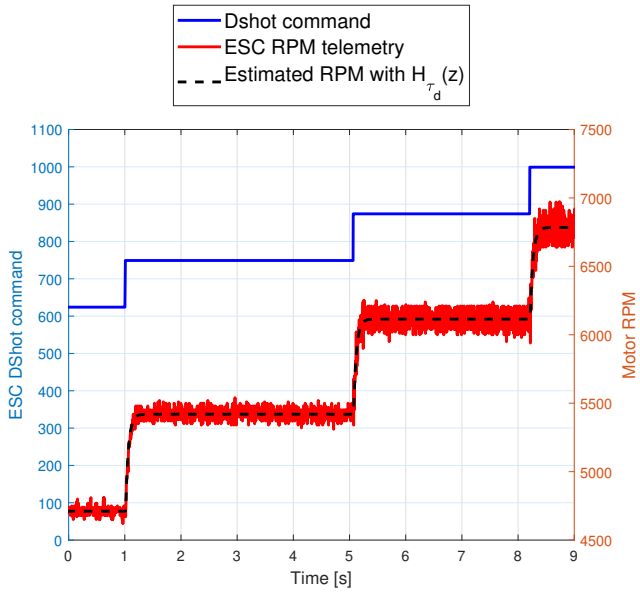

Figure 5: Identification of the motor dynamics.

0.039 for τ_d and 3 for N_d. It is important to note that these values are specific to the tested set of propellers and to an RPM control loop update rate of 500 Hz, matching the ESC telemetry refresh rate.

The gain K_{RPM}^{DShot}, on the other hand, represents the inverse of the control effectiveness, in this case the mapping from RPM to DShot commands. We opted for a constant gain of $K_{RPM}^{DShot} = 0.096$, which corresponds to the ratio between the maximum RPM and the maximum DShot command.

To illustrate the performance of the INDI RPM controller, a step response from an initial RPM of 5000 to a desired RPM of 10000 is depicted in Figure 6.

3.4 *Communication with the Primary Flight Computer*

For communication with the serial_act_t4 module in Paparazzi UAV presented in Section 2.2.2, the Teensy also utilizes UART communication using the same inbound and outbound structures defined in the serial_act_t4_data_in and serial_act_t4_data_out data structures. These data structures can be found in the Definitions.h file in the Teensy_actuators_PPZ

http://www.imavs.org/

http://www.imavs.org/

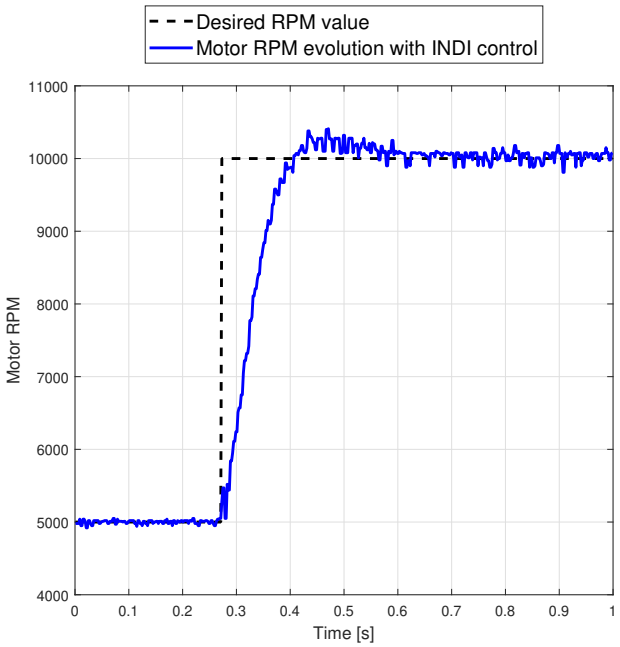

Figure 6: RPM step response using the INDI RPM controller.

GitHub repository[12].

It is crucial to ensure that the structures defined in the Paparazzi module and in the Teensy firmware are identical. Any inconsistencies between the structures will result in improper communication between the hardware modules.

4 COMPANION COMPUTER SUBSYSTEM

As mentioned in the introduction, the utilization of ARM-A family processors in conjunction with the embedded ARM-M family processor significantly enhances the computational capabilities of any robotic platform. In our specific implementation, we employed an OrangePi 5 to tackle the complex Nonlinear Control Allocation problem for the dual-axis tilting rotor quad-plane.

The communication with the custom ca_am_7 module running on the Primary Flight Computer through UART is implemented on a separate thread on the OrangePi 5. Unlike the Teensy 4.0, which has a single-core processor, the OrangePi 5 features an Octa-Core processor. This enables us to distribute the serial communication routine and the control routine across different threads running on different cores.

The software developed for this purpose is available in the GitHub repository OrangePi_PPZ[13]. The software is structured around the am7x.h and am7x.c files. The am7x.h file contains the inbound and outbound data structures for serial communication with the Primary Flight Computer. As mentioned in the previous section, it is crucial for these data structures to match those implemented in the Paparazzi UAV ca_am_7 module, as presented in Section 2.2.3.

[13] https://github.com/tudelft/OrangePi_PPZ

The am7x.c file implements UART communication within one thread and executes the MATLAB-generated function (or any other C function useful for the autopilot) on a separate thread. In our case, we execute a C-function automatically generated by the MATLAB C-coder toolbox. For information on generating a C function from a MATLAB function, please refer to the MATLAB documentation[6].

All the MATLAB-generated files are located in the MATLAB_generated_files folder of the OrangePi_PPZ repository[13] and can be invoked from the am7x.c file.

5 FLIGHT TEST RESULTS

To demonstrate the efficacy of the proposed Autopilot framework, we conducted tests using the dual-axis tilting rotor quad-plane depicted in Figure 1. During the flight, the companion computer subsystem computed the actuator solution, which was then transmitted to the actuator control and real-time feedback subsystem. Figure 8 presents a photograph of the Autopilot hardware, while Figure 7 illustrates the observed evolution of the actuators command and state during the flight test.

6 CONCLUSION

In this paper, we introduced a cutting-edge modular Autopilot framework designed to handle a multitude of actuators while providing real-time, high-frequency feedback on their state. We presented comprehensive software and hardware details necessary for implementing the framework on a wide range of UAVs using the Paparazzi UAV platform. We have also developed an INDI RPM controller leveraging the high-frequency feedback, enabling precise control of motor RPM. Furthermore, the modular Autopilot includes a companion computer subsystem, capable of efficiently solving complex real-time tasks to support the Primary Flight Computer. The effectiveness of the Autopilot functionalities was then demonstrated through a successful flight test of a dual-axis tilting rotor quad-plane equipped with the system.

ACKNOWLEDGEMENTS

The work was carried out within the Unmanned Valley Project. The authors would like to thank the "Europees Fonds voor Regionale Ontwikkeling(EFRO)" who is founding the Unmanned Valley project under grant code KvW-00168 for the South-Holland region.

REFERENCES

[1] C.DeWagter, B.Remes, E.Smeur, F.VanTienen, and R.Ruijsink. The nederdrone: A hybrid lift, hybrid energy hydrogen uav. *International Journal of Hydrogen Energy*, Volume 46, Issue 29, 2021.

[2] Ali Bin Junaid, Alejandro Diaz De Cerio Sanchez, Javier Betancor Bosch, Nikolaos Vitzilaios, and Yahya Zweiri. Design and implementation of a dual-axis tilting quadcopter. *Robotics*, 7(4), 2018.

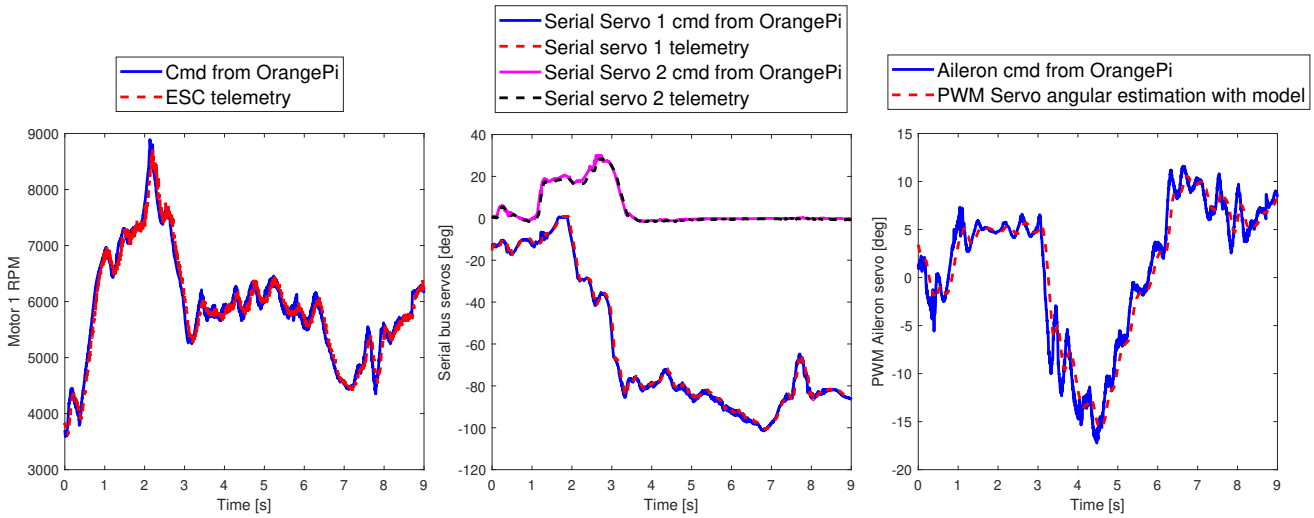

Figure 7: Actuators evolution during a portion of the dual-axis tilting rotor quadplane flight test.

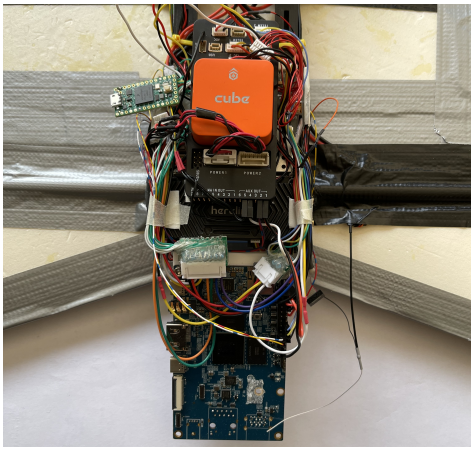

Figure 8: A picture of the Autopilot hardware mounted on the dual-axis tilting rotor quad-plane.

[3] A. Mancinelli, E.J.J. Smeur, B. Remes, and G.D. Croon. Dual-axis tilting rotor quad-plane design, simulation, flight and performance comparison with a conventional quad-plane design. *International Conference on Unmanned Aircraft Systems (ICUAS)*, 2022.

[4] Christos Papachristos, Kostas Alexis, and Anthony Tzes. Efficient force exertion for aerial robotic manipulation: Exploiting the thrust-vectoring authority of a tri-tiltrotor uav. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4500–4505, 2014.

[5] E.J.J. Smeur, G.C.H.E. de Croon, and Q. Chu. Cascaded incremental nonlinear dynamic inversion for mav disturbance rejection. *Control Engineering Practice*, 73:79–90, 2018.

[6] Gabriele Di Francesco, Massimiliano Mattei, and Egidio D'Amato. Incremental nonlinear dynamic inversion and control allocation for a tilt rotor uav. *AIAA Guidance, Navigation, and Control Conference*, 2014.

[7] Gareth Halfacree Eben Upton. *Raspberry Pi User Guide*. Wiley, 2016.

[8] J. Yang, A.G. Thomas, S. Singh, S. Baldi, and X. Wang. A semi-physical platform for guidance and formations of fixed-wing unmanned aerial vehicles. *MDPI sensors*, 2020.

[9] Gustavo Gargioni, Marco Peterson, J. B. Persons, Kevin Schroeder, and Jonathan Black. A full distributed multipurpose autonomous flight system using 3d position tracking and ros. *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1458–1466, 2019.

[10] Hannah Mohr. Uav implementation of distributed robust target location in unknown environments. *2020 IEEE Aerospace Conference*, pages 1–10, 2020.

[11] Alessandro Mancinelli, Bart D. W. Remes, Guido C. H. E. De Croon, and Ewoud J. J. Smeur. Real-time nonlinear control allocation framework for vehicles with highly nonlinear effectors subject to saturation. *Journal of Intelligent & Robotic Systems*, 2023.

[12] B. Gati. Open source autopilot for academic research - the paparazzi system. *2013 American Control Conference*, 2013.

http://www.imavs.org/