A Modular Software Architecture for UAVs

Taygun Kekec, Baris Can Ustundag, Mehmet Ali Guney, Alper Yildirim, Mustafa Unel
Faculty of Engineering and Natural Sciences
Sabanci University
Orhanli-Tuzla 34956, Istanbul, Turkey
Email: {ikekec,bcanustundag,maliguney,alperyildirim,munel}@sabanciuniv.edu

***Abstract*—There have been several attempts to create scalable and hardware independent software architectures for Unmanned Aerial Vehicles (UAV). In this work, we propose an onboard architecture for UAVs where hardware abstraction, data storage and communication between modules are efficiently maintained. All processing and software development is done on the UAV while state and mission status of the UAV is monitored from a ground station. The architecture also allows rapid development of mission-specific third party applications on the vehicle with the help of the core module.**

I. Introduction

The availability of inexpensive, lightweight and compact sensors, low-cost and thin computational platforms, and maturity of control system design capabilities have paved the way to extensive usage of Unmanned Aerial Vehicles (UAVs). The applications of UAVs are becoming more and more apparent. Missions performed by UAVs include surveillance [1], [2], reconnaissance [3], borderline security [4] and remote sensing of environment [5]. Researchers try to optimize cost, scale and flight endurance to come up with efficient solutions [6].

An UAV is governed by a flight computer system. The system reads and analyzes data from a wide variety of sensors and produces a mission flight plan. For observation purposes, UAV carries a payload for acquiring visual overview of the flight environment. While some of the preliminary works on the topic consisted of gathering visual data and processing it off-line, real-time processing is essential and indispensable for missions like threat detection and object tracking.

In addition to physical constraints of developing an UAV system [7], one needs a reliable, flexible, and scalable software component on the flight computer. The software component is responsible for providing hardware abstraction, triggering security checks, handling unexpected conditions, monitoring data and mission progress of the system. Moreover, the system must create a software infrastructure for newly added tasks so that new applications can communicate with the onboard software. Finally, the system must provide an interface for communication of different sources. In Jones' work [8], authors proposed a software architecture for the design and simulation of UAV-based setups. Their work mostly focused on developing a ground station module, can act as a simulator and command controller, providing hardware-in-the-loop capability, simulation sensorial inputs, routing of shared data and generating command requests. Using a graphical configuration system, the user can create artificial events for triggering new actions at specified times to test behaviors and responses of the multiple UAV system.

In Pixhawk project [9], an aerial middleware called MAVCONN, is proposed. Capabilities of the architecture is compared with ROS (Robotics Operating System) and LCM (Lightweight Communications and Marshalling). ROS is an open-source robotics operating system [10], developed by Willow Garage and it is preferred a lot due to its variety in software components. LCM [11] composes of libraries and software components for communication in soft real-time systems. MAVCONN acts as a bridge between ground operator and low level system components, and also provides hardware-level synchronization of visual and inertial data. It also exploits the upper layer of ROS architecture and uses LCM as the communication layer due to its real-time message transmission capabilities. One relevant observation noted in [9] is that many robotic systems still employ polling as part of their design scheme which adds delay to the system and consumes a lot of processing power due to the context switch. Asynchronous design is faster when compared to a polling based design. As asynchronous designes require threads, an onboard middleware needs to adopt multi-threaded implementation.

In Maza's work [12], authors proposed an architecture for UAV cooperation. Their architecture is divided into two layers. First layer, Executive Layer is responsible for generating high level decisions and task planning. Second layer, Onboard Deliberative Layer is responsible for the execution of the tasks. They define a task as having multiple discrete states where it can be nested into subtasks. The functionality of their system is shown on a load transportation application.

Lopez [13] proposed a middleware system, implementing common functionalities and communication channels. In their architecture, they propose a service container which acts as a plant for subscriber/requester data services. This container is the core of their architecture which handles name, network and finally resource management onboard. Container automatically handles message subscription, message failure conditions and message delivery. Their proposal is focused on the network centric low-resource embedded applications.

Honvault [14] proposed an architectural framework implementing core components for the development of fault tolerant and real-time applications. The framework which is built on top of a real-time operating system kernel has two facets. First layer provides key algorithms and services. Second layer allows development of new application context for new problems.

The literature survey shows that although UAVs have different capabilities and mission complexities, they require a unified software architecture where components communicate efficiently. Moreover the data must be analyzed during the

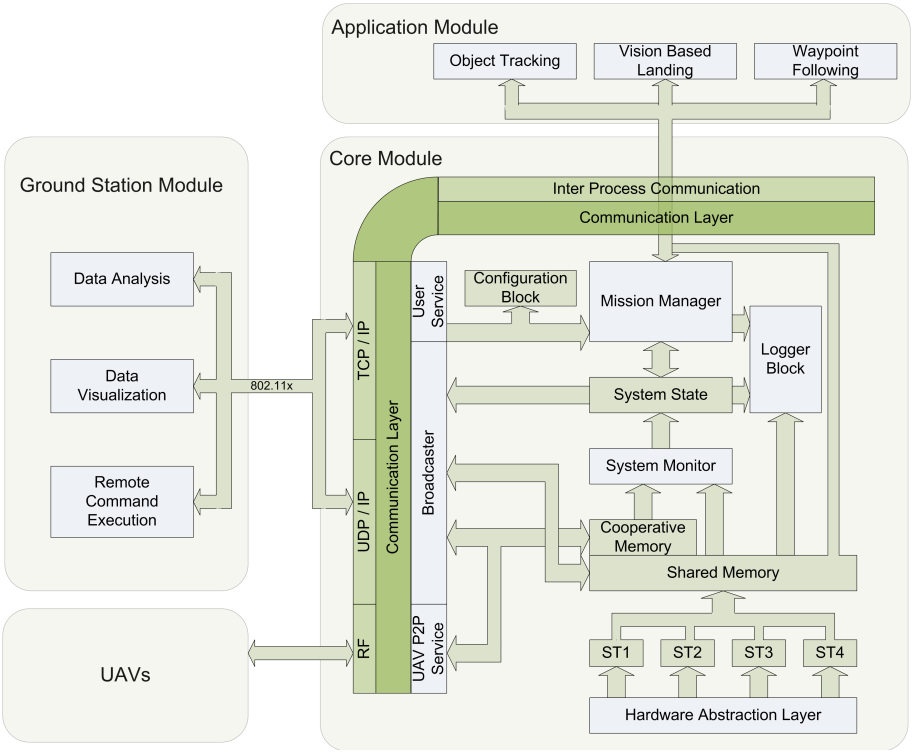

Fig. 1: Block diagram of proposed system architecture.

flight. In this work, we propose a generic architecture for achieving these goals. Unlike some cooperatively architectured systems, we narrow our focus on software framework of individual UAVs. Proposed onboard software framework maintains generic hardware abstraction, data storage and communication duties. The developers are able to monitor the data, create new task blocks where tasks communicate with onboard core module via Inter Process Communication.

The rest of the paper is organized as follows: In Section 2, we present our proposed architecture. In Section 3, we show implementation details of our architecture. Experimental results of the platform is presented in Section 4. The paper is concluded with some remarks and future directions in Section 5.

II. Proposed Architecture

In this section we will describe parts of our proposed architecture. Proposed architecture mainly consists of three main modules. First module, denoted as core module, is the heart of the architecture where all communication, data storage and mission management are handled. Second module is application module which includes mission-specific programs like object tracking and waypoint following. Third module is ground station module where operator performs data analysis, visualization and remote command execution. The overview of the proposed architecture can be seen in Figure 1. In what follows, we will describe functionality of core module's layers and blocks.

A. Core Module

Hardware Abstraction Layer

Hardware Abstraction Layer (HAL) is responsible for managing input/output operations of system peripherals and onboard sensors. Lower part of the layer consists of Serial, SPI, PPM and I2C communication stacks for data acquisition. Upper part of the layer is responsible for copying data to shared memory using multiple threads.

The working principle of the proposed hardware abstraction layer is time-driven where each thread has a predefined working period. Acquired data is copied to shared memory in parallel with the help of several threads. Each thread's period differs with respect to the update rate of the associated device. Using this methodology, shared memory gets an update each time a thread cycle is completed.

Shared Memory

In our proposed architecture, onboard core module stores all numerical data on a shared memory. The memory is updated at several intervals with the help of hardware abstraction layer threads. This area is not only accessible to all blocks of core module but also accessible read-only to application module. Because shared memory is volatile, all fields of shared memory are copied to a non-volatile disk memory using a logger block.

Cooperative Memory

The proposed architecture adopts decentralized communication between UAVs. There are two reasons for using a decentralized communication method. First, when only some

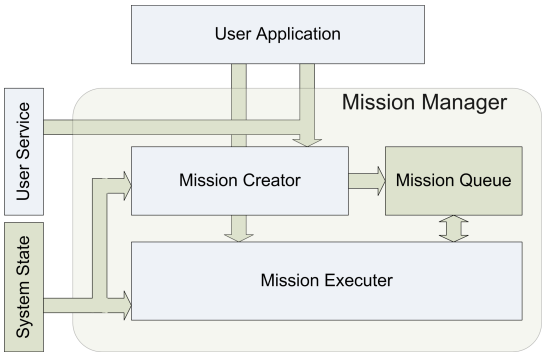

Fig. 2: Internal blocks of mission manager.

of UAVs are in the range of the ground station, remaining UAVs can receive updates through their peers. Second, some missions can require exchanging information in the fastest way, where delay caused by communicating over a ground station can severely affect sharing even packages of short length. Details and implementation of this block is reserved for future work.

Configuration Block

This block stores all system specific settings for onboard core module. Various settings include network settings, device access settings for hardware abstraction layer. This block is accessible to all blocks of core module. Moreover, operator can also access and modify system settings through a User Service block via TCP/IP connection.

System Monitor and System State

Core module has all numerical data required for performing algorithms. A flight system also requires detection of failures and malfunctioning. In this architecture, system monitoring block analyzes data residing in shared memory and produces logic-level system states such as validity of each sensor's functionality and battery level monitoring. Output of system monitoring block is a number of logical data stored in system state block. Logical data stored in system state is also used by mission manager, as most of the missions on UAV require certain state configuration.

Communication Layer

The onboard software architecture must transmit/receive various data. When the UAV is operational and core module is online, the communication requirements can be grouped into three parts.

First, core module must communicate with other onboard task-specific applications. When an application goes online, it also needs to provide packages about itself to the Mission Manager block such as Mission Definition, Mission Start, and Mission Status. Mission Manager will respond each interval whether it is suitable to continue the mission or notify termination of a mission by core module. This communication is done by using an IPC (Inter Process Communication) schema.

Second, core module must communicate with the ground station. An operator can also desire modifying configuration parameters or even trigger mission termination through mission manager. This type of communication is done by User Service block under TCP/IP. Furthermore shared memory and system state are broadcasted to the ground station through TCP/IP by Broadcaster block. Due to the high bandwidth requirements, Broadcaster block must transmit visual data to ground station under UDP/IP.

Third, core module must communicate and share system data with other UAVs. For this purposes, we reserve a UAV P2P(Peer to Peer) Service block. Adopting event-driven schema, the block updates tables in cooperative memory of UAV when a message is triggered. It can also serve as a pipeline for transmitting high priority messages of an UAV, to the ground station through a nearby UAV.

Logger Block

The architecture implements failure detection on system monitor block. However for simulation and playback purposes, volatile data storage must also be supplied with non-volatile storage on the system. For this purposes, logger block is responsible for storing three types of data: shared memory, system state and mission progress data. One can expect first two to be recorded during whole flight while missions are recorded from mission startup to mission completion.

In our logging implementation, we adopted a thread-based logging activity. Logging thread periodically writes shared memory and system state into flight logging directory. Moreover, the block receives periodic announcements of mission based data from mission manager block. Each mission has its own logging directory. The block can also be configured to broadcast all logs to the network. This is beneficial for small scale platforms as they may not have sufficient storage space.

Mission Manager

This block is responsible for registering new and monitoring ongoing missions. On new mission execution request, mission creator analyzes incoming mission request, validates request after analyzing mission requirements from system state block. Creator establishes new mission into mission queue as in Fig. 2.

Mission executor keeps track of missions in the queue. Each mission must report its progress and state to mission executor periodically. If no report is obtained from running mission or mission criteria fail to be sufficed, mission is immediately removed from the mission queue while generating a mission abort message. Missions also have priorities. Core module has a few built-in missions (e.g. emergency land) which have higher priority than developed missions. Automatic triggering of such a mission causes Mission Executor to put a new mission to Mission Queue resulting immediate execution of prioritized mission whilst current mission gets postponed.

B. Application Module

Application module is the space where mission based applications are stored. As operations like mission logging, mission preemption, accessing peripherals are already maintained by core module, one can focus on the task and algorithms while receiving essential services from core module.

Each mission start is triggered by the user from the ground station module. Mission manager checks whether mission requirements are fulfilled, and triggers execution of corresponding mission from application module. These requirements are periodically checked from Mission Executor subblock. All mission progress is logged and can be transferred through User Service to the ground station module. The mission must send acknowledgment to core module periodically or will be terminated due to security purposes.

C. Ground Station Module

Operator can view real-time video of UAV, and send mission execution commands like vehicle landing and vehicle take-off to core module. Core module's shared memory and system states are fully observable using this module. Due to high data transmission bandwidth, it operates through 802.11x wireless protocol.

III. Implementation

The proposed architecture is implemented on our UAV (Fig. 3). It employs a Gumstix Overo microcomputer and Texas Instruments' TMS320F28335 microcontroller for flight control. We implemented lower part of hardware abstraction layer on the microcontroller. Rest of the blocks of core module and application module are implemented on Gumstix microcomputer. Ground station module is implemented on a laptop.

A. Low and High Level Controllers

We employ TMS320F28335 microcontroller for low level control tasks as well as interfacing sensors. Microcontroller has 150Mhz processor speed, 68KB RAM and 512KB Flash memory. The microcontroller is highly capable; it supports 6 capture channels, 16 PWM channels and 16 ADC channels. It provides 96 interrupts of which 58 are reserved for input/output units. Unlike many microprocessors, TMS320F28335 employ zero clock cycle when switching between interrupts. We implemented 100Hz PID control for low level control task on the microprocessor. For tuning purposes, control gains can be modified during the flight from the ground station module where it will be directed to hardware abstraction layer by the help of the core module. Also calibration of the electronic speed controllers and other hardware follow the same procedure.

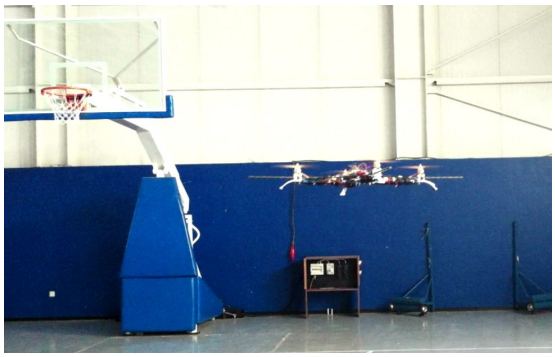

Fig. 3: Implementation platform SUQUAD.

Gumstix microcomputer is utilized as the high level controller of our system. The microcomputer is small and lightweight, weighting only 6 gr. It has a 600 MHz OMAP 3503 microprocessor and C64+ Digital Signal Processor, and a 256 MB DDR RAM. The microcomputer also has 4 hardware PWM channels and fine input/output capabilities where all voltage regulations are performed by its expansion board. The device runs Angstrom Linux distribution. We made slight modifications on the operating system for improving efficiency.

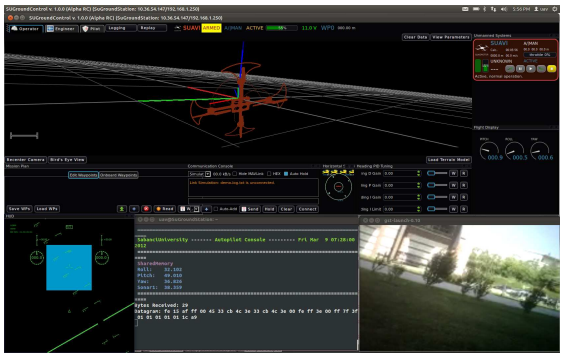

Fig. 4: Real-time data analysis on operator interface.

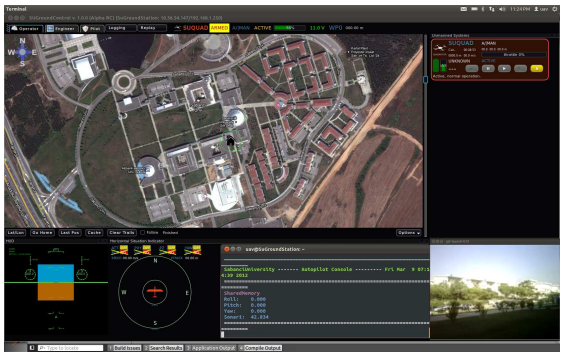

Fig. 5: UAV's position on earth and surveillance video on operator interface.

B. Sensors

We use OmniVision OV 3640 camera of E-Con Systems. The camera has 3.2 Megapixels resolution running under V4L2 driver. It supports resolutions from 320x240 to 2048x1536 supporting Raw RGB, RGB565, YUV, YCbCr image formats. Most of the time, image acquisition is done via microprocessor. However, utilizing Gumstix's DSP speeds up acquisition process significantly. In this work, we use DSPLink library of Texas Instruments for image acquisition so that microcomputer can allocate more processing time to different tasks. On user prompt, frames are transferred to the ground station under H264 encoding format.

We use CHR-6dm inertial measurement unit which combines 3d rate gyros, accelerometers, and magnetic sensors. IMU has 32 bit ARM Cortex processor where it comes with an onboard Extended Kalman Filter implementation reporting roll, pitch and yaw angles at up to 300Hz over a TTL (3.3V) UART interface. In order to measure distances and avoid

	Proposed	MAVCONN	ROS
Scale	lightweight	lightweight	middleweight
Availability	Open Source	Open Source	Open Source
Ground Station Module	+	+	-

TABLE I: Comparison of similar middlewares

	Proposed	PIXHAWK	Asct. Pelican
Autopilot	TMS320F28335	ARM7	ARM7
AP Mhz	150Mhz	60Mhz	60Mhz
AP RAM	68Kb	32Kb	32Kb
Open HW	-	+	-

TABLE II: Comparison of autopilot systems

obstacles, we use MaxBotix EZ4 ultrasonic sensors which give resolution of 1 inch with 20Hz reading rate. The sensor can measure up to 6.45 meters. For high level control tasks GPS module EM-406A, having sensivity of -159dBm, is used. Peer to peer radio frequency communication is done via Digi Zigbee OEM RF module. This device has a communication range of 100 m indoor and 1.6 km outdoor. The module supports point-to-point and peer-to-peer communication topologies.

C. Ground Station

Ground station module is implemented on a computer having 2.0 Ghz I5 Intel Core2Duo processor and 4GB RAM. The ground station is based on open source QGroundControl framework. We modified this open source software for our needs. The software also supports real-time plotting of data (Fig. 4). The operator can also track UAV's position on earth with built-in Google Earth plugin as well as on simulated 3D environment (Fig. 5). Although ground station supports a wide functionality, the implementation is noticeably slow. We plan to replace whole ground station system with a new one in future work.

IV. Experimental Results

We demonstrate an experimental flight of our platform, running proposed architecture. A comparison to similar available middlewares [9], [10] and autopilot platforms is shown at Table I and II. The key concept of our architecture is preserving minimality while providing essential functionality. Moreover, proposed architecture is open source and available. We implemented 100Hz PID control for attitude and altitude control tasks on the microprocessor. The attitude control keeps the orientation of the quadrotor to the desired value. The initial orientation of the quadrotor before takeoff was -0.1° of ϕ and 0.4° of θ due to non-flat surface. Results of attitude control during hover is shown in Fig. 8. It can be seen that attitude angles oscillate in bounded interval of $(-2^\circ, 2^\circ)$ whereas they rarely pass $\pm 1^\circ$. The attitude control is done using cheap sonar sensor. As measurement of sonar is extremely noisy and hard to model, we applied median filtering to the Z position measurements. Altitude control using filtered measurements is shown in Fig. 6. Control efforts of the flight can be seen in Fig. 7. It can be seen that there are very few momentary oscillations in $U1$ control. This shows that quality of hover is very good. Additional images of various flights is shown on Fig. 9.

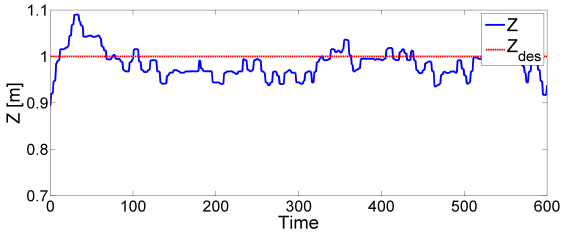

Fig. 6: Altitude Control Performance

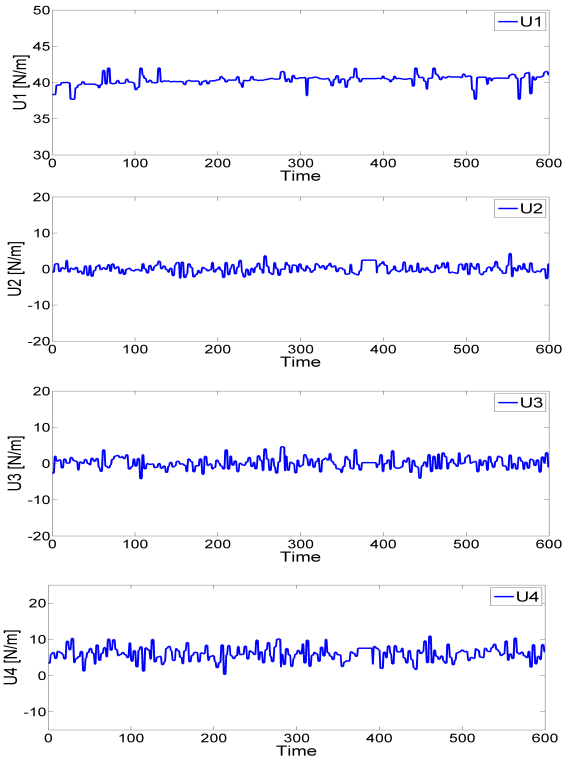

Fig. 7: Control Inputs

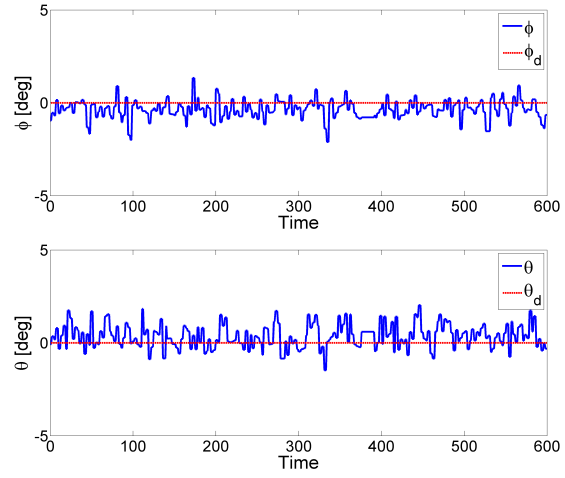

Fig. 8: Attitude Control Performance

Fig. 9: SUQUAD during flight experiments.

V. Conclusion and Future Works

In this work, we have proposed an onboard software architecture for manipulating common tasks. The architecture is responsible for memory management, distribution of sensory data, and communicating with other processes and ground station. It also simplifies mission-specific application development by providing a user-friendly interface.

The proposed system is implemented on our UAV using C++ language and continues to be developed. In future work, we plan to add an extensive P2P communication layout and converting mission manager into a cooperative mission scheduler. We also plan to develop a ground station module which is faster than current one. Moreover we will investigate whether real-time capabilities of Linux platform is powerful enough to implement the whole HAL on the high level computer.

References

[1] M. Quigley, M. Goodrich, S. Griffiths, A. Eldredge, and R. Beard, "Target acquisition, localization, and surveillance using a fixed-wing mini-uav and gimbaled camera," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, 2005, pp. 2600–2605.

[2] R. Beard, T. McLain, D. Nelson, D. Kingston, and D. Johanson, "Decentralized cooperative aerial surveillance using fixed-wing miniature uavs," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1306–1324, 2006.

[3] P. Iscold, G. A. S. Pereira, and L. A. B. Torres, "Development of a hand-launched small uav for ground reconnaissance," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 46, no. 1, pp. 335–348, 2010.

[4] J. Dufrene, W.R., "Mobile military security with concentration on unmanned aerial vehicles," in *Digital Avionics Systems Conference, 2005. DASC 2005. The 24th*, vol. 2, 2005, pp. 8 pp. Vol. 2–.

[5] Y. Lin, J. Hyyppa, and A. Jaakkola, "Mini-uav-borne lidar for fine-scale mapping," *Geoscience and Remote Sensing Letters, IEEE*, vol. 8, no. 3, pp. 426–430, 2011.

[6] H. Lim, J. Park, D. Lee, and H. J. Kim, "Build your own quadrotor: Open-source projects on unmanned aerial vehicles," *Robotics Automation Magazine, IEEE*, vol. 19, no. 3, pp. 33–45, 2012.

[7] E. Cetinsoy, S. Dikyar, C. Hancer, K. Oner, E. Sirimoglu, M. Unel, and M. Aksit, "Design and Construction of a Novel Quad Tilt-Wing UAV," *Mechatronics*, vol. 22, no. 6, pp. 723–745, 2012.

[8] E. D. Jones, R. S. Roberts, and T. C. S. Hsia, "Stomp: a software architecture for the design and simulation of uav-based sensor networks," in *ICRA'03*, 2003, pp. 3321–3326.

[9] L. Meier, P. Tanskanen, L. Heng, G. Lee, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision," *Autonomous Robots*, vol. 33, pp. 21–39, 2012.

[10] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[11] A. Huang, E. Olson, and D. Moore, "Lcm: Lightweight communications and marshalling," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, 2010, pp. 4057–4062.

[12] I. Maza, K. Kondak, M. Bernard, and A. Ollero, "Multi-uav cooperation and control for load transportation and deployment," *J. Intell. Robotics Syst.*, vol. 57, no. 1-4, pp. 417–449, Jan. 2010.

[13] J. López, P. Royo, E. Pastor, C. Barrado, and E. Santamaria, "A middleware architecture for unmanned aircraft avionics," in *Proceedings of the 2007 ACM/IFIP/USENIX international conference on Middleware companion*, ser. MC '07. New York, NY, USA: ACM, 2007, pp. 24:1–24:6.

[14] C. Honvault, M. Le Roy, P. Gula, J. C. Fabre, G. Le Lann, and E. Bornschlegl, "Novel generic middleware building blocks for dependable modular avionics systems," in *Proceedings of the 5th European conference on Dependable Computing*, ser. EDCC'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 140–153.