Authors' accepted manuscript: IEEE 23th Mediterranean Conference on Control and Automation (MED), 2015.
The final publication is available at: https://doi.org/10.1109/MED.2015.7158919

Enhancement of a commercial multicopter for research in autonomous navigation

Andres Gongora
System Engineering and Automation Department
University of Málaga (SPAIN)
Email: andresgongora@uma.es

Javier Gonzalez-Jimenez
System Engineering and Automation Department
University of Málaga (SPAIN)
Email: javiergonzalez@uma.es

***Abstract*— Multicopters are lightweight and maneuverable aerial vehicles yet unable to carry heavy payloads, such as large sensors or computers required for indoor autonomous navigation. Therefore, localization is usually performed by using vision-based solutions employing of either lightweight on-board cameras or external fixed cameras and a ground station for data-processing. Nevertheless, the current tendency is to use a low-power on-board computers to perform all computation on the multicopter itself. This paper covers the enhancement of a commercial multicopter, also called drone, with computation ability and sensorial devices for autonomous flight without the need of a ground-station. We describe the hardware and software integrated into the drone, which will be used for the future development of 6DoF navigation algorithms. The resulting system is able to work with most standard sensors and has the possibility to change them as needed. Also, we demonstrate the correct behavior of the drone by using a test navigation program that autonomously follows a moving beacon at constant distance and controlled altitude using an RGB-D camera and a sonar.**

***Keywords*— Unmanned systems, robotics, education and training**

Fig. 1. Hexacopter enhanced with an on-board computer and extrospective sensors: an RGDB-D camera for visual navigation and a sonar for altitude control.

I. Introduction

Multicopters, also called drones, are gaining attention in the robotic community due to their potential for aerial applications that demand small size, high maneuverability and low cost. They are specially suited for indoor flight due to their ability to hoover and move freely in all directions but also able to fly outdoors. However, one of their main drawback is the reduced maximum weight they can carry, imposing a limit to how much hardware for on-board computation can be built into them. This in turn prevents them to exploit most developments in autonomous robotic navigation, specially in GPS-denied environments, such as indoors, where additional sensors and computation are required.

Autonomous navigation involves the ability to plan obstacle-free paths and actions ahead and dynamically change them when new information about the surroundings becomes available (see for example [12] [10] [16] [18]). Executing all algorithms on board and eliminating the need for a ground station allows for true autonomous navigation. Some of the benefits are fast deployment of the system and the absence of problems caused by a remote connection, such as latency issues and connection losses.

In this paper we describe our approach to the problem of sensor and computation integration into multicopters without exceeding neither weight nor power limits. Starting with a commercial remote controlled hexacopter, we enhance it with an on-board Single Board Computer (SBC) and additional sensors. The SBC is aimed at the possibility to run on-board navigation algorithm that require a reasonable amount of computation power, and that otherwise would run on a ground-station. As for the additional sensors, the SBC makes it possible to connect any kind of generic digital sensor, such as video cameras or laser range-finders. In our setup we have an RGB-D camera, which requires considerable data processing, and a digital sonar, in order to also test simple sensors. The final setup is shown in Figure 1.

Additionally, we describe how we implement an Application Programming Interface (API) that simplifies the task of interfacing with the drone. It was developed in order to allow for fast and easy development of navigation programs that can be integrated into a robotic architecture [3] [11].

Finally, we test the extended features of the system by running a navigation program that uses the RGB-D camera to track a moving beacon, whereas altitude is controlled using sonar and inertial data with a non-linear complementary filter.

II. Background

Multicopter or multirotors are helicopter-like, aerial vehicles with more than two rotors. They are usually small, with a diameter less than 1 meter, and use batteries for energy storage. Because they do not have enough inertia, their flight is unstable, thus requiring constant readjustments of how power is distributed among the rotors to compensate for unwanted drifts. Although prior versions exist, the current format is relatively new (endings of the 2000 decade) being quadcopters (4 rotors) the most popular. Traditionally multicopters are remote controlled vessels that feature only an on-board micro-controller and Inertial Measurement Units (IMUs) for flight stabilization.

In order to use multirotors as aerial robots, that is, as drones, additional computation power is required for autonomous navigation algorithms. In the first attempts, this task has been carried out by ground computers that simply replace the human operator. Sometimes this implementations use additional sensors mounted on the drone [7] [9], but a well-proved approach consists of triangulating the vehicle's position using ground cameras [8] [14] [19]. Even though in essence the multirotor remains remote controlled, this solution is still widely used because of its simplicity and possibility to execute more computation-demanding algorithms.

Due to the recent appearance of small and lightweight yet powerful computers, drones can now carry part or even all needed computation power on-board, alongside additional sensors [13] [17] [22] [21]. This way it is possible to develop true autonomous drones that offers the possibility for fast deployment without the need for extra equipment.

III. Requirements

We use a RTF-Y6 hexacopter [4] in coaxial configuration by 3DRobotics, shown in Figure 3.a, meaning it has three axis with two rotors each. Control of the rotors is performed using Electronic Speed Controllers (ESC) which deliver power to the motors as told by the control layer.

In order to develop the multicopter into a drone with autonomous capabilities aimed at research and prototyping of algorithms for 6 Degrees of Freedom (DoF), the following consideration were taken.

- Lightweight and low power on-board computer with reasonable computation power for navigation algorithms, intended to act as on-board pilot of the vehicle.
- Ability to operate in GPS denied environments, such as indoors, where the drone has no initial reference of where it is or what surrounds it.
- Clean API to simplify the task of writing software for the on-board computer. Future researchers that work with the system should not need to concern with hardware or low-level software details.
- Possibility to communicate with a ground station, preferably using a widely available technology such as WiFi. This link si intended to communicate with the drone during flight, for example, to read telemetry data.
- Mechanism to remotely interfere and manually control the drone in case of emergency. Also this can be used to force the drone into certain situation and test how the navigation software behaves.

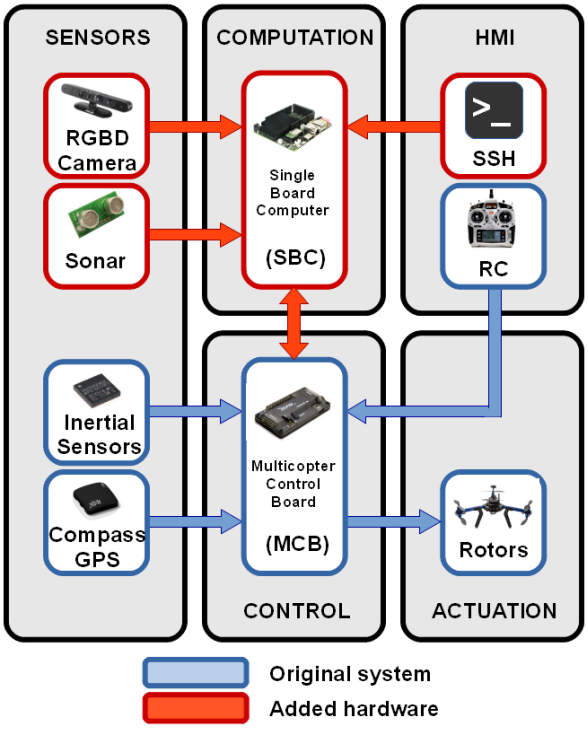

Fig. 2. Hardware architecture of the system. Blue color corresponds to the original system, while orange denotes the added elements. The SBC gives flight orders to the MCB, but is also able to read telemetry data from the MCB.

IV. Hardware architecture

This section presents an overview of the system's hardware architecture,depicted in Figure 2, which shows that the drone is composed of a commercial remote controlled hexacopter, shown in blue, and some additional hardware for autonomous flight, shown in orange. The communications between the on-board SBC and the Multicopter Control Board (MCB) is bidirectional, allowing the SBC to give flight-orders to the MCB and also read telemetry data from it.

Figure 2 also depicts 5 conceptual layers: perception, computation, control, Human-Machine Interface (HMI), and actuation, which are described next.

A. Human-Machine Interface (HMI)

In the commercial system the HMI consists of a Remote Controller (RC) that gives direct control to the user. The RC has been preserved for security purposes. With the RC it is possible to override the SBC in case it wreaks havoc. This is done at software level on the MCB by continuously monitoring channel 5 (out of 7), which determines whether to follow instructions from the SBC (channel 5 active) or from the RC (default). Thus, a switch on the RC allows us to easily change between manual and autonomous flight.

B. Human-Machine Interface (HMI)

It is possible to connect to the SBC via SSH in order to upload the navigation program and to access telemetry data. The connection can be established either using the SBC's WiFi module, or optionally, over USB when the drone is landed. Considering the limited range of WiFi, we use it only for development purposes. We are planning to add a dedicated radio transmiter/receiver for kilometer-range connections in future versions of the drone, such as XBee-Pro.

Alternatively, it is possible to use a radio module connected to the serial port of the MCB to receive telemetry data using MAVLink protocol. We did not resort of it in this first version because all our flights were done relatively close to the WiFi access point and because SSH was much faster to set up.

C. Computation

The computation layer makes it possible to execute on-board navigation algorithms. Therefore, it acts as pilot of the drone, substituting the former RC. Also, it is able of retrieving data from additional sensors and process it.

For this purpose we use a UDOO-quad 4 core cortex A9 1GHz ARM development board [6], shown in Figure 3.c. This board has a reasonable computation power while remaining low power and lightweight. Also, it has several I/O options and an integrated WiFI module.

D. Control layer

The control layer is in charge of controlling the vehicle and stabilizing the hexacopter. It uses inertial sensors to estimate current pose of the multicopter, and corrects it according to the indications from the computation layer.

Although the SBC outperforms the MCB and could integrate computation and control, the MCB has been retained since control loops are very CPU demanding, and would reduce the resources available for navigation algorithms.

The commercial multicopter we started with already integrated an APM-2.6 control board by 3DRobotics [1], shown in Figure 3.b. It runs ArduCopter, a open source multicopter control software, which we have modified to accept instructions from the on-board computer and to report sensor data back when requested.

E. Sensors

Inertial sensors and a compass are used by the control layer to stabilize the flight of the drone, therefore, they are directly connected to the board. Once processed, information about the pose can be accesses by the computation layer to obtain relevant data for the flight such as rotation angles or ascending speed.

Additional sensors can be connected to the computation layer as required by the navigation algorithms. For our test-setup we chose to use an RGB-D camera and a sonar. These sensors represent complete opposites of complexity, data throughput and computation requirements.

Each individual sensor is discussed below.

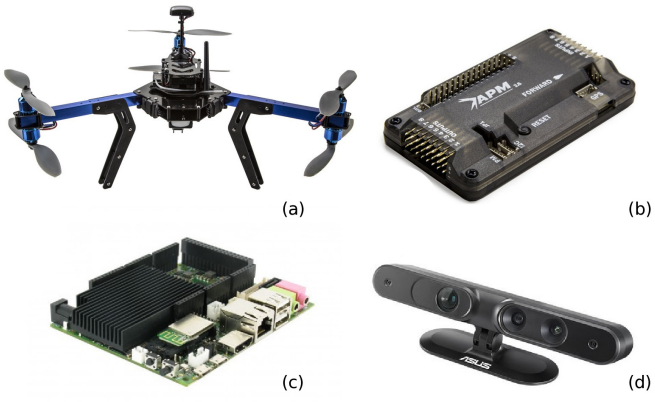

Fig. 3. Some of the employed hardware. (a) RTF-Y6 Hexacopter. (b) APM-2.6 Multicopter Control Board. (c) UDOO-quad 4 core 1GHz ARM development board. (d) Asus Xtion PRO live RGBD camera.

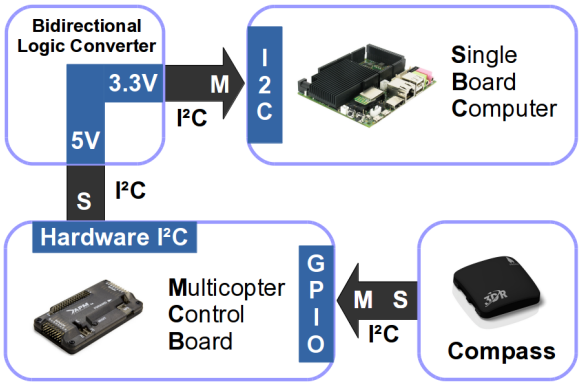

Fig. 4. Communications between the SBC and the MCB. The MCB uses a native I^2C bus to communicate with the SBC, and a software simulated I^2C with GPIO pins for the electronic compass. M denotes Master of the bus and S denotes Slave

1) Inertial sensors: The MCB comes with a MPU-6000 [2] integrated circuit by Invensense, which contains an electronic accelerometer and gyroscope.

2) GPS and Compass: Because electronic motors cause big magnetic field that disturb electronic compasses, and because a GPS needs a obstacle-free view of the sky, both sensors are mounted externally to the MCB. We use a Ublox LEA-6H GPS and HMC5883L compass combo.

3) RGBD camera: We use an ASUS Xtion PRO Live camera, shown in Figure 3.c. It provides video and depth with a maximum resolution of 640x480 and a maximum range of about 4 meters. The depth measure is very light sensible, and therefore not suited to work with direct sunlight exposure.

4) Sonar: A SRF-10 sonar on the bottom of the drone improves height estimations. The SRF-10 performs a measurement every 65 ms in its 6 meters practical maximum range with a resolution of 43mm.

V. Communications between SBC and MCB

The MCB is intended as a stand-alone solution for remote controlled multicopters, and in consequence has no connector foreseen for external commands except for the receiver of a

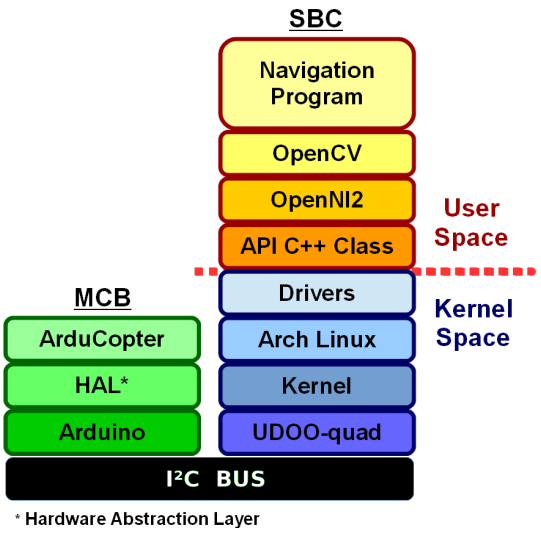

Fig. 5. Software hierarchy. The left stack is for the Multicopter Control Board. The right stack for the Single-Board Computer, which runs linux and is divided into kernel and user space.

remote controller. Using this connector would allow only to give commands to the MCB, but not to retrieve telemetry data. The MCB also features a serial port intended for MAVLink protocol, but we chose not to use it to minimize the modifications to the board and retain the possibility to connect a simple radio set for remote telemetry in future projects.

Our solution consists in using the MCB's Inter-Integrated Circuit (I^2C) port intended for the external electronic compass to establish a link with the SBC. But because I^2C is a single-master to multi-slave bus, it has no mechanism to allow a slave to send data to the master unless explicitly requested. In consequence, there would be conflicts if both boards shared the bus, as both need to be master. The MCB needs to control the compass and at the same time be controlled by the SBC.

Although it is possible to configure I^2C for multi-master communications, we decided to create a second, virtual, I^2C bus on the MCB and use it as shown in Figure 4. This way the MCB uses one as master with the electronic compass, and the other as slave with the SBC. The virtual I^2C bus is software emulated and available through General Purpose Input/Output (GPIO) pins. Because the virtual bus is considerably slower, it is connected to the compass, leaving all the bandwidth of the native bus for the SBC. Finally, because the SBC works at 3.3V and the MCB at 5V, a bidirectional 5 to 3.3 V-logic converter has been added.

VI. Software Architecture

As shown in Figure 5 the MCB runs ArduCopter, an open source software devised to control multicopters and which we modified to communicate with the SBC. Arducopter reads all inertial sensors, estimates the current attitude and calculates the needed thrust by each rotor to move as indicated by the SBC. Then, it sends the control values to Electronic Speed Controllers, which regulate the speed of the rotors.

The SBC runs Arch Linux, a non-real-time OS which, as usual, divides execution time into User Space and Kernel Space. The SBC is fast enough to meet al time requirements, thus we could afford not to use a real-time OS and therefore avoid additional over-head computation. It is intended that the current demo program and future navigation software run in User Space, while sensor drivers and MCB-related code reside in the Kernel Space.

A. User Space

On the top of the User Space is the autonomous navigation program, which exploits the computation capabilities of the SBC to execute robotic algorithms.

In order to work with the RGD-D camera we use OpenNI2, for its drivers. Also, we use OpenCV, which is a library for computer vision that allows for more advanced applications such as pattern recognition and visual tracking of objects. OpenNI2 was directly compiled and installed on-board, meaning that user-applications need not to be cross-compiled.

At the border line between User and Kernel Space is a C++ API aimed at reducing development time of future applications. It hides all details of how communications work between the SBC and the MCB, and also converts data reported by the MCB from non standard notation in integer format to metric units in floating point format. Additionally it creates several threads that are used to update attitude information and obtain parameters which are time related, such as speeds and accelerations.

B. Kernel Space

ArchLinux has been adopted on the SBC because its a lightweight and performance oriented Linux Distribution.

In the Kernel Space we run two I^2C drivers that communicate with the MCB and the sonar. They create binary files in the file-system with thread blocking access in order to increase performance.

VII. Testing the System

In order to test the performance fo the hardware and software integrated in the drone, we have developed a demonstration program. With the RGB-D camera the drone tracks a moving beacon of a previously defined color at constant distance. Altitude is maintained using a non-linear filter that combines the sonar with the accelerometer. Because following a beacon and controlling altitude are independent, they have been implemented to run on separate threads for performance reasons.

We do not intend to demonstrate the performance of a concrete navigation algorithm or technique. Our aim is to test the improved capabilities of the drone to execute tasks that demand a rational amount of on-board computation, such as real time image processing, and to test the communications between the different modules and sensors.

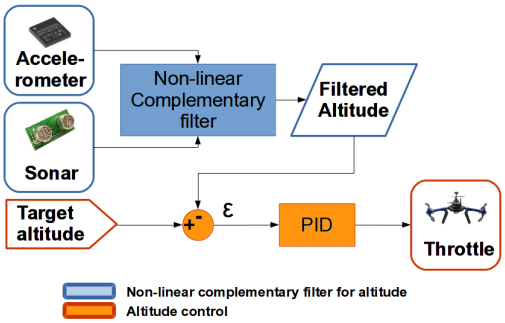

Fig. 6. Blue: non-linear complementary filter [20] for altitude that integrates twice the accelerometer data to obtain the current altitude and eliminates accumulative the bias using the data from the sonar. Orange: PID controller for altitude that commands the drone's throttle.

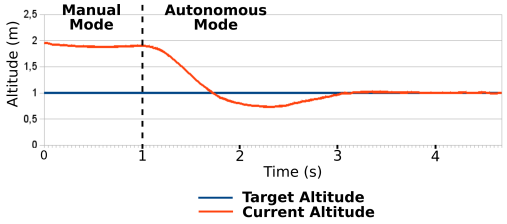

Fig. 7. Altitude PID controller output when setting 1 meter as target altitude and engaging autonomous mode flying at an altitude of 2 meters.

A. Controlling flight altitude

In order to fly at a given altitude, the drone needs to know its current altitude first. On one hand, using the accelerometer it is possible to know the relative altitude with respect to the lift-off altitude by integrating twice over time the vertical acceleration. Nevertheless, this information is not useful as integration slowly introduces a bias error with each iteration.

On the other hand, using the sonar alone is neither a good option, as it works only at a limited range (max 4 meters), has a relatively big error and is prone to important outlier samples.

Our solution was to implement a non-linear complementary filter [20] adapted to one dimension that uses the sonar for an absolute reference and the accelerometer to calculate the rate of change. This combination keeps the best of both sensors: accurate absolute value, high sensibility to small changes and immunity to outlier measurements.

Once the current altitude is known, the hexacopter's altitude is controlled with a PID controller that commands thrust. The altitude control thread behaves as depicted in Figure 6 and runs every 10 ms.

Figure 7 shows the response of the drone when setting the desired altitude to 1 meter and letting it fall from 2 meters. For this test we flow the drone in manual mode to the starting altitude and switched to autonomous mode to observe how it descended as it was expected to.

B. Following a moving beacon

The drone follows the beacon using the RGB-D camera. With the color output it is able to orient itself towards the beacon, and with the depth output it is able to maintain a given distance from it.

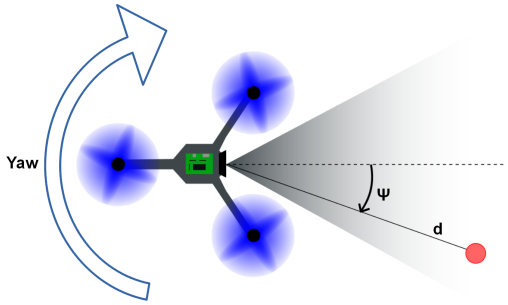

Fig. 8. The angle ψ is obtained with the centroid of the segmented color image that retains only pixels belonging to the beacon. Then, ψ is then used as error input for a PID controller that commands yaw in order reduce ψ to zero, effectively rotating the drone towards the beacon.

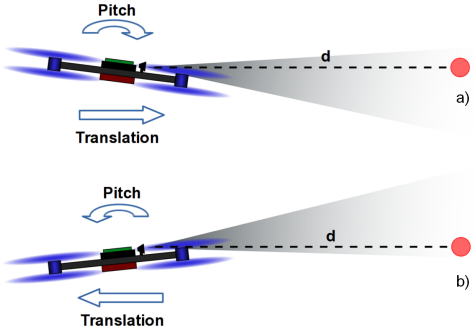

Fig. 9. Distance PID controller response. a) Current distance "d" is greater than target distance, the drone approaches by changing its pitch towards the beacon. b) Current distance "d" is smaller than target distance, the drone retreats by changing its pitch away from the beacon.

The beacon following thread searches for the beacon within the image using color information, and if found, determines its relative position. This is done using a HSV[1] filter from OpenCV to segment the image. Afterwards the centroid of the segmented image is calculated, which in turn allows to calculate the angle to the beacon, denoted ψ in Figure 8. Also with the segmented image, the distance d to the beacon is calculated as the mean value of the depth measured by the RGB-D camera.

The angle ψ is used as input for a PID controller that commands yaw to rotate the drone towards the beacon as shown in Figure 8. A second PID that commands pitch, as shown in Figure 9, controls the drone to keep it at a preconfigured distance from the beacon.

C. Performance of the system

We monitored the system using a WiFi connection to a laptop. The tests were performed attaching the beacon to a mobile robot moving at about 2 km/h and following it at 1 meter and by holding the beacon in one hand walking, and running, in both cases configured to follow at 2 meters.

[1]Hue Saturation Value, a format to represent colors with three numbers similar to RGB, but using a format which can be interpreted more naturally by a human.

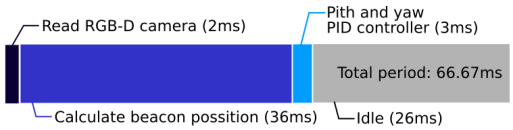

Fig. 10. Average execution time for the beacon following thread at 15 FPS (66.67 ms period).

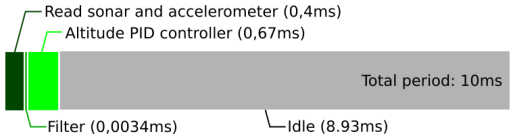

Fig. 11. Average execution time for the altitude control thread at 100 Hz (10 ms period).

The drone is able to identify the beacon in most light conditions, orient itself and follow it at the preconfigured distance. Using the RGB-D camera at 15FPS and the altitude control loop at 100Hz, the system load stayed between 21 and 23%. Setting the sampling frequency of the camera to 25FPS increased the system load to about 35%. For 1GB RAM, memory usage was constant at 2.6% for the test program and 1% for the OS.

Figures 10 and 11 show the average execution time for the demonstration program. Figure 10 shows the consumed time for the beacon-following thread running at 15 FPS, that is, every 66.67 ms, and Figure 11 shows the time for the altitude control thread running every 10 ms. The fact that none of the threads need to run their total period and that they execute concurrently on one of the four CPU's, agrees with an average system load of 23%.

VIII. Discussion and Future Work

The paper describes some key aspects of retrofitting a multicopter with an on-board computer and additional sensors with the goal of using it in future research and development of autonomous aerial navigation. The use of a small and lightweight Single Board Computer makes it possible to process sensor-data and execute software to fly itself without requirements for a ground station.

Nevertheless, our setup offers the possibility to create a TCP/IP connection over WiFi if required. We use it for external data logging and debugging, but it can optionally be used for additional external computation or to communicate with other robots.

The whole setup weights slightly less than 2.0 Kg with a 4200 mAH battery. Flight duration is 13 minutes hoovering and 7 minutes for a more dynamic flight. Also, at the expense of flight time, it is able to lift up to 700 g of additional payload.

Our future work is aimed at using the system to adapt and test navigation algorithm which we have developed for omni-directional ground robots. Also we want to exploit the advantages of visual and laser odometry for aerial navigation, because we have no other way to detect movement on the drone the way it is done with ground vehicles, such as counting wheel rotations.

A video [15] of the implemented demonstration program and the source code [5] are available online.

References

[1] Apm-2.6 multicopter control board. *[Online]. http://store.3drobotics.com/products/apm-2-6-kit-1*.

[2] Mpu6000 inertial sensor. *[Online]. http://www.invensense.com/mems/gyro/mpu6050.html*.

[3] Robotic operating system (ros). *[Online]. http://http://www.ros.org*.

[4] Rtf-y6 hexacopter. *[Online]. http://3drobotics.com/kb/diy-y6-kit/*.

[5] Source code (mapir webpage). *[Online]. mapir.isa.uma.es/mapirwebsite/index.php/mobile-robotics/182-drone1*.

[6] Udoo quad. *[Online]. http://shop.udoo.org/eu/product/udoo-quad.html*.

[7] Markus Achtelik, Abraham Bachrach, Ruijie He, Samuel Prentice, and Nicholas Roy. Autonomous navigation and exploration of a quadrotor helicopter in gps-denied indoor environments. In *First Symposium on Indoor Flight*, 2009.

[8] Markus Achtelik, Tianguang Zhang, Kolja Kuhnlenz, and Martin Buss. Visual tracking and control of a quadcopter using a stereo camera system and inertial sensors. In *IEEE International Conference on Mechatronics and Automation, ICMA 2009.*, pages 2863–2869.

[9] Ludovic Apvrille, Jean-Luc Dugelay, and Benjamin Ranft. Indoor autonomous navigation of low-cost mavs using landmarks and 3d perception. *Proc. Ocean and Coastal Observation, Sensors and Observing Systems*, 2013.

[10] Jose Luis Blanco, Juan Antonio Fernandez-Madrigal, and Javier González. A novel measure of uncertainty for mobile robot slam with rao—blackwellized particle filters. *The International Journal of Robotics Research*, 27(1):73–89, 2008.

[11] Jose Luis Blanco, Cipriano Galindo, Javier Gonzalez Monroy, and Javier Gonzalez-Jimenez. Open mobile robot architecture (openmora). *[Online]. http://www.mapir.isa.uma.es/openmora*.

[12] Jose Luis Blanco, Javier González, and Juan Antonio Fernández-Madrigal. Extending obstacle avoidance methods through multiple parameter-space transformations. *Autonomous Robots*, 24(1):29–48, 2008.

[13] Roland Brockers, Martin Hummenberger, Stephan Weiss, and Larry Matthies. Towards autonomous navigation of miniature uav. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 645–651, 2014.

[14] Matthew G Earl and Raffaello D'Andrea. Real-time attitude estimation techniques applied to a four rotor helicopter. In *43rd IEEE Conference on Decision and Control*, 2004.

[15] Andres Gongora and Javier Gonzalez-Jimenez. Demonstration video for enhancement of a commercial multicopter for research in autonomous navigation (youtube). *[Online]. https://www.youtube.com/watch?v=tq9NQ5rQ85Q*.

[16] Javier Gonzalez, Anthony Stentz, and Anibal Ollero. A mobile robot iconic position estimator using a radial laser scanner. *Journal of Intelligent and Robotic Systems*, 13(2):161–179, 1995.

[17] Slawomir Grzonka, Giorgio Grisetti, and Wolfram Burgard. A fully autonomous indoor quadrotor. *IEEE Transactions on Robotics*, 28(1):90–100, 2012.

[18] Mariano Jaimez-Tarifa, Javier González-Jiménez, and Jose Luis Blanco. Efficient reactive navigation with exact collision determination for 3d robot shapes. *International Journal of Advanced Robotic Systems*, 2015.

[19] Sebastian Klose, Jian Wang, Michael Achtelik, Giorgio Panin, Florian Holzapfel, and Alois Knoll. Markerless, vision-assisted flight control of a quadrocopter. In *International Conference on Intelligent Robots and Systems*, pages 5712–5717, 2010.

[20] Robert Mahony, Tarek Hamel, and Jean-Michel Pflimlin. Nonlinear complementary filters on the special orthogonal group. *IEEE Transactions on Automatic Control*, 53(5):1203–1218, 2008.

[21] Antonio J Muñoz and Javier Gonzalez. Two-dimensional landmark-based position estimation from a single image. In *IEEE International Conference on Robotics and Automation, 1998. Proceedings. 1998*, volume 4, pages 3709–3714, 1998.

[22] Teodor Tomic, Korbinian Schmid, Philipp Lutz, Andreas Domel, Michael Kassecker, Elmar Mair, Iris Lynne Grixa, Felix Ruess, Michael Suppa, and Darius Burschka. Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue. *IEEE Robotics & Automation Magazine*, 19(3):46–56, 2012.