International Journal of Current Engineering and Technology
E-ISSN 2277 – 4106, P-ISSN 2347 - 5161

Available at http://inpressco.com/category/ijcet

Research Article

Quadrotor Flight Control Software Development

Abubakar Surajo Imam[Å*] and Robert Bicker[Å]

[Å]School of Mechanical and System Engineering, Newcastle University, United Kingdom

Accepted 15 February 2014, Available online 20 February 2014, **Vol.4, No.1 (February 2014)**

Abstract

This paper presents a comprehensive development of a flight control software to facilitate autonomous and semi-autonomous control of a quadrotor. The software system consists of two parts, namely, the on-board flight control software and the user control interface. The software systems for the quadrotor perform tasks such as (i) hardware driving, (ii) input-output control law implementation, (iii) device-operation management, (iv) multiple-task scheduling, and (v) event management. The on-board flight control software is developed using a graphical programming software integrated development environment (IDE), namely Flowcode, developed by Matrix Multimedia. A behaviour-based architecture was adopted in the design of the autonomous flight control software, in which the operation of the quadrotor is organized in a variety of behaviours. A hierarchical and modularized structure is used to execute these behaviours and to integrate multiple control algorithms. The user control interface software was developed using the Processing language, and runs on Windows-based PC. The user control interface is equipped with a wireless modem for communicating with the on-board software system. The performance of the quadrotor flight control software was implemented, and a satisfactory result was obtained.

Keywords: *Quadrotor, Flight Control Software, Graphical Use Control Interface, Flowcode, Processing Language.*

1. Introduction

A quadrotor is a small responsive four-rotor rotorcraft controlled by the rotational speed of its rotors; it is compact in design with the ability to carry a high payload. Nowadays, the use of a quadrotor as a research platform is increasing within the academic research circle (Jun and Yuntang, 2011; Senkul and Altug, 2013; Wang *et al*., 2013; Jiang *et al*., 2013), due to the numerous advantages it offers; such as simplicity in design, size and cost. When fitted with robust flight controller and sensors, a quadrotor can be deployed to perform a number of tasks including aerial surveillance, supervision of natural disasters, detection and location of objects and persons.

Quadrotor flight control software performs tasks such as (i) hardware driving, (ii) input-output control law implementation, (iii) device-operation management, (iv) multiple-task scheduling, and (v) event management. A number of architectures for the development of flight control system software of small-scale aircraft have been reported in numerous studies. For instance, an on-board flight control software system for an autonomous helicopter has been demonstrated at the Institute for Computer Systems (ICS) at ETH Zurich (Kottmann, 1999), where on-board tasks such as (i) communications, (ii) data logging, and (iii) control, are described. In (Wills *et al*., 2001), the common object request broker architecture (CORBA) has been used to demonstrate the open control platform (OCP) with reconfigurability and interoperability for complex dynamic systems on a small-scale helicopter. A general application software framework for unmanned air vehicle (UAV) airborne computer based on the VxWorks real-time operating system was presented in (Zheng *et al*., 2009) with detailed designs of task classification, priorities and inter-task communications, where the schedulability analysis was implemented using the rate monotonic (RM) scheduling algorithm. Similarly, (Wu *et al*., 2011) reported the design of a flight control software based on logical and data view architecture for an unmanned helicopter flight control system which facilitated stable and reliable feather of the experimented vehicle. A simple flight control software has been designed in a project involving upgrading of an advanced toy radio-controlled helicopter to an unmanned aerial vehicle, where a miniature PC-104 computer system fitted with MEMS navigation and inertial measurement unit was used as the on-board flight controller (Cai *et al*., 2005). In this research, the procedure reported in (Dong *et al*., 2009) was adopted to develop a flight control software.

The software system consists of two parts, namely the on-board software system and control user interface. The on-board software system was developed using a graphical programming software integrated development environment (IDE), namely Flowcode, developed by Matrix Multimedia (Matrixmultimedia, 2014), which provides reliable support for use on robotics applications. Whilst the user control interface was developed using the

*Corresponding author: **Abubakar Surajo Imam**

Processing language IDE (Processing, 2013) and hosted on a Sony convertible tablet PC equipped with XBee wireless modem, which receives data from and sends commands to the quadrotor on-board flight controller. The interface provides a graphical layout comprising numerous buttons which facilitate the vehicle's control in both autonomous and semi-autonomous mode. A behaviour-based architecture was adopted in the design of the flight control software, in which the operation of the quadrotor is organized in a variety of behaviours, where the flight control laws are directly implemented on the on-board system. During flight, the avionic system, referred here as the NAVSENSOR module collects measurement signal of the current state of the vehicle and the output signal of the module generates control signals to drive the vehicle rotors. The on-board software system is capable of performing multiple tasks including hardware driving, device management, autonomous control, communications and data logging. It can also be pre-programmed to perform numerous autonomous flight manoeuvres such as hover, climb, forward flight, etc.

2. On-board Flight Control Software

Figure 7.1 depicts the framework of the flight control software for the quadrotor, in which the on-board flight control software portion consists of several blocks; each corresponds to a specific task.

2.1 NAV Block

The NAV block interacts with the NAVSENSOR module to retrieve the measured in-flight data and estimates the unmeasured flight states, which are essential for the autonomous control. The NAVSENSOR provides real-time measurements on (i) acceleration, (ii) angular velocity, (iii) magnetic field, (iv) geodetic position (in longitude, latitude, and altitude), among others, at a sampling rate of 50 Hz. 2.

2.2 AUTO Block

The AUTO block is for realizing the autonomous flight control of the vehicle, its main functions include obtaining the quadrotor flight status from globally shared data generated by the NAV block, executing control algorithms based on the flight status, and generating control signals to drive the vehicle's rotors.

2.3 SERVO Block

The SERVO block is for driving the quadrotor rotors; it also records the data of the throttle and manual/automatic switching channels for easy post-flight analysis.

2.4 COMM Block

The COMM block is for the communications between the avionic system and the user control interface through the wireless network, which perform necessary downloading of the in-flight data and uploading of the user-defined command/trajectory.

2.5 DATALOG Block

The DATLOG block is for real-time data logging, which saves all necessary in-flight data.

2.6 MAIN Block

The MAIN block is for managing all the tasks mentioned above.

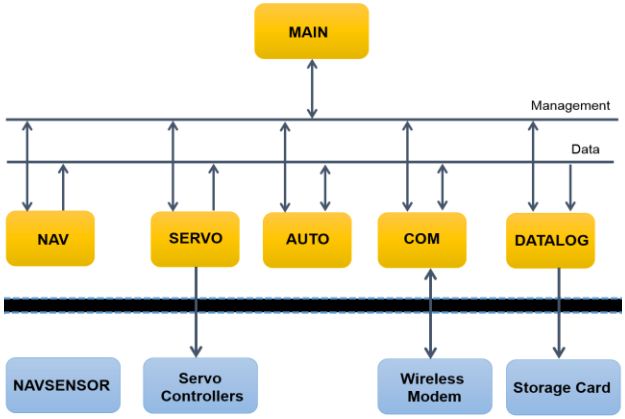

Fig. 1 Framework of the flight control software system.

3. Task Management

In order to schedule the tasks fairly, a round-robin scheduling algorithm is adopted in the flight control software system, where every task is designed to run for a specific period of time. Round-robin scheduling employs time-sharing, giving each task a time slot (its allowance of CPU time), and interrupts the task if it is not completed at the end of its assigned time, and resumes next time a time slot is assigned to that process. To ensure their reliable and harmonious execution, Fig. 2 depicts the management scheme for the tasks scheduling, individual task execution, and time allocation.

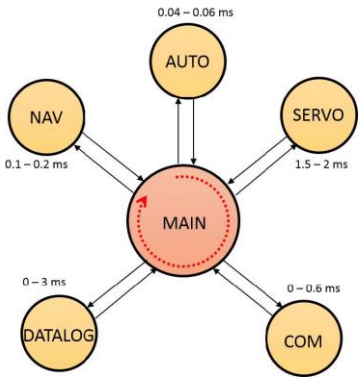

Fig. 2 Task scheduling of the flight control software system.

The task scheduling is performed by the MAIN task, where the dotted round arrow in the MAIN thread denotes the execution sequence of the other tasks. The individual tasks execution follow different ways, for the MAIN task,

its execution scheme is as in the program code, and has two key components, i.e., a timer and program loop. To ensure accuracy, a high-precision timer that provides nanosecond-level accuracy and a variety of system functions was used. It emits a pulse signal to activate the sequential loop of the MAIN task at a frequency of 50 Hz (a sampling rate of 20ms).

The MAIN program loops continuously sending pulse signals to activate the other tasks sequentially for performing their functions and waits for the corresponding accomplishment notification signals, in accordance with the pre-scheduled sequence. The execution of the tasks is carried out as follows: at the beginning of the task loop, it waits for a pulse signal from the operator via the user control interface. Once an execution signal is received, all steps in the task are sequentially executed. The task loop is terminated when an exit pulse issued by the MAIN task is received.

The description of the software processing is depicted by the stack on the right side of Fig. 3 along with the time-consumption list for the involved tasks. The figure gives an illustration of the time allocations for the flight control software; the stack on the left describes the software processing in every cycle, which totally consumes about 5.86ms.

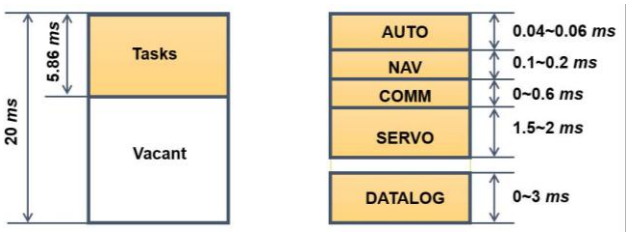

Fig. 3 Tasks time allocation.

4. On-board Flight Software Development

The on-board flight control software consists of two parts, i.e., hardware and software. The software was developed in the Windows 7 Professional operating system environment using Flowcode graphical programming IDE, developed by Matrix Multimedia and illustrated in (Imam, 2012). The hardware used include Arduino Mega microcontroller board (Arduino, 2013), and XBee wireless radio (XBee-PRO 900MHz, 2013). The control algorithm comprises a main task and four other tasks, which the Main task called in every loop cycle and run for their allotted time as detailed in the previous section. The algorithm is ported to the Arduino board and communicates with user control interface running on a PC via the XBee wireless network.

4.1 Main Task Algorithm

Upon powering the flight control system and prior the commencement of the MAIN task loop, the system goes through initializations and configurations settings of the hardware devices, during which time it performs the following: establishes communication with the remote wireless radio on the user control interface device; enables the capture/compare module of the Arduino microcontroller for generation of the PWMs; initializes the PWM duty for the rotors speed control, among others. Figure 4 depicts the Flowcode algorithm that implements the initialization and configuration settings.

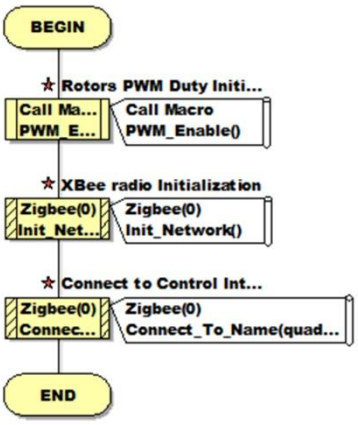

Fig. 4 The initialization and configuration settings Flowcode algorithm structure

During each cycle of the MAIN loop, it checks and processes the incoming command signals from the user control interface and calls the appropriate function responsible for handling the command to send the required control signal reference to the on-board flight controller. The received command signals from the user control interface are coded (e.g. '1', '2', '30', etc), and could mean any of the following: data-log flight information to the SD card, setting PWM duty cycle to the rotors, activate camera, or enable autonomous flight, etc. The structure of the main task algorithm is depicted in Fig. 5.

4.2 AUTO Task Algorithm

This task does not deal with any hardware component; its

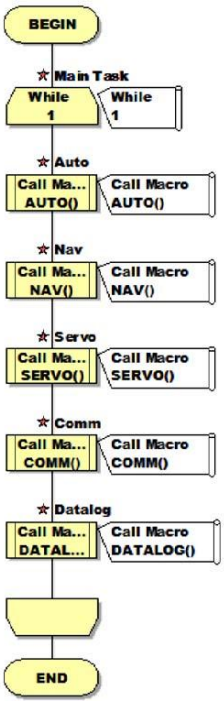

Fig. 5 The MAIN task algorithm structure.

time consumption totally depends on the complexity of the algorithms employed for the flight control system. For the quadrotor, it is generally within the range of 0.04-0.06ms. When this function is called in the main task, it enables the autonomous flight capability of the quadrotor based on the pre-stored flight mission. The function has other sub-functions that represents different flight missions/manoeuvres (Fig. 6), that can be implemented when selected. Each sub-function comprises the requisite rotors PWM signal references, position and orientation references, etc., for accomplishing a desired mission/manoeuvre.

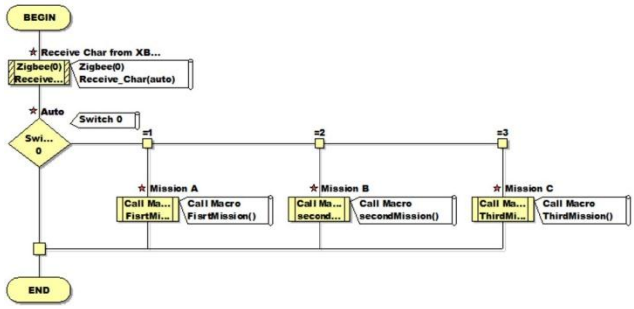

Fig. 6 The AUTO task algorithm structure.

4.3 SERVO Task Algorithm

This task time consumption comes from sending control signal (PWM) to drive the four vehicle's rotors every 16ms. The overall procedure for sending the control signal requires an average time of 1.5-2ms. Figure 7 depicts the algorithm structure of the SERVO task, which when called, updates the PWM duty cycle of the four rotors, to set the rotors speeds. The on-board Arduino microcontroller has a 10- bit PWM duty cycle register, in which approximately a '0' represents zero duty and 1100 represents 100% duty.

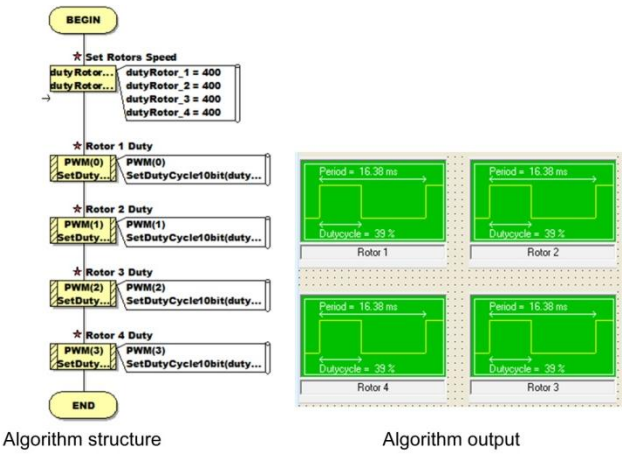

Fig. 7 The SERVO task algorithm structure.

4.4 COMM Task Algorithm

This task is responsible for communication between the on-board flight control software and the user control interface at 9600 baud rate. It has a time consumption of about 0.4 ms, it resides in both the serial read write. The former is for checking and decoding user commands, whereas, the latter covers data transferring to the serial port connected to the on-board wireless radio. When this task is called in the MAIN task, it runs an algorithm which consists of sending of in-flight data, delaying for 500 micro second, and receiving flight command from the remote user control interface. The Flowcode algorithm for this task is shown in Fig. 8.

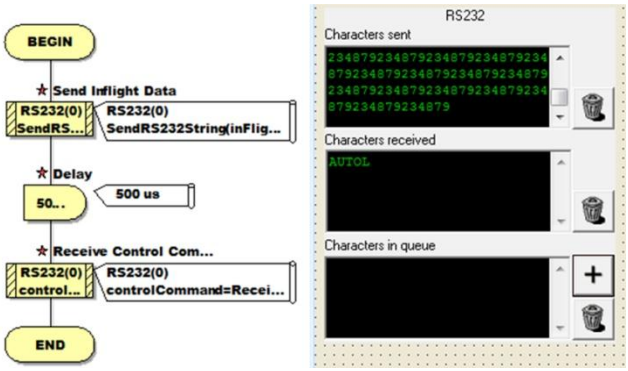

Fig. 8 The COMM task algorithm structure.

4.4 NAV Task Algorithm

This task time consumption consists of two parts, one for retrieving the measurement data from the NAVSENSOR module stored in the serial-port buffer, and other for writing the retrieved data to the serial port. It has been found experimentally, the total time consumption for the NAV task is within the range of 0.1-0.2ms. When this task is called by the MAIN task, it reads the IMU and GPS data, delays for 100 micro second, and then transmits the data via I2C serial port for navigation and telemetry purposes. Figure 9 depicts the Flowcode algorithm that implements the NAV task.

Fig. 9 The NAV task algorithm structure.

4.4 DATALOG Task Algorithm

This task depends on the size of the data package and the speed of the selected storage device. Data can be saved on the on-board microcontroller read-only-memory (ROM) or an external SD card. To improve the efficiency of the

flight control software module, this task is not issued until the data are piled up to a certain predefined volume. The maximum time consumption for task is about 3ms (experimentally established). Anytime this task is called, it executes an algorithm which involves opening of the data file and writing the most current in-flight data to it. The Flowcode algorithm for the implementation of this task is depicted in Fig. 10.

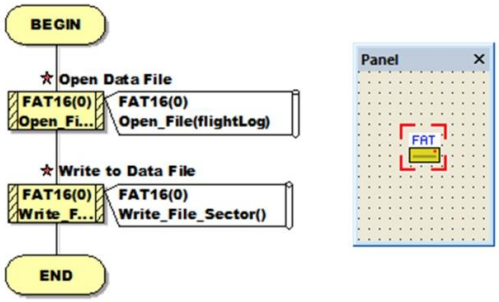

Fig. 10 The DATALOG task algorithm structure.

5. Graphical User Control Interface

This enables command and control of the quadrotor through a wireless communication network a 900 MHz XBee radio network. Flight commands are relayed from the PC hosting the interface to the on-board flight control system. The task of this interface is to provide a friendly and realistic interface for users to control the quadrotor using a standard PCs, tablet PCs, or smartphones. The interface was developed in the Windows 7 Professional operating system using Processing language. It features a graphical view with numerous buttons, which if operated sends control signals to the on-board flight control system, which generates appropriate control signal. The buttons represent commands for achieving various flight manoeuvres, e.g., climbing, descending, rolling, pitching, yawing, hovering and landing manoeuvres. The frame of the control interface is presented in the following subsections.

5.1 Framework of the Control Interface

The framework of the quadrotor graphical user-interface is depicted in Fig. 11.

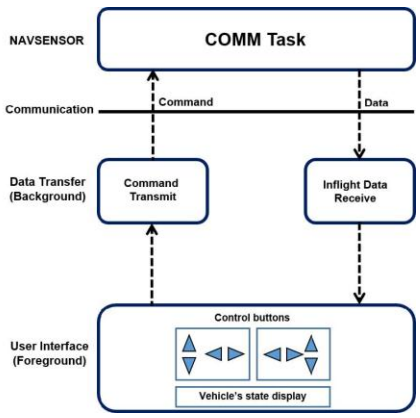

Fig. 11 The framework of the control interface.

It consists of two layers namely foreground and background layers.

The main function of the foreground layer is to provide a friendly graphical user interface (GUI) which allows the user to directly interact with this layer. The main window of this interface contains a group of buttons which facilitates the control of the quadrotor's numerous flight aspects, and can displaying the vehicle's in-flight data. The buttons control the aspects of the vehicle including throttle position; pitch, roll and yaw motions; arming of the motor controller drivers, emergency landing, autonomous flight activation etc. The numerical values of the vehicle in-flight data such as position and velocity can also be viewed.

The background layer handles data transfer. This layer communicates with the software module of the avionic system (i.e., the COMM task) through the wireless communication network. The data-receiving thread in the background layer continuously reads the serial port connected to the wireless modem attached to the device hosting the user interface. The data-transmitting thread of the background layer, on the other hand, keeps checking to capture the user command issued in the foreground layer and then writing it to the serial port.

The Processing programming was used in the development of the above software framework. Processing is an open source programming language and integrated development environment (IDE) built for the electronic arts, new media art, and visual design communities with the purpose of teaching the fundamentals of computer programming in a visual context, and to serve as the foundation for electronic sketchbooks (Ben and Casey, 2010).

5.2 Foreground Layout of the Control Interface

Figure 12 depicts the schematic of the foreground layout of the interface developed using the Processing IDE and Windows OS-enabled PC. Communication between the interface and the on-board flight controller is handled by the Processing serial library via the XBee wireless network.

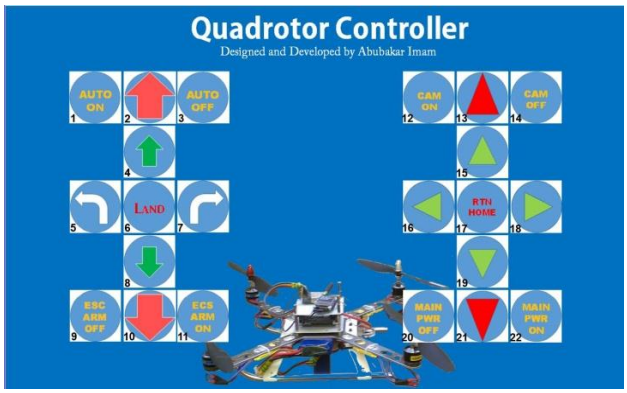

Figure 12 Foreground layer of the control interface.

The foreground layout comprises twenty one buttons in two groups (positioned side-by-side). The left button

group provides provision for the control of the vehicle's autonomous capability, ascend/descend, left/right turn (yaw motion control), and ESCs arming/unarming. Whilst the right button group provides means of controlling the roll and pitch motion of the vehicle, activating the on-board camera, connecting the main power source to the on-board electronics, as well as commanding the vehicle to return home in emergency situations. Table 1 illustrates the functions of these buttons.

Table 1 Function of the foreground buttons

Serial	Button Number	Function
1	1 and 3	Autonomous control ON/OFF
2	4 and 8	Low altitude ascend/descend
3	2 and 10	High altitude ascend/descend
4	5 and 7	Left/right turn
5	6	Emergency landing
6	9 and 11	ESCs arming/unarming
7	12 and 14	On-board camera ON/OFF
8	15 and 19	Pitch ±5 degree
9	13 and 21	Pitch ±15 degree
10	16 and 18	Roll ±5 degree
11	17	Return home
12	20 and 22	Power ON/OFF

To demonstrate how the buttons were generated and how they function, the following code listing depicts the portion of the Processing code for the AUTO button.

```
(1) import processing.serial.*;
(2) int[ ] controls = new int[ ] {1, 2, 3, 4, 5, 6, 7, 8, 9, . . .42};

    (3) ImageButtons autoOn;
    (4) ImageButtons autoOff;
    (5) PImage bg;
    (6) PFont myFont;

(7) void controlGUI() {

    (8) bg = loadImage("QuadBG.png");
    (9) background(bg);
    (10) int buttonSize = 100;
    (11) PImage selected = loadImage("ControlSelect.png");

(12) PImage autoOnImage = loadImage("autoOn.png");

    (13) PImage  autoOn2Over = loadImage("autoOn2.png");
    (14) autoOn = new ImageButtons(143, 150, buttonSize, buttonSize,
         autoOnImage, autoOn2Over, selected);


(15) PImage autoOffImage = loadImage("autoOff.png");

    (16) PImage NlightOff2Over = loadImage("autoOff2.png");
    (17) autoOff = new ImageButtons(385, 150, buttonSize, buttonSize,
         autoOffImage, autoOff2Over, selected);
    }

(18) void setup()  {

    (19) String XBeePort = Serial.list()[0];
    (20) port = new Serial(this, "COM2", 9600);
    (21) size(1489, 863);
    (22) smooth();
    (23) controlGUI();
    }

(24) void updateDisplay() {

    (25) autoOn.update();
    (26) autoOn.display();
    (27) autoOff.update();
    (28) autoOff.display();

}
(29) void sendCommands() {

    (30) if(autoOn.pressed) {
         (31) port.write(controls[0]);
    }
    (32) if(autoOff.pressed) {
         (33) port.write(controls[2]);

}
(34) void draw() {

    (35) updateDisplay();
    (36)sendCommands();

}}
```

5.2 Emergency Handling

A safety measure has been built in the control system to handle emergencies which may arise due to system(s) failure. Numerous factors could lead to the flight control system failures, amongst which are drastic environmental changes, hardware failure, GPS disorder, and problems in the control system software implementation. For this reason, a mechanism for handling emergency situations is built in the on-board software system. On every cycle, before applying the control action, the control task thread checks all data received from the NAVSENSOR and other devices for abnormality. Once any abnormality is observed, the emergency control function is called up immediately to sequentially (i) send an alert signal to the ground control station to inform the operator to take over the control authority, (ii) drive and maintain all four input channels to their trimmed values in the hovering condition, and (iii) abort the running task and execute landing algorithm. The mechanism also logs-in flight information during emergency situations, which can be used later to identify the cause of the incident.

6. Autonomous Flight Control Implementation

The implementation of the autonomous control laws in the AUTO task is one of the most essential issues in the design of the flight control software module for the quadrotor. A comprehensive flight plan generally consists of multiple flight patterns or behaviours, which commonly have different control parameter settings and even require different control algorithms. These issues have been carefully considered in the design of the flight control software development procedure in this study. A behaviour-based flight scheduling block was employed, which organizes various behaviours in a flight plan. In the architecture, any operation of the quadrotor is recognized as a specific behaviour with specific parameters (such as control signal references and execution time limitations). A hierarchical control system block is created to receive the behaviour and realize the corresponding autonomous control algorithm. It consists of two layers with multiple control algorithms being integrated. The following provides detailed descriptions of the flight scheduling and control system blocks with an example to demonstrate its implementation procedure.

6.1 Flight Scheduling Block

The flight scheduling block manages and coordinates flight plans for the quadrotor; a specific flight plan can be either pre-stored in the avionic system or generated online from the ground control station.

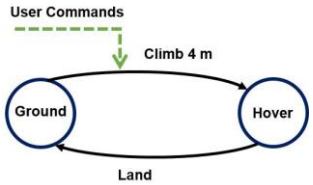

Fig. 13 Structure of the hierarchical flight control system.

Figure 13 depicts a simple flight plan, in which the quadrotor is required to start from ground, climbs to 4 m, hovers for 2 minutes and descends to the ground.

It can be noted that the flight plan consists of nodes and arrows. Each node represents a sub-plan or a specific behaviour, whereas each arrow denotes an event. The flight scheduling block employs an event-driven mechanism, when an event is triggered, the flight scheduling block transfers from one behaviour (or sub-plan) to another. An event can be triggered by either one of the following sources: (i) change on the surrounding environment (e.g., high wind), (ii) change in working situations (e.g., hardware component malfunction and data loss), (iii) change in in-flight status (e.g., the achievement of a certain flight mission), or (iv) user command uploaded from the ground control station. Depending on the triggered source, the flight scheduling block determines what behaviour is to be performed and transmits it to the consequent control system block for further action. It is clear that the flight scheduling block acts like a decision maker or a commander.

6.2 Flight Control System Block

The function of the flight control system block is to decode the behaviours received and to drive the rotors of the quadrotor. The control system block is developed based on a hierarchical structure (Fig. 14), and its structure is based on the quadrotor flight control system reported in (Imam and Bicker, 2014).

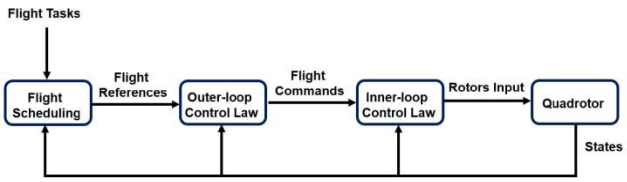

Fig. 14 Control system block.

The inner loop guarantees the asymptotic stability and good disturbance-rejection capacity of the quadrotor, and the outer loop follows the behaviour request to guide the vehicle for achieving a predefined flight trajectory with a desired orientation. The outer loop is responsible for generating control references for the inner loop. The two layers work together to realize the vehicle's autonomous control.

6.3 AUTO Task Implementation

The following gives a specific example which illustrates the implementation of the AUTO task on the quadrotor system (Fig. 15) developed in (Imam and Bicker, 2014a; Imam and Bicker, 2014b). The AUTO task is implemented on this quadrotor as follows:

1) The triggering execution commands are pre-programmed on to the flight control system, and the quadrotor was set to a stable hover.
2) After receiving the command, the quadrotor performed a forward flight with a maximum speed of 5m/s to a destination 50m away from the starting position. Upon reaching the destination, the vehicle hovered for 10s and then made a 180 degree head turning motion with a constant yaw rate, and maintained the hovering manoeuvre at the destination for 10 s, with its front pointing to the starting point.
3) The next behaviour, triggered after the completion of the hovering manoeuvre was flight back to the starting point.
4) The overall flight plan ended again with a stable hover at the starting point.

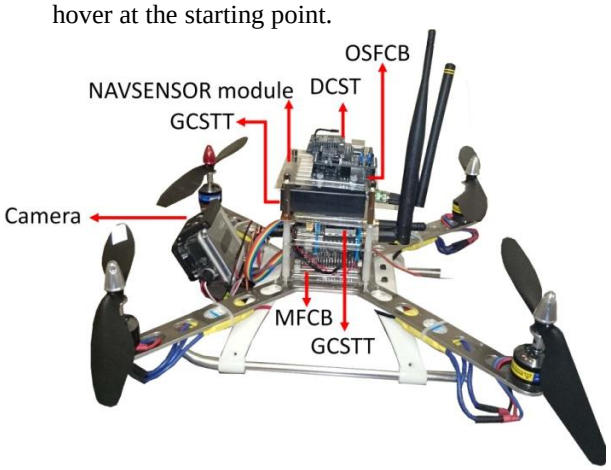

Fig. 15 Quadrotor system.

The specific behaviour set consists of Hover, Forward Flight and Head Turn. Assuming that there is no obstacle between the starting and destination points, the behaviour sequence shown in Fig. 16 is obtained, which is sequentially passed to the control system block and to generate the flight pattern shown in Fig.17.

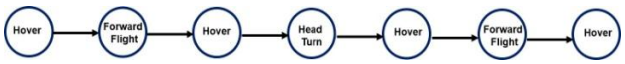

Fig. 16 AUTO task implementation

For a specified behaviour, information transferred from the flight-scheduling block to the control system block consists of two parts, necessary control reference signals (e.g., position and velocity references) and flag signals for activating control algorithms. In this example, the flags are set to load the PID/MPC inner-loop and outer-loop control laws into the control system block, and the corresponding control gain matrices are then retrieved from the globally shared memory for generating the driving signals of the servo actuators.

Conclusions

This paper has presented the development of the flight control software, which facilitates the control of the quadrotor in both semi-autonomous and autonomous modes. It comprises two modules namely, the on-board flight control software and user control interface. The on-board flight control software was developed using the Flowcode graphical programming software and ported to Arduino microcontroller board. A behaviour-based

architecture was used in the development of the on-board flight control system, in which the operation of the quadrotor was organized in a variety of behaviours. A hierarchical and modularized structure was used to execute these behaviours, and to integrate multiple control algorithms. Whilst, the user control interface was developed using the Processing language and hosted on a commercial operating system, Windows 7 Professional. It enables command and control of the quadrotor through a wireless communication channel, namely, 900 MHz XBee radio network. Flight commands are relayed from the software user interface to the on-board control system via XBee wireless network. The task of the software user control interface is to provide a friendly and realistic graphical user interface (GUI) for users to control the quadrotor using a Window-based PC or a smartphone. The performance of flight control software was evaluated, where the autonomous flight capability of a quadrotor was tested by implementing the AUTO task (presented in Subsection 6.3), and a satisfactory result was obtained.

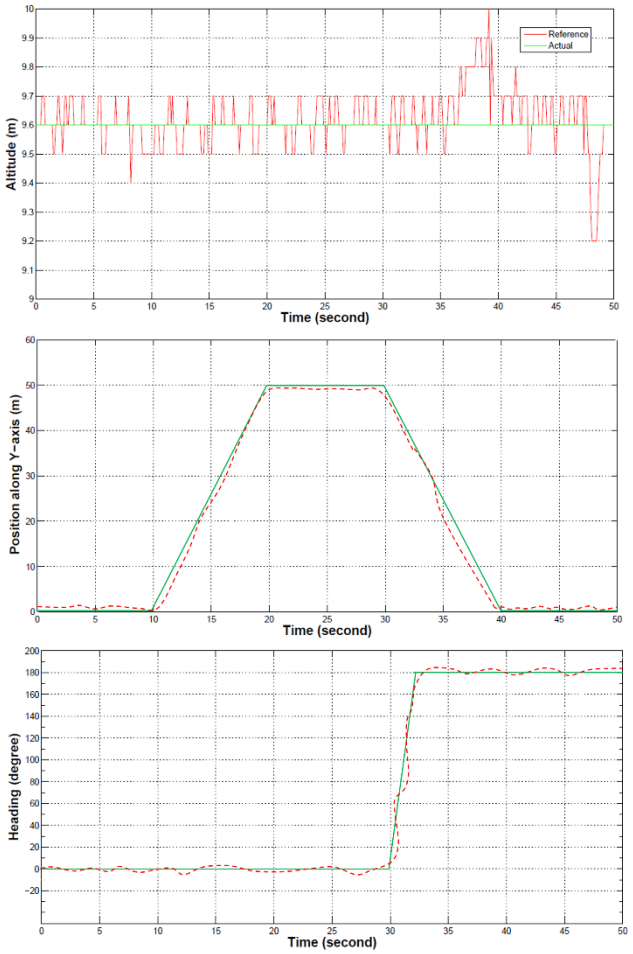

Fig. 17 AUTO task implementation result.

References

Arduino (2013), 'Arduino mega 2560', http://arduino.cc/en/Main/ArduinoBoardMega2560.

Ben, F. & Casey, R. (2010), Getting Started with Processing: A Hands-on Introduction to Making Interactive Graphics.

Cai, G., Peng, K., Chen, B. M. & Lee, T. H. (2005), Design and assembling of a uav helicopter system, in 'proceedings of the International Conference on Control and Automation (ICCA), 2005', Vol. 2, pp. 697–702.

Dong, X., Chen, B. M., Cai, G., Lin, H. & Lee, T. H. (2009), Development of a comprehensive software system for implementing cooperative control of multiple unmanned aerial vehicles, in 'poceedings of the IEEE International Conference on Control and Automation (ICCA), 2009', pp. 1629–1634.

Imam, A. S. (2012), Mechatronics for Beginners: 21 Projects for PIC Microcontrollers, Author House United Kingdom.

Imam, A. S. & Bicker, R. (2014a), 'Design and construction of a small-scale rotorcraft uav system', International Journal of Engineering Science and Innovative Technology (IJESIT) 3, 96–109.

Imam, A. S. & Bicker, R. (2014b), 'Quadrotor comprehensive identification from frequency responses', International Journal of Scientific and Engineering Research (IJSER) 5, in Press.

Jiang, J., Qi, J., Song, D. & Han, J. (2013), Control platform design and experiment of a quadrotor, in 'proceedings of the 32nd Chinese Control Conference (CCC), 2013', pp. 2974–2979.

Jun, L. & Yuntang, L. (2011), Dynamic analysis and pid control for a quadrotor, in 'proceedings of the IEEE International Conference on Mechatronics and Automation, (China), pp. 573-578'.

Kottmann, M. (1999), Software for model helicopter flight control, Technical report, Eidgenossisches Technische Hochschule Zurich.

Matrixmultimedia (2014), 'Flowcode 5 electronic systems design software', http://www.matrixmultimedia.com/. Processing (2013), http://processing.org/.

Senkul, F. & Altug, E. (2013), Modeling and control of a novel tilt-roll rotor quadrotor uav, in 'Unmanned Aircraft Systems (ICUAS), 2013 International Conference on', pp. 1071–1076.

Wang, F., Xian, B., Huang, G. & Zhao, B. (2013), Autonomous hovering control for a quadrotor unmanned aerial vehicle, in 'proceedings of the Control Conference (CCC), 2013', pp. 620–625.

Wills, L., Kannan, S., Sander, S., Guler, M., Heck, B., Prasad, J., Schrage, D. & Vachtsevanos, G. (2001), 'An open platform for reconfigurable control', Control Systems, IEEE 21(3), 49 –64.

Wu, Y., Hu, F. & An, J. (2011), Design and implementation of flight control system software for unmanned helicopter, in 'proceedings of the International Conference on Computer Science and Network Technology (ICCSNT), 2011', Vol. 4, pp. 2196–2200.

XBee-PRO (2013), http://www.digi.com/xbee/.

Zheng, Z., Zhu, M. & Jiang, G. (2009), Flight control software design for stratospheric aerostat, in 'proceedings of the International Conference on Computational Intelligence and Software Engineering, CiSE 2009', pp. 1–5.