

# Communication and Control Software Development for Experimental Unmanned Aerial System – Selected Issues

Dariusz Rzonca<sup>1</sup>, Sławomir Samolej<sup>1</sup>,  
Dariusz Nowak<sup>2</sup>, and Tomasz Rogalski<sup>2</sup>

<sup>1</sup> Department of Computer and Control Engineering,  
Rzeszow University of Technology,  
al. Powstancow Warszawy 12, 35-959 Rzeszow, Poland  
`drzonca@prz.edu.pl`, `ssamolej@prz.edu.pl`

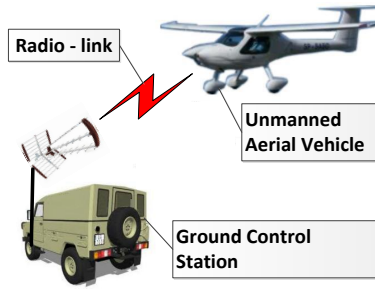
<sup>2</sup> Department of Avionics and Control Systems,  
Rzeszow University of Technology,  
al. Powstancow Warszawy 12, 35-959 Rzeszow, Poland  
`darnow@prz.edu.pl`, `orak1@prz.edu.pl`

**Abstract.** Recently the functionality and applicability of Unmanned Aerial Systems have been dynamically growing. In the paper such an experimental system, based on the "MP-02 Czajka" ultra light plane, is presented. Generation of the C source code for the autopilot firmware in the Matlab environment is discussed. Furthermore, several issues raised during implementation of the STANAG 4586 communication protocol are considered.

**Keywords:** Unmanned Aerial Systems, STANAG 4586, Matlab Embedded Coder, Simulink Coder, source code generation

## 1 Introduction

Unmanned Aerial Systems (UAS) development has become a well defined and dynamically changing area of knowledge [1–3] as well as a growing branch of both Polish national [4, 5] and global [6, 7] industry. Modern UAS consist of an Unmanned Aerial Vehicle (UAV) cooperating with a Ground Control Station (GCS) (comp. Fig. 1). Typically, the UAV mission is to follow a 4D trajectory in an autonomous way when the UAV's payload executes its predefined tasks. GCS should enable to plan, supervise and manually conduct a mission. A GCS operator supervises mission execution and is able to take over the UAV any time to manually correct the flight route or the payload state. Every year UAS have to conduct more complex missions and are gradually becoming the controlled airspace members [8]. This in consequence forces rapid development of communication, control and software components of such systems.



**Fig. 1.** General overview of the Experimental UAS

This paper discusses some selected control engineering related problems solved during an experimental UAS development. The experimental system is a hardware - software platform created for research on advanced control systems for UAS [8,9], including automatic take-off, landing, and autonomous on-airfield navigation.

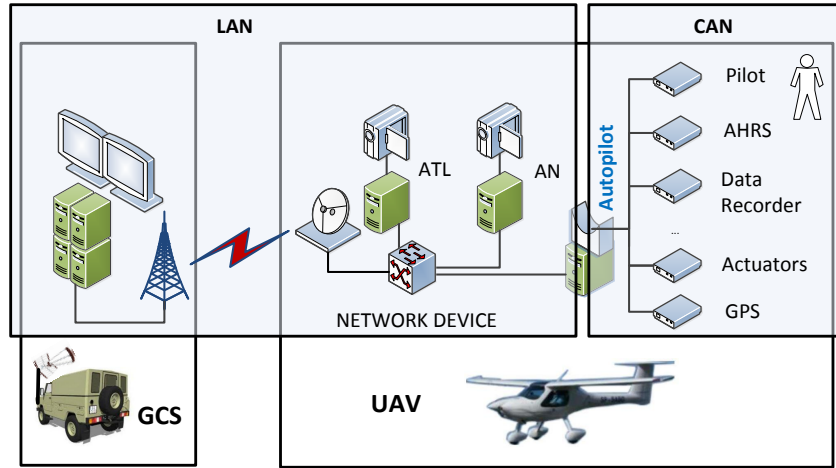
From the control engineering point of view the experimental UAS constitutes a real-time distributed control system integrated by means of a heterogeneous computer network. The software of the system is regularly updated due to research continuation. Simultaneously, the computer network structure evolves as some new on-board devices are introduced. This in consequence enforces a specific integration and the software development approach which will be presented in subsequent sections of the paper. The main subjects raised in this paper are GCS – UAV communication protocol implementation issues and automatic code generation for the UAV's autopilot.

The subsequent subsections of the paper are organised as follows: firstly, the experimental UAS system is presented. The on-going system architecture is adopted for the current research task which involves a development of new automatic take-off and landing procedures as well as on-airfield navigation ones. The new procedures extend the modern UAS operability to commercial airfields and controlled airspace. Secondly, selected inter UAV and GCS communication problems are explained. This implies a discussion on communication software development. The following part describes automatic generation of the C source code for the autopilot firmware in the Matlab environment.

## 2 Experimental UAS Overview

Main components of the experimental UAS consist of the "MP-02 Czajka" [9, 10] ultra light plane and the GCS installed on a car (comp. Fig. 1). The GCS and UAV communicate via a digital data link. All hardware and software components of the UAS, apart from the core flying and driving platform, are original solutions developed by partners of the research programmes mentioned.

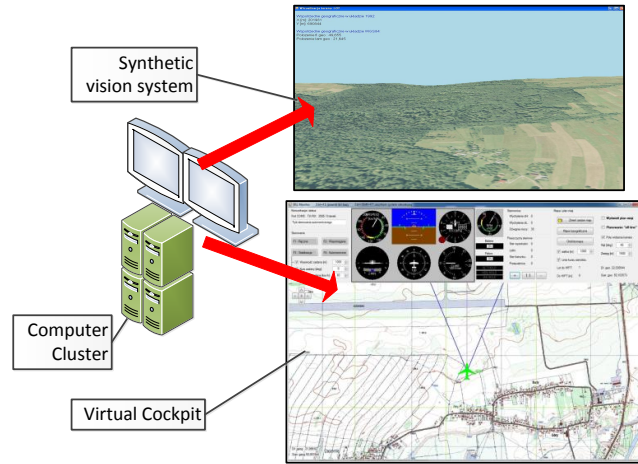
Figure 2 depicts the UAS from the hardware integration perspective. The ultra light plane has been adapted to perform unmanned flights. Its ailerons, flaps,



**Fig. 2.** UAS' Control Hardware Components

elevator and rudder as well as engine can be operated remotely with the aid of digital servos. The in-built *autopilot computer* produces setpoint values for all the servos on the basis of reference signals acquired from an Attitude Heading Reference System (AHRS), GPS, a flight trajectory plan and a GCS, if necessary. Yet, the plane can be still operated by a human pilot located on-board. This makes it possible to partially test the developed control and communication subsystems in the air. The autopilot, AHRS, Human Pilot, Data Recorder, GPS, and servos (actuators) are connected via a local Controlled Area Network (CAN) [11] running CAN Aerospace [12] protocol. At the current system development phase two additional, physically separated digital control subsystems are introduced. They are responsible for automatic take-off and landing (ATL) and on-airfield navigation (AN) of the UAV. The new modules – autopilot and data link, are connected by an IEEE 802.3 (Ethernet) based Local Area Network (LAN). This heterogeneous network structure reflects system evolution. A preliminary on-board network was CAN. Due to system development and incorporation of data-intensive applications a new IEEE 802.3 based network has been introduced on-board.

The GCS is a computer cluster which supports planning, execution and recording of a flight mission. A dedicated graphics interface supports these tasks (comp. Fig. 3). The *mission planner* software unit is a separate non-control program where a flight plan is generated. A flight plan includes a set of waypoints the UAV has to go through as well as a set of manoeuvres the UAV is obliged to do, if necessary, around these points. The *mission executioner* software unit combines different data streams to graphically depict UAV's current state for the operator. It combines video, map database as well as a current aircraft's speed, position, course and orientation. This unit also enables to take over the



**Fig. 3.** GCS Graphics Components

UAV and lets the operator control it manually. Usually, a separate GCS module supports an UAV's payload control (such as on-board camera operation).

The GCS and UAV belong to the same LAN segment. They are connected by a pair of specialised radio modems which enable to set up a VPN connection between these two remote UAS components. The operation range of the UAV is up to 80 km. This solution gives the possibility to set up communication between UAV and GCS as standard TCP/IP or UDP/IP based data streams. At the current system development stage a "standard" IEEE 802.3 network has been applied as a new UAS communication medium. This implies some doubts about data transfer predictability and timekeeping which have been already discussed in [13]. Simultaneously, the new UDP/TCP/IP/Ethernet UAS LAN enables to apply an advanced STANAG 4586 [14] protocol dedicated to sophisticated UAV's management and control. Its implementation will be the subjects of the following sections.

The central UAV control module is the *autopilot*. Its main task is to stabilise the aircraft in the air and let it follow the 4D trajectory during mission execution. It is also the bridge between the on-board CAN and LAN. Finally, it interpreters data streams produced by GCS, ATL and AN, and reports the UAV status to the GCS. As this module has the predominant role for the whole system execution, its software is analysed and generated automatically. One of the subsequent sections will raise the problem in a detailed way.

### 3 GCS to UAV Communication Standard

One of the new trends in UAS development is standardisation of the interface between the GCS and the UAV. The idea comes from the NATO where different UAVs could be controlled by GCSs provided by any ally. This also gives the op-

portunity to split the generation of GCSs and UAVs between separate providers. A significant step towards standardisation was STANAG 4586 [14] publication. This document provides a complete protocol specification that covers most of data exchange that may occur between the GCS and UAV.

UDP Source Port	UDP Destination Port
UDP Packet Length	UDP Checksum
Sequence#	Message Length
Source ID	
Destination ID	
Message Type	Message Properties
Data (Message Payload)	
Optional Checksum	

**Fig. 4.** STANAG 4586 Message Wrapper

STANAG 4586 utilises UDP/IP protocol as a background. All messages have the same wrapper structure as in Fig. 4. The *source* and *destination* IDs reflect individual UAS element or even a dedicated device on-board which sends/receives data. *Message type* is a STANAG 4586 defined message number. Each message has predefined data to transfer. Some messages may require receiver's automatic response. The protocol behaviour depends on the message type and it is possible to derive precise message – respond pattern that must be implemented for proper protocol implementation.

Figure 5 depicts a sample message payload. This particular message reports the current UAV internal state to the GCS. Each data element has its type and interpretation. For example "Integer 3" means that data will be stored in a signed three-byte word, and "Unsigned 5" informs that data will be stored in an unsigned five-byte word. The "Time Stamp" data element enables to latch and transfer the time when the message has been created. The message data should follow most significant byte first order. This implies specific protocol implementation rules: message payload will include untypical (in the length) data elements, where byte order has to be strictly preserved. The current experimental UAS configuration offers all of technical facilities to effectively implement the STANAG 4586 based communication layer.

## 4 STANAG 4586 Communication Library Implementation

Implementation of STANAG 4586 communication library for the described project has been prepared in a way to meet several requirements. The library has to be written in C99 language (ISO/IEC 9899:1999 [15]) for maximum portability. In fact C90 standard would be even more portable, but some features, e.g.

Inique ID	Field	Element name/ descriptopn	Type	Units	Range
4000.00	0	Presence Vector	Unsigned 3	Bitmapped	No Restrictions
0101.01	1	Time Stamp	Unsigned 5	0,001 s	See Section 1.7.2
0101.04	2	Latitude	Integer 4	BAM	$-\pi/2 \leq x \leq \pi/2$
0101.05	3	Longitude	Integer 4	BAM	No Restrictions
0101.06	4	Altitude	Integer 3	0,02 m	$-1,000 \leq x \leq 100,000$
0101.07	5	Altitude Type	Unsigned 1	Enumerated	0 or 1 or 2 or 3
0101.08	6	U_Speed	Integer 2	0,05 m/s	$-1,000 \leq x \leq 1,000$
0101.09	7	V_Speed	Integer 2	0,05 m/s	$-1,000 \leq x \leq 1,000$
0101.10	8	W_Speed	Integer 2	0,05 m/s	$-1,000 \leq x \leq 1,000$
0101.11	9	U_Accel	Integer 2	0,005 m/s <sup>2</sup>	$-100 \leq x \leq 100$
0101.12	10	V_Accel	Integer 2	0,005 m/s <sup>2</sup>	$-100 \leq x \leq 100$
0101.13	11	W_Accel	Integer 2	0,005 m/s <sup>2</sup>	$-100 \leq x \leq 100$
0101.14	12	Roll	Integer 2	BAM	No Restrictions
0101.15	13	Pitch	Integer 2	BAM	$-\pi/2 \leq x \leq \pi/2$
0101.16	14	Heading	Integer 2	BAM	No Restrictions
0101.17	15	Roll Rate	Integer 2	0,005 rad/s	No Restrictions
0101.18	16	Pitch Rate	Integer 2	0,005 rad/s	No Restrictions
0101.19	17	Heading Rate	Integer 2	0,005 rad/s	No Restrictions
0101.20	18	Magnetic Variation	Integer 2	BAM	No Restrictions

**Fig. 5.** STANAG 4586 Message 4000

long long int type, which is necessary, are introduced in C99. The same library is used on different hardware platforms, e.g. ADS512101 board – autopilot (Freescale MPC5121e processor, VxWorks 6.8 operating system) and PC computer – GCS module. It is noteworthy that these platforms differ in endianness, with the ADS512101 board being a big endian (BE) whereas PC is a little endian (LE) machine. Of course the library should work in both cases, and it would be beneficial if the code was endian independent, so no assumption about byte order should be made and direct byte swapping should be avoided. Instead the conversions should base on  $\gg=8$  and  $\ll=8$  bit shifts, which in fact will be optimized by a compiler to correct byte transfers, regardless of target endianness. An example of the converting macro using byte shifts is shown below.

```
#define stanag_set_msb(pointer, val, size) signed char i; \
for (i = size-1; i >= 0; i--) \
{ \
*((unsigned char *)pointer+i) = val & 0xFF; \
val >>= 8; \
}
```

As already mentioned, the STANAG 4586 communication protocol also introduces variables which are three or five bytes long. Such variables should be converted by the communication library to the smallest types in C which are able to represent them, i.e. `long int` and `long long int`. For such purpose appropriate types representing STANAG 4586 have been defined (e.g. `typedef unsigned long stanag_u3;` etc.), together with constants of their sizes (e.g. `#define stanag_u3_size 3` etc.). Unfortunately such conversions between variables of different sizes, as well as possible difference in endianness, exclude the possibility of treating STANAG 4586 frame as `struct` in C. Instead it has to be implemented as an array of unsigned chars, and dedicated functions have to be prepared for conversion (serialization) of the variables from C types to the STANAG 4586 frame, and vice versa. Of course in such case the bodies of several convert functions are very similar, differing mainly in handled data type. The "don't repeat yourself" (DRY) principle of software development aims at reducing unnecessary repetitions, so similar parts of code should be avoided. In this case it is not trivial, because C language lacks sophisticated features like templates or polymorphism, but similar functionality can be achieved using macros with `##` symbol, that will produce different conversion functions automatically, basing on a single piece of code. An exemplary source code of such a macro is provided below.

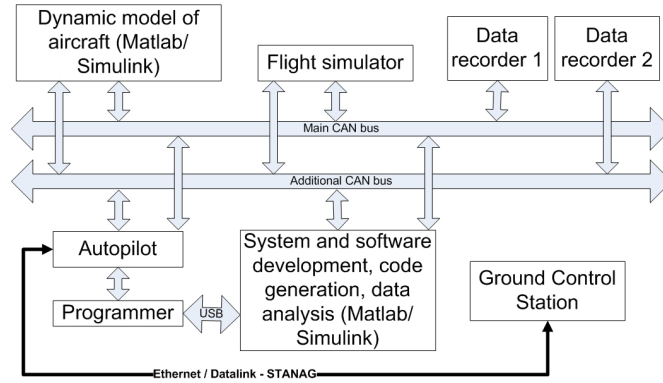
```
#define MAKE_SET_MSB_FROM(x) \
void set_msb_from_##x(void * where, stanag_##x value) \
{ \
    stanag_set_msb(where, value, stanag_##x##_size); \
}
```

The macro `MAKE_SET_MSB_FROM` is called multiple times with different parameters to create several conversion functions. Example call `MAKE_SET_MSB_FROM(u3)` will produce `void set_msb_from_u3(void *, stanag_u3)` function for conversion from `stanag_u3` variable stored as `unsigned long` in any endian to an array of three bytes in big endian (most significant byte first). Other set functions, as well as get functions for vice versa conversions, are implemented similarly.

For continuous testing of the implemented code for regression errors during library development stage, `assert` macros have been used for runtime checks of the converted values. The library has been tested both on the ADS512101 board, as well as on a PC. Final compilation of the embedded and desktop applications have been performed with `NDEBUG` defined, so the assertions were removed.

## 5 Flight control system designing – code generation

Simulation testing is an essential stage in the design process of flight control and navigation systems for unmanned aerial vehicles. It allows for early identification of possible errors that may occur at both the hardware level and the software level of every part of the system. A very useful tool in the design process of a flight control system is a software-in-the-loop simulation testbed (Fig. 6).



**Fig. 6.** The simulation testbed structure

The test station consists of computer hardware equipped with specialized software, such as a simulation environment (flight simulator), applications for flight parameters recording and analysis, computational and simulation software for designing control and navigation algorithms, development tools for the source code generation, its compilation and implementation in memory of CPU [16]. The main component of the testbed is the on-board control computer (*autopilot*) that communicates with the ground control station subsystem by using datalink and STANAG 4586 protocol. This computer has the identical configuration as the one installed in the UAS (comp. fig. 2). Data exchange between another components is performed via CAN bus. The simulation environment contains the dynamic model of MP-02A aircraft and its actuators. Thanks to source code generation technology, the testbed allows to design flight control and navigation algorithms, to verify instantly their correctness, and to prepare control computers for tests in flight [17].

Matlab/Simulink software modules such as Embedded Coder and Simulink Coder have been applied for autopilot's control algorithm code generation. These development tools are equipped with user friendly interfaces that allow to design control algorithms with the use of block diagrams and clear forms. Expanded block diagrams of algorithms designed by using Simulink have been converted into a source code in C programming language. The source code is split into several files that can be attached to the project, compiled by C language compiler, and finally stored in flash memory of control computer's CPU.

The control computer software is modular and composed of elements responsible for receiving and processing data according to specific control laws and navigation algorithms developed and generated with the use of Matlab/Simulink (Fig. 7). The main component of the control system is a module called *mission controller*, which can work in one of three modes, follow a mission plan on the basis of flight parameters and data from ground control station. Control system unit represented by the block called *navigation* includes a set of navigation features. They are used to calculate the desired heading and desired altitude based





## 6 Conclusions

The development of the experimental Unmanned Aerial System has been described and several selected issues were thoroughly discussed. Implementation of the STANAG 4586 communication standard increases interoperability and flexibility. The UAV also benefits from automatic generation of several parts of autopilot control software in the Matlab environment.

## References

1. Lozano R. (Editor), *Unmanned Aerial Vehicles: Embedded Control*, Wiley-ISTE, (2010).
2. Valavanis K. P. (Editor): *Advances in Unmanned Aerial Vehicles, State of the Art and the Road to Autonomy*, Springer, (2007).
3. Lam T. M. (Editor): *Aerial Vehicles*, In-Tech, (2009).
4. UAS producer website <http://www.eurotech.com.pl/produkty-i-uslugi/2/lotnictwo-i-awionika>, (2017).
5. UAS producer website <http://www.wb.com.pl/Rozwiazania,Systemy-C4ISR,Systemy-rozpoznania.html>, (2017).
6. UAS producer website <http://www.avinc.com/uas/>, (2017).
7. UAS producer website <http://www.ga-asi.com/products/aircraft/>, (2017).
8. European ERA GRANT Webpage <http://www.era-research-project.org/>, (2017).
9. Nowak D., Rogalski T., Walek L.: System lot jako latajace laboratorium, *Technika Transportu Szybowego* 12/2015, pp. 1122–1126, (2015), (in Polish).
10. MP-02 Czajka ultralight aircraft producer website <http://www.mp-02.pl/>, (2017).
11. Natale M. D., Zeng H., Giusto P., Ghosal A.: *Understanding and Using the Controller Area Network Communication Protocol, Theory and Practice*, Springer Science+Business Media, (2012).
12. CAN Aerospace, Interface specification for airborne CAN applications V 1.7, [http://www.canaerospace.net/tl\\_files/downloads/canaerospace/canas.17.pdf](http://www.canaerospace.net/tl_files/downloads/canaerospace/canas.17.pdf), (2017).
13. Samolej S., Rogalski T., Rzońca D.: Wybrane problemy wytwarzania systemów czasu rzeczywistego dla bezpilotowych statków powietrznych, in Madeyski L. Kosiuczenko P., Bolanowski M. (eds.), *Inżynieria oprogramowania i systemy czasu rzeczywistego: od badań do praktycznych zastosowań*, *Zeszyty Rady Naukowej Polskiego Towarzystwa Informatycznego* pp. 137–155, (2017), (in Polish).
14. STANAG 4586 (EDITION 3) – Standard Interfaces of UAV Control System (UCS) for NATO UAV Interoperability, NSA//1235(2012)4586, (2012).
15. ISO/IEC 9899:1999: *Programming languages – C* (1999).
16. Nowak D., Kopecki G., Orkisz M., Rogalski T., Rzucidło P.: The Selected Innovative Solutions in UAV Control Systems Technologies. In: Nawrat. M A. (eds) *Innovative Control Systems for Tracked Vehicle Platforms. Studies in Systems, Decision and Control*, vol 2, pp. 39–56. Springer, Cham (2014)
17. Dołęga B., Kopecki G., Tomczyk A.: Possibilities of using software redundancy in low cost aeronautical control systems, 2016 IEEE Metrology for Aerospace (MetroAeroSpace), Florence, pp. 33–37 (2016).
18. Pieniążek J., Ciecinski P., Walek L., Nowak D.: Integrated measurement system for UAV, 2015 IEEE Metrology for Aerospace (MetroAeroSpace), Benevento, pp. 431–436 (2015).