논문 21-46-04-13 The Journal of Korean Institute of Communications and Information Sciences '21-04 Vol.46 No.04
https://doi.org/10.7840/kics.2021.46.4.679

PX4 기반 무인비행체의 GPS 시간 동기화 오차 최소화에 관한 연구 및 구현

심 후 엽•, 주 현 태*, 김 황 남°

Minimizing GPS Clock Synchronization Error for PX4-Based Unmanned Aircraft

Hooyeop Shim•, Hyeontae Joo*, Hwangnam Kim°

요 약

드론과 같은 무인비행체를 기반으로 Time Sensitive Network(TSN)를 구축하기 위해서는 매우 정밀하게 노드 간 시간 동기화가 이루어져야 한다. 이를 위해서는 각각의 무인비행체가 GPS 모듈로부터 수신한 시간 정보를 이용하여 비행 컨트롤러의 시간을 동기화 할 때 발생하는 오차를 최소화해야 한다. 본 논문에서는 무인비행체의 기본적인 하드웨어 구성을 변경하지 않고 GPS 모듈과 직접 연결된 비행 컨트롤러를 GPS 시간으로 동기화하는 방법을 제안한다. 비행 컨트롤러는 1초에 5번 이상 GPS 모듈로부터 시간 정보가 포함된 메시지를 수신하고 RTOS의 시간을 업데이트 하지만 비행 컨트롤러 간 시간 동기화 오차는 평균 21.64ms로 매우 크게 발생한다. 이러한 시간 동기화 오차와 S/W 지연을 최소화하기 위해서 우리는 GPS 모듈과 비행 컨트롤러의 각종 파라미터를 일치시키고 PX4 펌웨어의 코드를 개선하며 RTOS의 기본 설정을 변경했다. 이를 통해 비행 컨트롤러 간 발생하는 시간 동기화 오차를 최대 84.1% 감소시킬 수 있었다.

Key Words : GPS Clock Synchronization, TSN, UAV, PX4, RTOS

ABSTRACT

In order to build an Time Sensitive Network(TSN) based on unmanned aircraft such as drones, clock synchronization between nodes must be very precise. To do this, it is necessary to minimize the error that occurs when each unmanned aircraft synchronizes the clock of the flight controller using the time information received from the GPS module. This paper proposes a method to synchronize the clock of the flight controller directly connected to the GPS module without changing the general hardware configuration of the unmanned aircraft. The flight controller receives a message containing time information from the GPS module more than 5 times per second and updates the clock of the RTOS, but the clock synchronization error between controllers is very large, with an average of 21.64ms. To minimize the synchronization error and S/W delay, we match various parameters of the GPS module and flight controller, improve the firmware source code running on the controller, and change the default settings of the RTOS. This can reduce the clock synchronization error between flight controllers by up to 84.1%.

※ 본 연구는 방위사업청과 국방과학연구소가 지원하는 군집형 무인 CPS 특화연구실 사업의 일환으로 수행되었습니다.(UD190029ED)
• First Author : Korea University Dept. of Electrical and Computer Engineering, hooyp@korea.ac.kr, 학생회원
° Corresponding Author : Korea University Dept. of Electrical and Computer Engineering, hnkim@korea.ac.kr, 종신회원
* Korea University Dept. of Electrical and Computer Engineering, motern800@korea.ac.kr, 학생회원
논문번호 : 202012-317-D-RN, Received December 17, 2020; Revised January 22, 2021; Accepted January 27, 2021

Ⅰ. 서 론

최근 산업자동화, 자율주행, IoT 등의 기술이 발전함에 따라 실시간, 저지연 통신에 대한 수요가 급증했고 deterministic한 패킷 기반 네트워크를 구축하기 위해 Time Sensitive Network(TSN) 관련 표준이 제정됐으며 관련 기술이 발전해오고 있다. TSN의 성능을 최대한 발휘하기 위해서는 네트워크 내 모든 노드들의 시간을 정밀하게 동기화해야 하며 이를 위해 IEEE TSN TG에서는 기존에 채택된 표준들을 개량한 IEEE 1588 v2 Precision Time Protocol(PTP), IEEE 802.1 AS-2011 등과 같은 표준을 재차 정의하고 있다[1].

무선 네트워크 분야에서도 이러한 TSN 개념을 적용한 Wireless TSN을 구축하고자하는 움직임이 있어왔으며 기존 TSN과 유사한 수준의 정밀도를 가지도록 시간 동기화가 이루어져야만 유무선 TSN 링크 및 노드가 혼재되어 있는 전체 네트워크를 효과적으로 통합할 수 있고 중앙 컨트롤러에 의해 TSN의 핵심 기능이 작동하게 되며 초저지연, 초정밀 네트워크를 구축할 수 있다. 이미 TSN TG은 무선 네트워크 환경에서도 정확하게 시간을 동기화하기 위하여 WiFi의 IEEE 802.11 Time Measurement(TM)를 사용하도록 802.1 AS 표준을 정의했으며, 그 외에도 802.11 WiFi, 802.15.4 Zigbee 등의 기술을 이용하여 무선 네트워크의 노드 간 시간을 정밀하게 동기화하는 다양한 기술들이 연구되고 있다[2].

그러나 A.Nasrallah et al.은 최근 시간 동기화 분야에서 동기화를 위해 종단 간 메시지를 교환하는 방법의 단점을 지적했다[3]. 일반적으로 네트워크에 속한 모든 노드의 시간을 정밀하게 동기화하기 위해서 PTP와 Network Time Protocol(NTP)과 같은 양방향 시간 동기화 기법을 사용한다. PTP의 경우에는 전체 네트워크를 master와 slave로 구분하고 모든 slave 노드들의 시간을 master 노드의 시간과 동기화하기 위해 수차례 패킷을 교환하여 전송 및 처리 지연 등의 여러 요인으로 인해 발생한 시간오차를 최소화한다. 또한 IEEE 802.11 Timing Synchronization Function(TSF) 기술과 같이 Access Point(AP)가 자신의 시간 정보를 주변 노드로 브로드캐스팅하는 단방향 시간 동기화 기법도 이미 표준화되어 널리 사용되고 있다. 그러나 양방향 또는 단방향 시간 동기화 기법의 특성상 시간 정보를 포함한 메시지를 전달하기 위하여 별도의 트래픽이 생성될 수밖에 없으며 해당 논문의 저자들은 시간 동기화의 정밀성을 유지하기 위해 주기적으로 발생하는 트래픽이 일반적으로 발생하는 트래픽에 영향을 미치기 때문에 전체 네트워크의 처리량 또한 감소할 수 있다고 분석했다.

다수의 무인비행체가 군집을 이루어 하나의 무선 네트워크를 형성할 때, 자세 제어, 군집대형 변경, 임무 전달 및 수행 등의 제어 패킷이 최대한 손실되지 않고 모든 노드에 전달되어야 한다[4]. 따라서, 이 경우에도 노드 간 시간이 매우 정밀하게 동기화되어야 한다. 이를 위해서는 먼저 무인비행체를 구성하는 H/W의 특징을 고려해야 하는데 대부분의 경우, IoT 장치에 주로 사용되는 저전력 Micro Controller Unit(MCU) 기반의 비행 컨트롤러가 기체를 제어하는 핵심 역할을 수행하고 위치, 속도, 시간 등의 정보를 파악하기 위해 GPS 모듈이 부착된다. 또한 무인비행체의 여러 상태정보를 파악하고 지상국 또는 주변 노드와 무선 네트워크를 구성하는 등의 다양한 기능을 수행하기 위해 비행 컨트롤러에 네트워크 모듈이 부착된 Single Board Computer(SBC)를 연결하여 운용할 수 있다[5].

하지만, 무인비행체를 구성하는 H/W에는 대부분 성능이 좋지 않은 프로세서들이 탑재되어 있고 PCI Express(PCIe) 또는 mini-PCI Express (mPCIe) 인터페이스를 제공하지 않기 때문에 다른 시간 동기화 연구에서 사용되는 고성능 Network Interface Card(NIC)와 같은 H/W를 추가로 운용할 수 없다. 여러 제한사항을 고려했을 때 무인비행체에 가장 적합한 시간 동기화 방법은 WiFi나 Zigbee 등의 다른 통신 프로토콜을 사용하지 않고 기본적으로 포함된 GPS 모듈을 통해 수신하는 위성 시간을 기준으로 비행 컨트롤러에서 실행되는 운영체제의 시간을 동기화하는 것이다. 본 논문에서 다루는 PX4 펌웨어 또한 GPS 시간이 포함된 메시지를 수신하자마자 시간 정보를 추출하고 이를 기반으로 운영체제의 시간을 업데이트한다. 여기서 PX4는 리눅스 재단이 운영하는 Dronecode 프로젝트의 일부로서 무인비행체를 제어하는 알고리즘이 포함된 펌웨어와 비행 컨트롤러의 프로세서, 메모리, 입출력 장치 등을 제어하는 Real-time Operating System(RTOS)로 구성된 무인비행체 플랫폼을 의미한다[6].

무인비행체에 부착된 GPS 모듈이 여러 위성으로부터 시간 정보를 수신할 때, 수신 신호의 세기와 위성 배치, 전파 간섭 등의 다양한 요인이 영향을 미치지만 대부분 나노초 단위의 매우 작은 시간오차가 발생한다. 하지만 GPS 시간 정보가 각 무인비행체의 비행 컨트롤러에 전달되고 PX4 펌웨어가 이를 파싱하여 RTOS의 시간을 업데이트 할 때까지 전송 및 처리 지

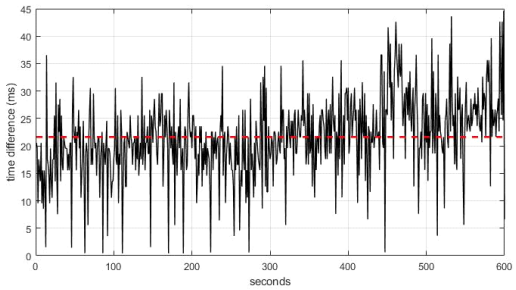

그림 1. 개선 전 비행 컨트롤러 간 시간 동기화 오차
Fig. 1. Clock sync error between flight controllers without any improvement

연, 클럭 오실레이터의 부정확성 등 다양한 요인이 영향을 미치게 된다. 그림 1은 본 논문에서 제안하는 개선사항들을 적용하기 전 두 대의 비행 컨트롤러에서 발생하는 시간 동기화 오차를 600초 간 측정한 결과이며 빨간 점선으로 표시된 바와 같이 21.64ms의 매우 큰 오차가 발생하게 된다. 본 논문은 2020년 하계종합학술발표회에서 발표된 논문[7]을 확장하여 연구한 것으로 GPS 모듈과 비행 컨트롤러의 각종 파라미터를 일치시키고 PX4 펌웨어 코드 개선 및 RTOS 설정 변경을 통해 비행 컨트롤러의 시간을 업데이트하는 과정에서 발생 가능한 S/W 지연을 최소화하는 방안을 제안한다. 또한 대표적인 무인비행체인 드론을 이용하여 테스트베드를 구축하고 시간 동기화 오차를 측정함으로써 성능평가를 수행한다.

Ⅱ. 본 론

이 장에서는 정확한 위치정보를 얻기 위해 GPS 모듈을 주로 사용하는 드론이 GPS 위성으로부터 수신한 시간 정보를 기준으로 동기화를 수행했을 때 드론 간에 발생하는 시간 동기화 오차를 최소화하기 위한 방안을 세부적으로 기술한다. 본 논문에서는 제안된 기법을 실제 드론에 구현하였으며 해당 드론을 구성하는 여러 H/W 및 S/W에 대한 세부정보는 표 1과 같다.

표 1. 드론의 H/W 및 S/W에 대한 세부정보
Table 1. Specification of the drone's H/W and S/W

Component	Description
Drone Model	DJI F550
Flight Controller(MCU)	Pixhawk1 (Cortex M4, STM32F427)
Firmware(version)	PX4(1.9.0)
GPS Module	Here+(U-blox M8P)

2.1 파라미터 설정

2.1.1 GPS 모듈 파라미터 설정

PX4 펌웨어의 소스코드에서 GPS 모듈로부터 메시지를 수신하고 처리하는 기능을 분석해보면 U-blox M8P 모듈의 경우는 measurement rate가 5hz로 기본 설정되어 있는 것을 확인할 수 있다. 이는 위치, 속도, 시간 등의 정보가 포함된 메시지를 업데이트하는 주기를 의미하며 비행 컨트롤러는 1초에 5번 해당 메시지를 수신하게 된다. 또한 U-blox에서 배포한 매뉴얼에 따르면 펄스 신호와 관련된 설정을 변경함으로써 GPS 메시지를 전송하는 시점을 설정할 수 있다[8]. 본 논문에서는 GPS 모듈의 여러 파라미터 중 UBX-CFG-TP5에 해당되는 속성을 표 2와 같이 설정했다.

이와 같이 파라미터를 설정하면 GPS 모듈은 기준 시간의 매초가 시작되는 순간에 펄스 신호를 발생시키며, 그 후 드론의 현재 위치 및 속도를 추정하고 시간 정보를 이에 더하여 GPS 메시지를 생성한다. 아래 그림 2를 통해 펄스 신호는 1초 마다 생성되고 GPS 메시지는 정확히 200ms 간격으로 1초에 5번 생성되는 것을 알 수 있다. 또한 이렇게 생성된 메시지는 시리얼 포트를 통해 비행 컨트롤러로 전송된다. 모든 드론에 부착된 GPS 모듈의 파라미터를 동일하게 설정

표 2. UBX-CFG-TP5 파라미터 설정
Table 2. UBX-CFG-TP5 parameter setting

Classification	Name	Value
Parameters	freqPeriod	1000000(us)
	pulseLenRatio	100000(us)
Flags	active	1
	isFreq	0
	isLength	0
	lockGnssFreq	1
	alignToTow	1
	polarity	1

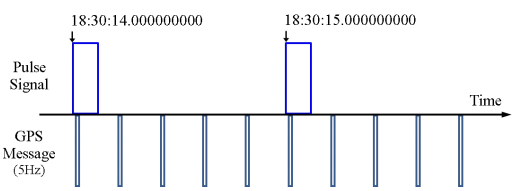

그림 2. 펄스 신호 생성 후 GPS 메시지 전송
Fig. 2. Sending GPS Message After Pulse Signal Generation

한다면 각각의 비행 컨트롤러는 같은 시점에 GPS 메시지를 수신할 수 있게 된다.

2.1.2 비행 컨트롤러 파라미터 설정

다양한 종류의 무인비행체를 관리하고 통제하는 기능을 제공하는 Qground Control, Mission Planner와 같은 오픈소스 S/W를 이용하여 비행 컨트롤러의 파라미터를 수정하여 펌웨어에 적용할 수 있다. 그 중 GPS 모듈과 연결되는 포트에 대한 Baudrate 값을 나타내는 SER_GPS1_BAUD 파라미터의 경우, 초기값으로 특정 속도(bps)가 지정되어 있지 않고 Auto로 설정되어 있다. 이 경우, PX4 펌웨어는 GPS 모듈의 시리얼 포트에 설정된 Baudrate 값을 알 수 없으므로 {38400, 57600, 9600, 115200, 230400}와 같이 선언된 배열의 첫 번째 원소부터 해당 포트의 Baudrate으로 설정해가며 시리얼 통신이 연결되는지 점검하는 과정을 수행한다. 연결 실패 시에는 배열의 다음 원소를 Baudrate으로 설정하여 같은 과정을 반복하고 정상적으로 연결된 후에는 GPS 모듈의 프로토콜 버전 정보를 확인한다. 만약 프로토콜 버전이 M8P 모듈에서 사용하는 버전과 같거나 높으면 Baudrate은 펌웨어에 의해 115200bps의 속도로 설정된다. 그러나 특정 사용자에 의해 파라미터 설정이 변경되거나 낮은 버전의 GPS 모듈을 사용할 경우 서로 다른 Baudrate 값이 설정된 상태로 시리얼 통신이 연결된다. 이 경우 고정된 크기의 GPS 메시지를 수신할 때 비행 컨트롤러 마다 각기 다른 시간이 소요되며 결국 시간 동기화 오차는 더욱 크게 발생하게 된다. 이와 같은 문제를 해결하기 위해 본 논문에서는 SER_GPS1_BAUD 파라미터와 GPS 모듈의 UBX-CFG-PRT 파라미터를 115200bps로 일치했다. PX4 펌웨어의 경우, SER_GPS1_BAUD 파라미터에 특정 속도값을 지정하면 상기에 언급한 비효율적인 과정을 거치지 않고 바로 시리얼 통신을 연결할 수 있으며 각각의 비행 컨트롤러 마다 서로 같은 수준의 전송 지연이 발생하게 되므로 시간 동기화의 정확성을 더욱 향상시킬 수 있다.

2.2 PX4 펌웨어 분석 및 개선

이 절에서는 오픈소스로 공개된 PX4 펌웨어 소스코드[9] 중 GPS 관련 핵심 기능의 실행 프로세스를 분석하고 어떤 부분을 수정하여 개선했는지 기술한다.

2.2.1 비행 컨트롤러 부팅 후 GPS 드라이버 실행

비행 컨트롤러 최초 부팅 시 그림 3에 나타난 GPS 드라이버 초기 실행 과정에 의해 /etc/init.d 내

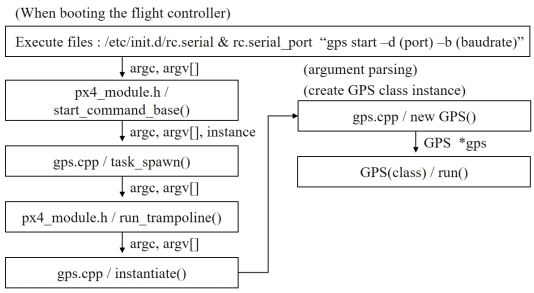

그림 3. GPS 드라이버 초기 실행 과정
Fig. 3. The process of initiating the GPS driver

rc.serial이라는 파일이 자동으로 실행된다. 이 파일은 같은 디렉토리 내 rc.serial_port를 실행하고 사전에 설정된 파라미터를 참조하여 시리얼 포트, Baudrate 등 핵심 기능 실행에 필요한 정보들을 추출한다. 그리고 'gps start -d (시리얼 포트) -b (Baudrate)', 'mavlink start -d (시리얼 포트)' 와 같은 명령어를 수행하여 GPS와 Mavlink 드라이버를 각각 실행시킨다.

start 명령어가 식별되면 펌웨어는 각 드라이버와 관련된 task가 수행 중인지 점검하고 사전에 수행되고 있는 task가 없는 경우에 해당 task를 실행한다. 이어서 GPS 모듈이 연결된 시리얼 포트와 Baudrate으로 설정한 값이 GPS::instantiate (int argc, char *argv[], Instance instance) 함수를 통해 인자로 전달되고 이를 파싱한 후 알맞은 멤버 변수에 각각 저장한다. 최종적으로 GPS 클래스의 객체를 생성하면서 생성자의 인자로 해당 변수 내 저장된 값들을 전달하게 된다.

2.2.2 GPS 모듈의 설정 초기화 및 데이터 수신

PX4 펌웨어는 기본적으로 4가지 종류의 GPS 모듈을 지원하지만 사용자가 별도로 지정하지 않는 경우에는 그림 4에 나타난 GPS 모듈 초기 설정 및 GPS 메시지 수신 과정과 같이 자동으로 U-blox사의 GPS 모듈을 지원하기 위해 GPSDriverUBX 클래스의 객체를 생성한다. 그 후 시리얼 연결 설정을 위해 configure() 함수가 호출되는데 M8P 모듈의 경우는 configureDevicePreV27() 함수가 호출되며 UBX-CFG-RATE, NAV5, PRT 등의 파라미터에 초기값을 설정하여 GPS 모듈로 전송한다.

초기 설정 후 GPS 모듈로부터 데이터를 수신하기 위해 receive() 함수를 반복적으로 호출하며 gps.cpp에 선언된 GPS::callback() 함수가 호출된다. callback 함수가 호출된 후, GPS 모듈로부터 데이터 수신 시 가장 중요한 역할을 하는 pollOrRead() 함수가 호출되는데 이 때 uint8_t 타입의 변수 150개로 이루어진 배

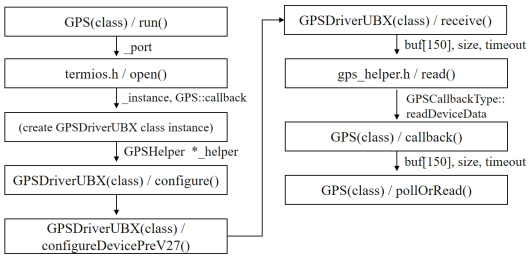

그림 4. GPS 모듈 초기 설정 및 GPS 메시지 수신 과정
Fig. 4. The process for initial setup and GPS message reception

열이 생성되고 해당 배열의 포인터도 함께 인자로 전달된다. pollOrRead() 함수의 세부 동작과정을 살펴보면 GPS 모듈과 수립된 시리얼 연결에 대해 poll() 함수를 호출하여 지정된 시간 동안 POLLIN 이벤트가 발생하는지 확인한다. GPS 모듈이 데이터를 송신할 때 POLLIN 이벤트가 발생하는데 이 때, ioctl() 함수를 호출하여 읽을 수 있는 데이터의 크기를 확인한다. PX4 펌웨어는 읽을 수 있는 데이터의 크기가 32바이트 보다 작으면 Baudrate에 따라 계산된 Sleeptime만큼 대기하고 32바이트 보다 크면 termios.h에 정의된 read 함수를 호출하여 데이터를 바이트 단위로 수신한다.

2.2.3 RTOS 시간 최신화 및 uORB 메시지 생성

GPS 모듈로부터 시간 정보가 포함된 GPS 메시지가 수신될 때 마다 펌웨어는 setClock() 함수를 호출하여 RTOS 시간을 최신화한다. 2.1.1절에서 서술한 바와 같이 GPS 메시지는 1초에 5번 수신하기 때문에 RTOS 시간은 200ms 간격으로 업데이트 된다. 펌웨어가 GPS 메시지를 수신한 후에 파싱을 통해 세부 정보들을 추출하며 그 후에는 vehicle_gps_position_s, satellite_info_s 등의 uORB(micro Object Request Broker) 메시지를 생성한다. 이러한 메시지들은 GPS 모듈로부터 수신한 위치, 속도, 시간 등의 정보들을 포함하고 있으며 mavlink 프로토콜을 통해 비행 컨트롤러 외부의 다른 장치들로 전달된다.

2.2.4 PX4 펌웨어 개선

PX4 펌웨어에서는 termios.h의 read 함수가 비교적 많은 컴퓨팅 자원을 사용하기 때문에 시스템의 성능 저하를 방지하고자 Sleeptime을 도입한다. 다음에 읽을 데이터의 크기가 MinimumBytes로 정의된 특정 임계치보다 크면 계속 데이터를 수신한다. 이와 반대로 임계치보다 작은 크기의 데이터를 수신하게 될 경우에는 Baudrate을 기반으로 식 (1)과 같이 Sleeptime을 계산하고 그만큼 대기하게 된다. GPS 모듈과 비행 컨트롤러의 파라미터를 설정할 때 115200의 속도로 Baudrate을 일치시켰으므로 Sleeptime은 2777us로 산출된다.

$$Sleeptime = \frac{MinimumBytes \times 10^6}{(Baudrate/10)} \quad (1)$$

드론이 다수일 때, 이와 같은 구조에서는 각각의 드론이 Sleeptime 시간 동안 대기하는 시점과 그 횟수가 서로 다르게 나타나고 이로 인해 드론 간 시간 동기화 오차가 증가하는 문제가 발생한다. 다음에 읽어들일 데이터 크기가 특정 임계치 보다 작거나 큰 경우를 모두 예측할 수 없으므로 본 논문에서는 데이터를 수신할 때 마다 서로 같은 시간만큼 대기하도록 코드를 개선함으로써 시간 동기화 오차에 미치는 영향을 최소화했다. 대기 시간을 정하기 위해서는 RTOS의 1 tick 간격을 고려해야 하는데 초기값이 1000us로 설정되어 있기 때문에 펌웨어에서는 마이크로 초 단위로 대기할 수 없다. 이에 따라 본 논문에서는 밀리 초 단위의 상수인 5000(us)을 대기 시간으로 설정했다.

2.3 RTOS 코드 분석 및 개선

PX4는 다양한 종류의 비행 컨트롤러를 지원하기 위해 각각의 장치에 탑재된 프로세서, 메모리에 대한 정보와 입출력 포트 개수 및 속성 등에 대한 정보가 모두 포함된 설정 파일들을 제공한다. 이러한 설정 파일을 기반으로 비행 컨트롤러를 효율적으로 제어하기 위해 NuttX OS라는 오픈소스 RTOS를 사용한다. NuttX OS는 Pixhawk1에 탑재된 Cortex M4 칩(STM32F427) 뿐만 아니라 수많은 저전력, 고성능 MCU를 완벽히 지원한다. 이 절에서는 펌웨어에서 수신한 GPS 시간 정보가 RTOS로 전달되어 시간이 업데이트되는 과정을 분석하며 RTOS 설정 중 어떤 부분을 개선했는지 자세히 기술한다.

2.3.1 GPS 시간을 기반으로 RTOS 시간 최신화

2.2.3절에서 서술한 바와 같이 GPS 메시지가 수신될 때 마다 RTOS의 시간이 업데이트 되는데 이 때 그림 5의 RTOS 시간 최신화 과정에 나타난 것과 같이 펌웨어는 setClock() 함수를 호출하고 callback 함수를 거쳐서 px4_time.h에 정의된 px4_clock_settime() 함수를 호출한다. 이 때, GPS 시간 정보가 저장된 timespec 구조체의 포인터를 인자로 전달하며 RTOS는 clock_settime.c 에 정의된 clock_settime() 함수를

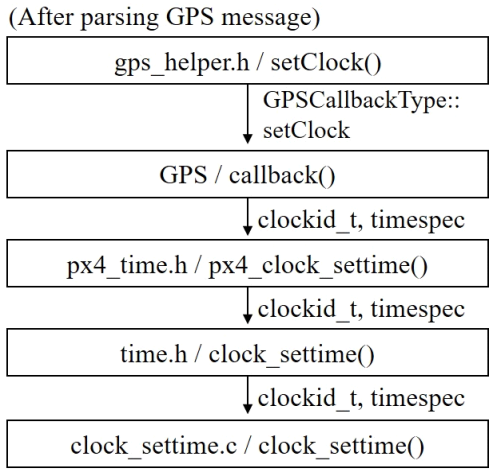

그림 5. RTOS 시간 최신화 과정
Fig. 5. The process of updating RTOS clock

호출한다. Pixhawk1의 H/W 특성상 밀리 초 이하의 정밀한 시간 정보를 저장할 수 있는 고정밀 Real Time Clock(RTC)이 없기 때문에 부팅 후 지난 시간을 나타내는 elapsed time을 이용하여 정밀한 시간을 제공한다.

먼저 시간을 업데이트하는 경우, 펌웨어로부터 전달된 GPS 시간에서 elapsed time을 뺀 값을 base time으로 저장한다. 그 후 펌웨어가 RTOS의 현재 시간을 확인할 경우에는 기존에 저장했던 base time에 해당 시점의 elapsed time을 더하여 반환한다. 그러므로 RTOS는 elapsed time만 정밀하게 관리된다면 GPS 모듈에 의해 제공되는 나노초 단위의 매우 정밀한 시간을 지속적으로 유지할 수 있다. 이를 위해서는 정확히 1 tick으로 설정된 시간이 지날 때마다 인터럽트가 발생하여 tick 값이 1씩 증가되어야 한다.

그러나 비행 컨트롤러는 제한된 프로세싱 성능을 가진 MCU를 기반으로 RTOS와 펌웨어를 동시에 실행하며 수많은 인터럽트를 처리한다. 만약, 현재 처리 중인 인터럽트가 있거나 우선순위가 높은 인터럽트가 대기 중일 때 tick 값을 증가시키는 인터럽트가 발생한다면 제 때 처리될 수 없고 이로 인해 드론 간 시간 동기화 오차가 크게 발생하게 된다. 본 논문에서는 이와 같은 문제를 해결하고 시간 동기화 정확성을 향상시키기 위해 다음과 같이 RTOS의 설정값을 변경하였다.

2.3.2 1 tick의 시간간격 설정 변경

RTOS는 MCU가 제공하는 SYSTICK 기능을 이용하여 1 tick에 설정된 시간이 지나면 인터럽트를 발생시킨 후 elaspsed time을 계산하는데 사용되는 uint32_t 타입의 g_system_timer 변수 값을 1씩 증가시킨다. 1 tick의 간격은 CONFIG_ USEC_PER_TICK 값에 따라 다르게 적용되며 최초 설정값은 1000us(1ms)로 되어있다. 초기 설정 값이 적용된 상황에서는 비행 컨트롤러 간 시간 동기화 오차를 마이크로 초 수준으로 감소시킬 수가 없기 때문에 1 tick 시간간격 설정을 50, 200, 500 등 다양하게 변경해가며 드론에 실제 적용하였다. 아래 표 3은 CONFIG_USEC_PER_TICK 값 변경에 따른 초당 인터럽트 발생 횟수를 보여주며 해당 설정 값이 작을수록 초당 인터럽트 발생 횟수는 증가하고 이에 반해 설정 값이 클수록 초당 인터럽트 발생 횟수는 감소한다는 것을 알 수 있다.

표 3. 1 tick 시간간격 설정 변경에 따른 초당 인터럽트 발생 횟수
Table 3. Number of interrupts per second due to changing 1 tick interval setting

	1 Tick interval			
	50	200	500	1000
Interrupts per Second	20,000	5,000	2,000	1,000

2.3.3 SYSYICK 인터럽트의 우선순위 변경

Cortex M4의 경우, Nested Vectored Interrupt Controller(NVIC) 기능을 제공하여 240개의 서로 다른 인터럽트를 효율적으로 관리할 수 있고 각 인터럽트에 0 ~ 255 사이의 값을 우선순위로 부여할 수 있다[10]. 아래 그림 6은 Cortex M4의 인터럽트 벡터 테이블로 이를 통해 전체 인터럽트가 시스템 인터럽트와 일반 인터럽트로 구분되는 것을 알 수 있다. SYSTICK 인터럽트는 시스템 인터럽트의 한 종류로서 시스템 핸들러에 의해 처리되고 System Handler Priority Register (SHPR)에 우선순위 값이 저장된다. 그리고 일반 인터럽트는 Interrupt Service Routines(ISR)에 의해 처리되며 우선순위 값은 Interrupt Priority Register(IPR)에 저장된다. 아래 그림 7은 SYSTICK 인터럽트의 우선순위가 SHPR 1, 2, 3 중 SHPR3에 저장되는 것을 보여준다. 또한 그림 6에 표기된 시스템 인터럽트 중 Reset, Non-maskable Interrupt(NMI), HardFault는 각각 -3, -2, -1로 우선순위 값이 고정되어 있으며 이 값은 변경할 수 없다. 그러나 이 외에 모든 인터럽트는 가장 높은 우선순위인 0부터 가장 낮은 우선순위인 255까지 어떤 값으로도 설정할 수가 있다.

Exception number	IRQ number	Offset	Vector
16+n	n	0x0040+4n	IRQn
.	.	.	.
.	.	.	.
.	.	.	.
		0x004C	
18	2	0x0048	IRQ2
17	1	0x0044	IRQ1
16	0	0x0040	IRQ0
15	-1	0x003C	Systick
14	-2	0x0038	PendSV
13			Reserved
12			Reserved for Debug
11	-5	0x002C	SVCall
10			Reserved
9			
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value

그림 6. 인터럽트 벡터 테이블[10]
Fig. 6. Interrupt vector table[10]

31 ~ 24	23 ~ 16	15 ~ 0
PRI_15	PRI_14	Reserved

Table 4-23 SHPR3 register bit assignments

Bits	Name	Function
[31:24]	PRI_15	Priority of system handler 15, SysTick exception
[23:16]	PRI_14	Priority of system handler 14, PendSV
[15:0]	-	Reserved.

그림 7. SHPR3의 bit 할당 구조[10]
Fig. 7. The bit assignments of SHPR3[10]

NuttX OS가 부팅될 때 stm32_irq.c 파일에 선언된 up_irqinitialize() 함수가 호출되며 SHPR 1 ~ 3에 저장된 모든 인터럽트의 우선순위를 0x80으로 초기화하며 이 때, SYSTICK 인터럽트의 우선순위 또한 0x80으로 설정된다. 그러므로 본 논문에서는 SYSTICK 인터럽트의 우선순위를 최대한 높게 설정하여 1 tick을 더욱 정밀하게 증가시키기 위해 up_prioritize_irq() 함수를 사용했다. 최초 부팅 시 up_prioritize_irq(15, 0x00)와 같이 함수를 호출하면 그림 7에 나타난 SHPR3의 PRI_15 필드에 0x00 값이 할당된다. 즉. SYSTICK 인터럽트의 우선순위는 설정 가능한 범위 안에서 가장 높게 상향 조정된다.

Ⅲ. 성능 평가

3.1 실제 드론을 이용한 테스트베드 구축

본 논문에서 실제 구현에 사용한 Pixhawk 1의 H/W 특성상 General Purpose Input/Output (GPIO) 핀이나 parallel 포트를 지원하지 않기 때문에 펄스 신호를 발생시켜 시간 동기화 오차를 측정할 수 없다. 그러므로 mavlink 프로토콜 중 PING 메시지를 활용하여 드론 간 시간 동기화 오차를 측정하는 테스트베드를 구축했다. 이를 위해 리눅스 기반의 테스트 프로그램을 개발했고 PX4 펌웨어를 수정했으며 테스트베드의 실제 구성은 그림 8에 나타난 바와 같다. 본 논문에서는 실제 드론을 이용하여 아래와 같은 절차로 성능 평가를 수행했다.

3.1.1 시간 동기화 오차 측정 방법 및 절차

(1) 한 대의 노트북에 두 대의 비행 컨트롤러를 각각 micro USB케이블로 연결한다.
(2) 노트북에서 실행되는 테스트 프로그램은 두 개의 스레드를 생성하고 각 스레드는 연결된 컨트롤러와 시리얼 통신을 연결한 상태로 대기한다. 측정 시작 시간이 도래한 순간부터 스레드는 주기적으로 mavlink 프로토콜의 PING 메시지를 전송한다. 이 때, PING 메시지에는 노트북에 설치된 운영체제의 시간 정보가 추가된다.
(3) PX4 펌웨어는 PING 메시지를 수신하자마자 RTOS의 현재 시간과 PING 메시지에 저장되어 있던 시간 정보를 배열에 저장한다.
(4) 테스트 프로그램은 사전에 정해진 개수만큼 PING 메시지를 전송한 후, 테스트 종료 메시지를 전송하고 프로그램을 종료한다.

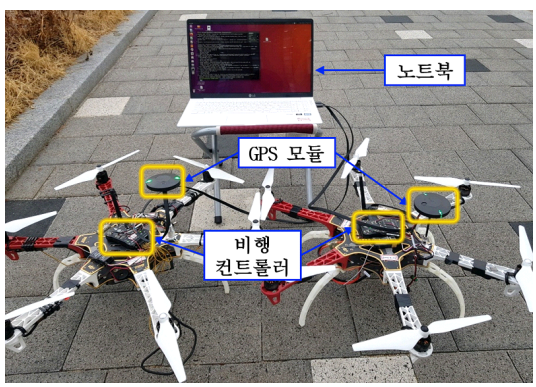

그림 8. 드론 간 시간 동기화 오차 측정을 위한 테스트베드
Fig. 8. Test-bed for clock sync error measurement between drones

(5) PX4 펌웨어는 테스트 종료 메시지를 수신하면 배열에 저장된 정보들을 SD카드에 기록한다.

3.2 성능 평가 결과

두 대의 비행 컨트롤러를 대상으로 단계별 개선안을 적용하여 시간 동기화 오차를 측정한 결과는 아래 표 4와 같다. GPS 모듈과 비행 컨트롤러의 파라미터를 일치시키고 PX4 펌웨어 코드를 개선한 경우가 표 4의 'STEP Ⅰ'에 해당되고 600초 간 시간 동기화 오차를 측정한 결과 평균 6.38ms로 기록되었다. 동일한 실험 결과를 그래프로 출력한 결과는 그림 9에 해당되며 빨간 점선은 평균 동기화 오차를 나타낸다. 또한 RTOS 설정을 변경하여 1 tick의 시간간격을 다양하게 설정하고 SYSTICK 인터럽트의 우선순위를 향상시킨 경우가 표 4의 'STEP Ⅱ'에 해당된다. 1 tick 시간간격을 50us로 설정했을 때 평균 시간 동기화 오차가 가장 작은 3.44ms로 측정됐고 동일한 실험 결과를 그림 10과 같이 그래프로 나타냈다. 1 tick 시간간격을 50us로 설정한 경우 인터럽트는 20배 더 많이 발생하지만 해당 인터럽트의 우선순위가 높게 설정되었기 때문에 더욱 정밀하게 tick을 증가시킬 수 있다. 결과적으로 STEP 1, 2를 모두 적용했을 때 비행 컨트롤러 간 시간 동기화 오차를 최대 84.1% 감소시킬 수 있었다.

표 4. 비행 컨트롤러 간 시간 동기화 오차
Table 4. Clock sync error between flight controllers

	Default	STEP Ⅰ	STEP Ⅱ		
1 Tick interval (us)	1000	1000	500	200	50
Mean error (ms)	21.64	6.38	5.37	6.15	3.44

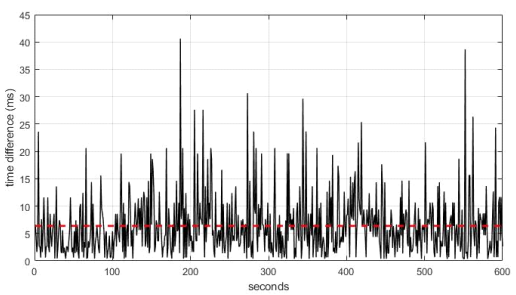

그림 9. STEP 1 적용 시 비행 컨트롤러 간 시간 동기화 오차
Fig. 9. Clock sync error between flight controllers after applying STEP 1

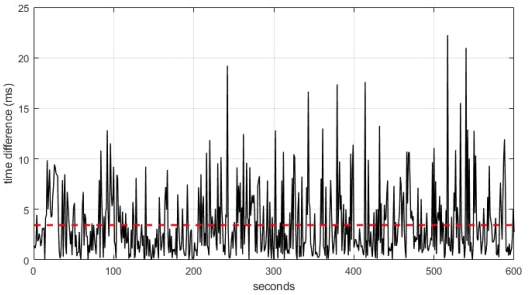

그림 10. STEP 2 적용 시 비행 컨트롤러 간 시간 동기화 오차(1 tick 시간간격=50 us)
Fig. 10. Clock sync error between flight controllers after applying STEP 2(1 tick interval=50)

Ⅳ. 결 론

이번 연구를 통해 GPS 모듈과 비행 컨트롤러의 각종 파라미터를 일치시키고 PX4 펌웨어 코드를 개선하며 RTOS 설정을 변경한 결과, 무인비행체의 비행 컨트롤러 간 발생하는 평균 시간 동기화 오차를 최대 84.1% 감소시킬 수 있었다. 그러나 GPS 시간을 기준으로 비행 컨트롤러의 시간을 동기화했음에도 불구하고 밀리 초 단위의 시간 오차가 발생하는 이유는 탑재된 MCU의 최대 클럭이 168Mhz로 높지 않고 High Speed External(HSE) 오실레이터의 정밀도 또한 -500 ppm에서 +500 ppm의 큰 폭으로 변동할 수 있기 때문이다[11]. 향후에는 이러한 제한사항을 극복하기 위해 GPS 모듈에서 생성되는 펄스 신호를 고성능 프로세서와 오실레이터가 장착된 SBC로 직접 전달하여 더욱 정밀하게 시간을 동기화하는 연구를 진행할 계획이다.

References

[1] A. Mildner, "Time sensitive networking for wireless networks-a state of the art analysis," *Network*, vol. 33, 2019.

[2] A. Mahmood, et al., "Clock synchronization over IEEE 802.11－A survey of methodologies and protocols," *IEEE Trans. Ind. Informatics*, vol. 13, no. 2, pp. 907-922, 2016.

[3] A. Nasrallah, et al., "Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research," *IEEE Commun. Surv. & Tuts.*, vol. 21, no. 1, pp.

88-145, 2018.

[4] J. Y. Lee, A. Y. Chung, H Shim, C. Joe, S. Park, and H. Kim, “UAV flight and landing guidance system for emergency situations,” MDPI sensors, vol. 19, no. 20, p. 4468, Oct. 2019.

[5] I. Um, S. Park, H. T. Kim, and H Kim, Configuring RTK-GPS architecture for system redundancy in multi-drone operations, *IEEE Access*, vol. 8, pp. 76228-76242, Apr. 2020.

[6] J. H. Lee and J. Cha, “Analysis on initialization process of PX4 Firmware open source for unmanned aerial vehicle,” *Korea Software Congress 2017*, pp. 734-735, Busan, Korea, Dec. 2017.

[7] H. Shim and H. Kim, “A study on the minimization of GPS time synchronization errors of unmanned aircraft based on PX4,” in *Proc. Symp. KICS*, pp. 78-79, Pyeongchang, Korea, Aug. 2020.

[8] U-blox, *u-blox 8 / u-blox M8 Receiver description*(2020), Retrieved Dec. 16, 2020, from https://www.ublox.com/sites/default/files/products/documents/u-blox8-M8_ReceiverDescrProtSpec_%28UBX-13003221%29.pdf.

[9] PX4-Autopilot, *PX4*, Retrieved Dec. 16, 2020, from https://github.com/PX4/PX 4-Autopilot.

[10] ARM, *Cortex M4 Devices Generic User Guide*(2010), Retrieved Dec. 16, 2020, from https://static.docs.arm.com/dui0553/a/DUI0553A_cortex_m4_dgug.pdf.

[11] STMicroelectronics, *STM32F427xxSTM32F429xx*(2018), Retrieved Dec. 16, 2020, from https://www.st.com/resource/en/datasheet/DM00071990.pdf.

심 후 엽 (Hooyeop Shim)

2013년 2월 : 국민대학교 경영정보학부 졸업

2021년 2월 : 고려대학교 전기전자공학부(공학석사)

2013년 3월~현재 : 대한민국 육군 대위

<관심분야> 무인이동체, 무선 네트워크, 강화학습

[ORCID:0000-0002-9955-6847]

주 현 태 (Hyeontae Joo)

2017년 2월 : 서울시립대학교 전가전기컴퓨터공학부 졸업

2020년 3월~현재 : 고려대학교 전기전자공학부 석박사통합과정

<관심분야> 무인이동체, 무선 네트워크, 가상화

[ORCID:0000-0002-3753-364X]

김 황 남 (Hwangnam Kim)

1992년 3월 : 부산대학교 컴퓨터공학과(공학사)

1994년 2월 : 서울대학교 컴퓨터공학과(공학석사)

2004년 2월 : 미국 Urbana-Champaign 소재 Illinois 주립대학 컴퓨터과학과(공학박사)

1994~1999년 : LG 전자 주임연구원

2004~2005년 : 미국 Urbana-Champaign 소재 Illinois 주립대학 Post Doctorate Fellow

2005~2006년 : 삼성전자 책임연구원

2012~2012년 : 미국 LA 소재 California 주립대학 방문연구원

2006~현재 : 고려대학교 전기전자공학부 교수

<관심분야> 유무선 네트워크, CPS, 무인이동체, 군집지능

[ORCID:0000-0003-4322-8518]