<학술논문>

항공기 충돌회피 시스템: C++ 기반의 알고리즘 구현과 RTOS를 이용한 MCU 성능 비교

김동신* · 박인혁*
* 한국항공대학교 항공우주 및 기계공학부

Aircraft Collision Avoidance System: Algorithm Design based on C++ and Performance Comparison of MCU using RTOS

Dong Sin Kim* , In Hyeok Park*
* School of Aerospace and Mechanical Engineering, Korea Aerospace Univ.

(Received June 22, 2018 ; Revised June 25, 2018 ; Accepted June 30, 2018)

Key Words: Aircraft Collision Avoidance System(항공기충돌방지장치), Real Time Operation System(실시간 운영체계), Throughput(처리시간), Memory usage(메모리 사용량),

초록:
본 논문에서는 C++ 언어를 기반으로 항공기 충돌회피 알고리즘을 구현하고, 동일한 알고리즘과 운영체계를 탑재한 MCU(Micro-Controller Unit) 간의 성능 비교를 수행한다. FreeRTOS는 RTOS의 한 종류이며, 사용한 항공기 충돌 회피 알고리즘은 수직 회피 알고리즘, 수평 회피 알고리즘, 수직과 수평을 결합한 3차원 회피 알고리즘이다. MCU로는 ATmega2560과 STM32 ARM Cortex-M3와 STM32 ARM Cortex-M4를 사용하였다. 충돌 회피 알고리즘은 C++ 언어로 구현하였으며, MCU 간의 성능 비교는 동일한 알고리즘 실행파일을 각 MCU에서 구동하였을 때 측정한 처리시간과 메모리 사용량을 기준으로 수행하였다.

Abstract: In this paper, we compare the performance of three MCU using RTOS and aircraft collision avoidance algorithms. we used FreeRTOS, one of RTOS, and 3 types of aircraft collision avoidance algorithm.(vertical avoidance algorithm, horizon avoidance algorithm, and three dimensional collision avoidance algorithm combined vertical and horizon avoidance algorithm.) MCU for comparison are ATmega2560, ARM Cortex-M3, and ARM Cortex-M4. Collision avoidance algorithm is written by C++, Downloaded and executed the code to the MCU. For compare performance of two MCU, we estimate Throughput and SRAM usage.

†Corresponding Author, ds.kim@kau.kr

1. 서 론

국제공항협의회(Airports Council International; ACI)에 따르면, 2016년 전 세계 항공 교통량은 승객 수를 기준으로 77억 명으로, 연간 4.9%씩 성장하여 2031년에는 두 배가 될 것으로 전망된다. 항공 교통량은 지속적으로 증가하는데 반해, 공역 내 수용량(Capacity)은 한정되어 있다. 이에 따라, 공역 포화상태에 대비하여 효율적으로 공역을 관리하기 위한 연구들이 진행되고 있다.(1)

대표적으로 유로컨트롤(Eurocontrol)에서 연구·시범운영 중인 자유항로공역(Free Route Airspace; FRA) 개념이 있다. 자유항로공역은 기존 항공교통업무 항로(Air Traffic Services Route)를 따라 비행하는 대신에, 공역의 입항지점과 출항지점을 결정한 후, 공역사용자(조종사 등)가 자유롭게 경로를 계획하여 비행할 수 있도록 허가한 공역이다. 현재 유럽 국가들은 자유항로공역의 야간운영 → 주말운영 → 주중운영으로 점진적인 운영 확대를 계획하고 있다. 자유항로공역의 도입은 항공기를 보다 효율적으로 운용할 수 있어 연료 소모량을 줄일 수 있고, 공역 내 수용량을 증대시킬 수 있을 것으로 기대된다.(2)

하지만, 자유항로공역 개념이 실현·확산되기 위해서는, 해당 공역 내에서 자유롭게 비행하는 항공기 간의 안전이 확실하게 보장되어야 한다. 따라서 꾸준하게 증가하고 있는 항공 교통 수요에 대비하여, 효율적인 공역관리에 대한 연구뿐만 아니라 항공기 간의 충돌감지 및 회피 기술에 대한 연구 역시 병행되어야 할 것이다.

본 논문에서는 C++ 언어를 기반으로 항공기 충돌회피 알고리즘을 구현하고, 동일한 알고리즘과 운영체계를 탑재한 MCU(Micro-Controller Unit) 간의 성능 비교를 수행한다. MCU는 8Bit 프로세서인 ATmega2560, 32Bit 프로세서인 ARM Cortex M3, ARM Cortex M4를 사용한다. 또한, 시간에 민감한 실제 항공기 운용 환경을 고려하여 RTOS (Real-Time Operating System)를 운영체계로 탑재한다. 탑재된 RTOS는 오픈소스인 FreeRTOS이다. 동일한 알고리즘 실행파일을 이용하여, FreeRTOS가 탑재된 3개의 MCU에서 처리시간과 메모리 사용량을 측정한다.

2. 본 론

2.1 알고리즘 특성 및 운동방정식

본 연구에서는 TCAS(Traffic Collision Avoidance System) 수직회피 알고리즘[3], Andrew Trapani 논문의 수평회피 알고리즘[4], 김영래 논문의 3차원 회피 알고리즘[5]을 사용하였다. 각 회피 알고리즘 특성은 Table.1과 같다.

Table 1 Collision avoidance algorithm characteristics

항목	수직	수평	3차원
상태변수 추정방법	Nominal	Nominal	Probabilistic
상태변수 차원	Vertical	Horizontal	H & V
충돌회피 방법	Prescribed	Prescribed	Prescribed
회피기동 방법	Vertical	Horizontal	H & V
다중충돌 회피	Pair-wise	Pair-wise	Pair-wise
충돌회피 책임	Cooperative	Cooperative	Cooperative

좌표계는 Fig.1과 같이 표현된다. 회피 알고리즘에 사용된 운동방정식은 식 (1)~(3)과 같다. 여기서, V는 항공기의 속도, γ는 비행경로각, 그리고 θ는 방위각을 나타낸다.

선회 운동방정식은 식 (4)~(6)과 같다. R은 선회 반경, ϕ는 항공기의 뱅크각, g는 중력 가속도, t는 선회를 시작한 시점부터의 경과된 시간, x_0, y_0는 선회 시작 전 항공기 좌표, 그리고 $sign$는 부호함수로 괄호 안의 값의부호에 따라 + 혹은 - 값을 반환한다.

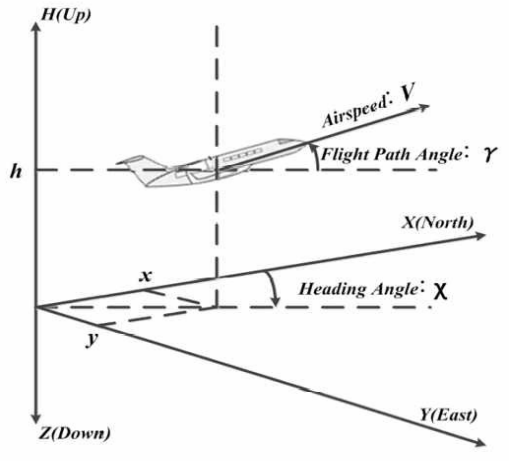

Fig. 1 Aircraft coordinate system

$$\dot{x} = V\cos\gamma\cos\theta \tag{1}$$

$$\dot{y} = V\cos\gamma\sin\theta \tag{2}$$

$$\dot{z} = V\sin\gamma \tag{3}$$

$$R = \frac{V^2}{g \cdot \tan\phi} \tag{4}$$

$$\triangle\theta = \frac{t \cdot g \cdot \tan\phi}{V} \tag{5}$$

$$\begin{aligned} P &= [x, y] \\ &= [x_0 + R \cdot sign(\triangle\theta) \cdot (\cos(\theta) - \cos(\theta + \triangle\theta)), \\ &\quad y_0 + R \cdot sign(\triangle\theta) \cdot (-\sin(\theta) + \sin(\theta + \triangle\theta))] \end{aligned} \tag{6}$$

2.2 알고리즘 Flowchart 및 Pseudocode

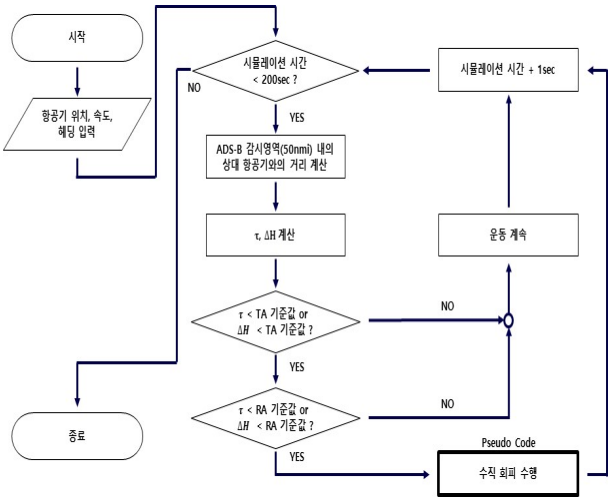

Fig. 2 Vertical avoidance flowchart

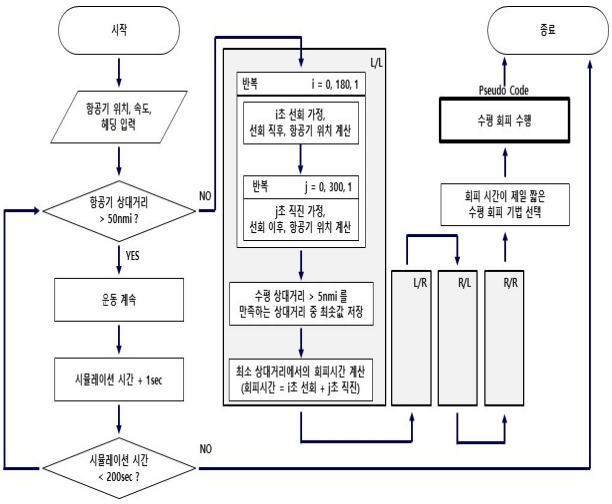

Fig. 3 Horizontal avoidance flowchart

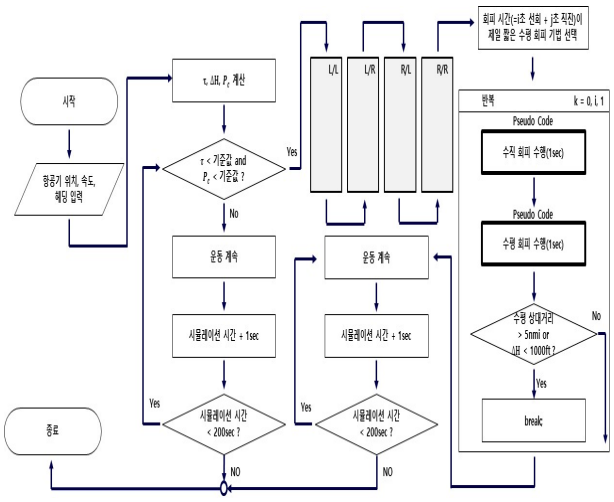

Fig. 4 Three-dimensional avoidance flowchart

1. get Altitude.A = The Altitude of the Aircraft A;
2. get Altitude.B = The Altitude of the Aircraft B;
3. **if** *Altitude.A is greater than or equal to Altitude.B* **then**
   The H-axis Velocity of the Aircraft A = 1500 feet/min;
   The H-axis Velocity of the Aircraft B = -1500 feet/min;
   else
   The H-axis Velocity of the Aircraft A = -1500 feet/min;
   The H-axis Velocity of the Aircraft B = 1500 feet/min;
   end
4. update Altitude.A;
5. update Altitude.B;

Algorithm 1: Do Collision Avoidance, Vertical

Fig. 5 Vertical avoidance pseudocode

1. get Time.A = The Time for the Aircraft A to complete a turn;
2. get Time.B = The Time for the Aircraft B to complete a turn;
3. get Delta.Psi.A = The Turn Angle for the Aircraft A;
4. get Delta.Psi.B = The Turn Angle for the Aircraft B;
5. get Velocity.A = The X-axis or Y-axis Velocity of the Aircraft A;
6. get Velocity.B = The X-axis or Y-axis Velocity of the Aircraft B;
7. get Position.A = The X-axis or Y-axis Position of the Aircraft A;
8. get Position.B = The X-axis or Y-axis Position of the Aircraft B;
09. update Position.A =+ Time.A x Velocity.A x sin(Delta.Psi.A);
10. update Position.B =+ Time.B x Velocity.B x sin(Delta.Psi.B);

Algorithm 2: Do Collision Avoidance, Horizontal

Fig. 6 Horizontal avoidance pseudocode

논문(3-5)을 참고하여 Fig.2~4와 같이 알고리즘 Flowchart를 작성하였다. Fig.5~6은 각각 수직회피와 수평회피에서의 알고리즘 Pseudocode 중 일부이다.

2.3 시뮬레이션 초기조건

작성한 Flowchart 및 Pseudocode를 바탕으로, 수직, 수평, 그리고 3차원 충돌회피 알고리즘을 C++ 언어로 구현하였다. 시뮬레이션을 수행하기 위하여, 한 쌍의 항공기가 서로 횡단하는 상황(#1)과 마주보며 가까워지는 상황(#2)을 정의하였다. 각 상황에 대한 초기조건은 Table.2~3과 같다.

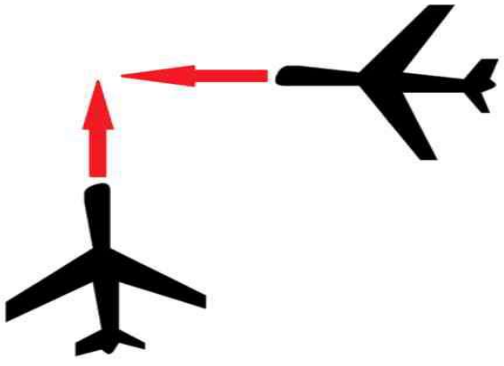

Fig. 7 Pair of aircraft, Scenario #1

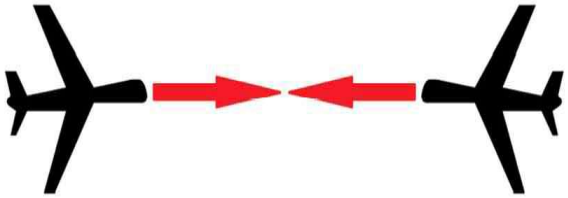

Fig. 8 Pair of aircraft, Scenario #2

Table 2 Initial condition of scenario #1

	항공기 A	항공기 B
초기위치	$(0,0,1)nmi$	$(12,12,1)nmi$
속도	$300knot$	$380knot$
상승/하강 속도	$1{,}500\,ft/\min$	$1{,}500\,ft/\min$
회피 뱅크각	20°	20°
헤딩	0°	270°

Table 3 Initial condition of scenario #2

	항공기 A	항공기 B
초기위치	$(-12,12,1)nmi$	$(12,12,1)nmi$
속도	*$300knot$	$380knot$
상승/하강 속도	$1{,}500\,ft/\min$	$1{,}500\,ft/\min$
회피 뱅크각	20°	20°
헤딩	90°	270°

*수평회피의 경우, $380knot$가 사용됨

2.4 시뮬레이션 결과그래프

앞서 정의한 2가지 상황에 대하여 수직, 수평, 그리고 3차원 충돌회피 시뮬레이션을 수행하였다. Fig.9~16은 3차원 충돌회피 시뮬레이션 결과 그래프이다.

2.4.1 Scenario #1

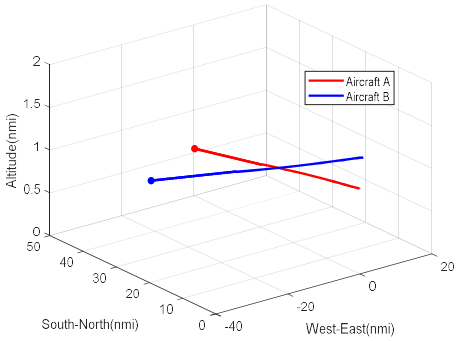

Fig. 9 3-Dim. resolution 3-D view

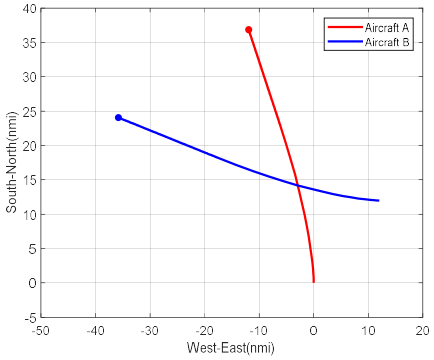

Fig. 10 3-Dim. resolution top view

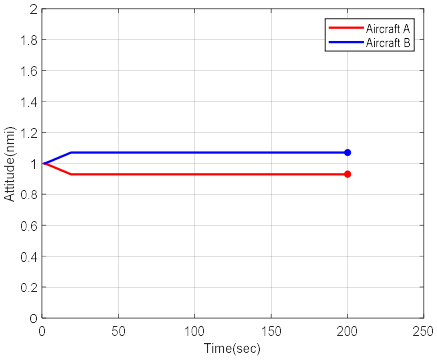

Fig. 11 Altitude of both aircraft

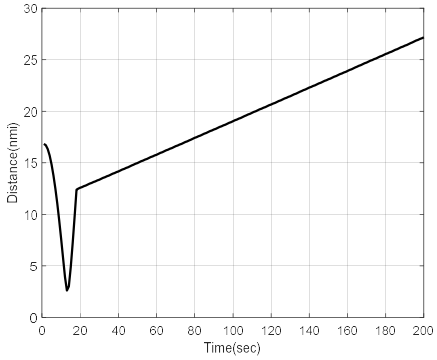

Fig. 12 Range of both aircraft

2.4.2 Scenario #2

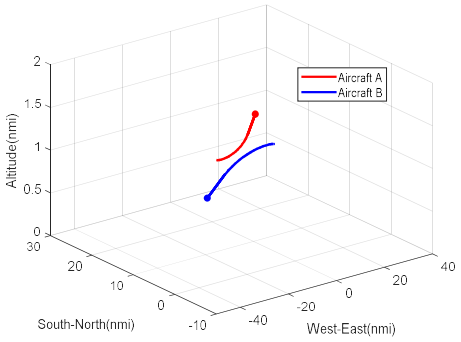

Fig. 13 3-Dim. resolution 3-D view

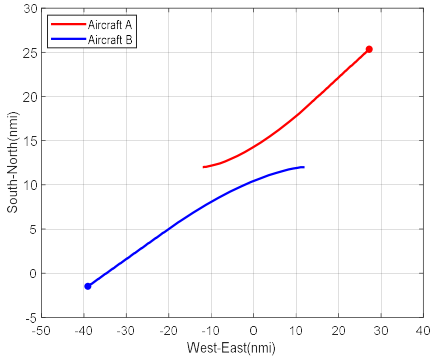

Fig. 14 3-Dim. resolution top view

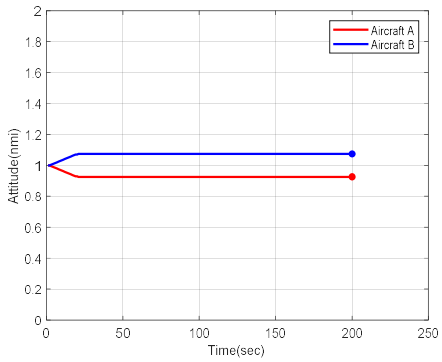

Fig. 15 Altitude of both aircraft

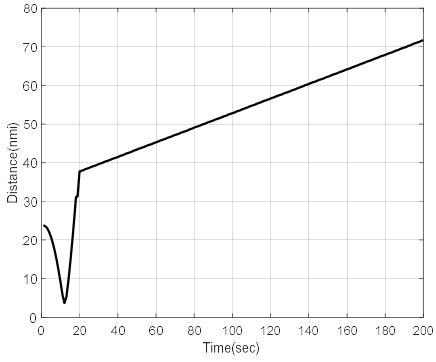

Fig. 16 Range of both aircraft

2.5 MCU 간의 성능 비교

2.5.1 결과데이터: 처리시간

Table 4 Throughput of ATmega2560

위치데이터 출력 함

(unit: sec)	#1	#2
수직	5.03	5.04
수평	601.63	617.3
3차원	76.43	35.29

위치데이터 출력 **않음**

(unit: sec)	#1	#2
수직	0.19	0.19
수평	596.76	612.43
3차원	71.63	30.47

Table 5 Throughput of ARM Cortex M3

위치데이터 출력 함

(unit: sec)	#1	#2
수직	3.13	3.23
수평	241.65	248.95
3차원	23.93	12.31

위치데이터 출력 **않음**

(unit: sec)	#1	#2
수직	0.08	0.09
수평	238.30	245.55
3차원	20.27	8.65

Table 6 Throughput of ARM Cortex M4

위치데이터 출력 함

(unit: sec)	#1	#2
수직	3.13	3.23
수평	69.65	72.19
3차원	9.66	6.22

위치데이터 출력 **않음**

(unit: sec)	#1	#2
수직	0.05	0.05
수평	66.34	68.86
3차원	5.97	2.57

2.5.2 결과데이터: 메모리 사용량

Table 7 Memory usage of MCU

(unit: byte)	ATmega2560	Cortex M3	Cortex M4
수직	3,791	5,876	5,876
수평	3,967	6,076	6,068
3차원	3,897	5,876	5,868

FreeRTOS가 탑재된 3개의 MCU에서 알고리즘 실행파일을 각각 구동했을 때, 처리시간과 메모리 사용량은 Table.4~7과 같다.

처리시간은 소수점 셋째자리에서 반올림하여 표기하였고, 메모리 사용량은 숫자 그대로 표기하였다. MCU가 시뮬레이션을 수행하는지 확인하기 위하여, 매초 두 항공기의 위치데이터를 개발 컴퓨터의 터미널(Terminal) 프로그램 상에 출력하도록 코드(Code)를 작성하였다. Table.4~6을 보면, 위치데이터를 출력할 경우, 모든 회피유형에서 균일하게 ATmega2560에서 약 5초, Cortex M3, M4에서 약 3초가 더 소요되는 것을 알 수 있다. 그러나 터미널 프로그램 상에 위치데이터를 출력하는데 소요되는 시간은 회피 시뮬레이션 시간과 관련이 없으므로, 위치데이터를 출력하는 경우와 출력하지 않는 경우를 구분하여 모두 표기했다.

2.5.3 결과데이터 분석: 처리시간

처리시간 결과데이터에 대하여, 아래의 3가지 변수에 따른 분석가능한 모든 경우의 수를 정리해보면 Table.8과 같다. 총 32가지의 경우의 수가 있지만, 중복을 제외하면 16가지 방법으로 처리시간 결과데이터를 분석할 수 있다.

●출력여부 : 출력O, 출력X

●회피유형 : 수직, 수평, 3차원

●시나리오 : Scenario #1, Scenario #2

Table 8 All possible combination of variables

항목	변수1	변수2	비고
	출력여부	**회피유형**	
1	출력O	수직	
2	출력O	수평	
3	출력O	3차원	분석 수행함
4	출력X	수직	
5	출력X	수평	
6	출력X	3차원	분석 수행함
	출력여부	**시나리오**	
7	출력O	#1	분석 수행함
8	출력O	#2	
9	출력X	#1	분석 수행함
10	출력X	#2	
	회피유형	**시나리오**	
11	수직	#1	
12	수직	#2	
13	수평	#1	
14	수평	#2	
15	3차원	#1	분석 수행함
16	3차원	#2	
	회피유형	**출력여부**	
17	수직	출력O	1번과 동일
18	수직	출력X	4번과 동일
19	수평	출력O	2번과 동일
20	수평	출력X	5번과 동일
21	3차원	출력O	3번과 동일
22	3차원	출력X	6번과 동일
	시나리오	**출력여부**	
23	#1	출력O	7번과 동일
24	#1	출력X	9번과 동일
25	#2	출력O	8번과 동일
26	#2	출력X	10번과 동일
	시나리오	**회피유형**	
27	#1	수직	11번과 동일
28	#1	수평	13번과 동일
29	#1	3차원	15번과 동일
30	#2	수직	12번과 동일
31	#2	수평	14번과 동일
32	#2	3차원	16번과 동일

Table 9 Analysis of Item3

●출력여부 : 위치데이터 출력하는 경우						
●회피유형 : 3차원						
	2560	2560	M3	M3	M4	M4
시나리오	#1	#2	#1	#2	#1	#2
상대 값	3.19	1.47	1.00	0.51	0.40	0.26

■세 MCU의 처리시간은 #2보다 #1에서 더 오래 걸린다.

■동일 시나리오에서 2560-M3 간의 처리시간은 약 2.49배 차이를 보이며, 2560-M4 간의 처리시간은 약 8.56배 차이를 보인다. M3-M4 간의 처리시간은 약 3.44배 차이를 보인다.

■동일 시나리오에서 2560의 처리시간이 가장 오래 걸렸으며 M4의 처리시간이 가장 짧게 걸린다.

Table 10 Analysis of Item6

●출력여부 : 위치데이터 출력하지 **않는** 경우						
●회피유형 : 3차원						
	2560	2560	M3	M3	M4	M4
시나리오	#1	#2	#1	#2	#1	#2
상대 값	3.53	1.50	1.00	0.43	0.29	0.13

■세 MCU의 처리시간은 #2보다 #1에서 더 오래 걸린다.

■동일 시나리오에서 2560-M3 간의 처리시간은 약 3.51배 차이를 보이며, 2560-M4 간의 처리시간은 약 11.90배 차이를 보인다. M3-M4 간의 처리시간은 약 3.38배 차이를 보인다.

■동일 시나리오에서 2560의 처리시간이 가장 오래 걸렸으며 M4의 처리시간이 가장 짧게 걸렸다.

Table 11 Analysis of Item7

●출력여부 : 위치데이터 출력하는 경우			
●시나리오 : #1			
상대 값	2560	M3	M4
수직	1.61	1.00	1.00
수평	192.34	77.28	22.28
3차원	24.44	7.65	3.09

■세 MCU의 처리시간은 수평 회피유형에서 가장 오래 걸렸으며, 수직 회피유형에서 가장 짧게 걸렸다.

■수직 회피유형에서 2560-M3 간의 처리시간은 약 1.61배 차이를 보이며, 2560-M4 간의 처리시간은 약 1.61배 차이를 보인다. M3-M4 간의 처리시간은 유사하다.

■수평 회피유형에서 2560-M3 간의 처리시간은 약 2.49배 차이를 보이며, 2560-M4 간의 처리시간은 약 8.63배 차이를 보인다. M3-M4 간의 처리시간은 약 3.47배 차이를 보인다.

■3차원 회피유형에서 2560-M3 간의 처리시간은 약 3.19배 차이를 보이며, 2560-M4 간의 처리시간은 약 7.91배 차이를 보인다. M3-M4 간의 처리시간은 약 2.48배 차이를 보인다.

Table 12 Analysis of Item9

●출력여부 : 위치데이터 출력하지 **않는** 경우			
●시나리오 : #1			
상대 값	2560	M3	M4
수직	2.27	1.00	0.63
수평	7104.25	2836.88	789.80
3차원	852.74	241.26	71.01

■세 MCU의 처리시간은 수평 회피유형에서 가장 오래 걸렸으며, 수직 회피유형에서 가장 짧게 걸렸다.

■수직 회피유형에서 2560-M3 간의 처리시간은 약 2.27배 차이를 보이며, 2560-M4 간의 처리시간은 약 3,60배 차이를 보인다. M3-M4 간의 처리시간은 약 1.59배 차이를 보인다.

■수평 회피유형에서 2560-M3 간의 처리시간은 약 2.50배 차이를 보이며, 2560-M4 간의 처리시간은 약 8.99배 차이를 보인다. M3-M4 간의 처리시간은 약 3.59배 차이를 보인다.

■3차원 회피유형에서 2560-M3 간의 처리시간은 약 3.53배 차이를 보이며, 2560-M4 간의 처리시간은 약 12.01배 차이를 보인다. M3-M4 간의 처리시간은 약 3.40배 차이를 보인다.

Table 13 Analysis of Item15

●회피유형 : 3차원		
●시나리오 : #1		
상대 값	위치데이터를 출력하는 경우	
2560	M3	M4
3.77	1.18	0.48

상대 값	위치데이터를 출력하지 **않는** 경우	
2560	M3	M4
3.53	1.00	0.29

■위치데이터 출력에 소요되는 시간은 2560에서 약 5초, M3, M4에서 약 3초이다.

■위치데이터를 출력하는 경우, 2560-M3 간의 연산시간은 약 3.19배 차이를 보이며, 2560-M4 간의 연산시간은 약 7.85배 차이를 보인다. M3-M4 간의 약 2.46배 차이를 보인다.

■위치데이터를 출력하지 않는 경우, 수평 회피유형에서 2560-M3 간의 연산시간은 약 3.53배 차이를 보이며, 2560-M4 간의 연산시간은 약 12.17배 차이를 보인다. M3-M4 간의 연산시간은 약 3.45배 차이를 보인다.

2.5.4 결과데이터 분석: 메모리 사용량

메모리 사용량 결과데이터에 대하여, 분석하여 정리해보면 Table.14와 같다.

Table 14 Memory usage of MCU

(unit: byte)	2560	M3	M4
수직	3,791	5,876	5,876
수평	3,967	6,076	6,068
3차원	3,897	5,876	5,868
평균	3,885	5,943	5,937

■동일 회피유형에서 M3, M4의 메모리 사용량은 비슷하며, 2560의 메모리 사용량이 더 적다.

3. 결 론

본 논문에서는 C++ 언어를 기반으로 항공기 충돌회피 알고리즘을 구현하고, 동일한 알고리즘과 운영체계를 탑재한 3종류의 MCU 간 성능비교를 수행하였다.

사용된 충돌회피 알고리즘은 TCAS 수직회피 알고리즘, Andrew Trapani 논문의 수평회피 알고리즘, 김영래 논문의 3차원회피 알고리즘이다. MCU는 8bit 프로세서인 ATmega2560, 32bit 프로세서인 ARM Cortex-M3, ARM Cortex-M4를 사용하였다. 또한, 신속한 의사결정(Time-critical)이 요구되는 실제 항공기 운용 환경을 고려하여, 운영체계는 FreeRTOS를 선택하였다. MCU 간의 성능비교는 동일한 알고리즘 실행파일을 각 MCU에서 구동하였을 때 측정한 처리시간과 메모리 사용량을 기준으로 수행하였다.

(1) 처리시간의 경우, 분석가능한 모든 경우의 수를 정리하여 체계적으로 결과데이터를 분석해 볼 수 있었다.

위치데이터를 출력하는 경우가 출력하지 않은 경우에 비해 더 많은 시간이 소요되는 이유는 출력을 위한 절대시간이 필요하기 때문이다. (ATmega2560에서 약 5초, Cortex M3, M4에서 약 3초로 측정되었다.)

ATmega2560에 비해 Cortex M3, M4가 모든 회피유형과 시나리오에 대하여 더 빠른 처리시간을 보였다. 그 이유는 8bit 프로세서와 32bit 프로세서의 차이로, 후자가 더 넓은 Bandwidth와 더 빠른 Clock Speed를 갖추고 있기 때문이다.

수직, 수평, 3차원 회피유형 중에서 수직회피의 처리시간이 가장 짧게 걸렸으며, 수평회피의 처리시간이 가장 오래 걸렸다. 그 이유는 수평회피의 경우 배열계산을 통해 알고리즘 연산이 더욱 복잡해졌기 때문이다.

(2) 메모리 사용량의 경우, 동일 회피유형에서 Cortex M3, M4의 메모리 사용량은 비슷했으며, ATmega2560의 메모리 사용량이 더 적은 경향을 보였다.

그 이유는 32bit 프로세서가 8bit 프로세서에 비해 명령어 집합(Instruction Set)이 더 많고, 동일한 변수더라도 자료형(Data Type)이 더 크며, 메모리 주소(Memory Address) 구성이 더 복잡하기 때문이다. 따라서 비교적 구조가 단순한 ATmega2560에서 메모리 관리(Memory Management)가 보다 효율적으로 수행될 수 있었다.

오늘날의 민간 항공기는 오직 수직방향으로의 회피기동을 통해 충돌회피를 수행하고 있다. 수직회피 알고리즘은 단순하기 때문에 처리시간(반응)이 빠르고 위험수준이 상대적으로 낮다는 장점을 가진다. 하지만, 동시에 항공기 분리까지의 소요시간이 상대적으로 오래 걸리는 한계가 있다.[5] 훗날 MCU 성능이 향상되어 수직회피와 3차원회피 간의 처리시간 격차가 더욱 줄어들게 된다면, 민간 항공기에도 3차원회피 알고리즘이 적용될 수 있을 것으로 기대된다.

후 기

본 연구는 한국항공대학교 항공우주 및 기계공학부 종합설계 과목을 통해 작성되었습니다. 그동안 정성을 다해 지도해주신 이상철 교수님께 감사드립니다. 그리고 연구에 도움주신 조교님들에게도 감사의 말씀을 드립니다.

참고문헌
(References)

(1) ACI, “ACI releases the World Airport Traffic Forecasts 2017-2040”, https://goo.gl/Eagvna

(2) Eurocontrol, “Free Route Airspace”, http://www.eurocontrol.int/articles/free-route-airspace

(3) James K. Kuchar, Ann C. Drumm, 2007, The Traffic Alert and Collision Avoidance System, Lincoln Laboratory Journal, Vol. 16, No.2.

(4) Andrew J. Trapani, 2008, “Performance Analysis of a Horizontal Separation Assurance Algorithm for Short-range Conflict Detection and Resolution”, Master of science in computer engineering graduated school of santa cruz, university in california, pp. 1~24.

(5) Kim, Young-Rae, 2012, “A Development and Verification of 3-D Resolution Algorithm for Aircraft Collision Avoidance”, Dept. of Aerospace and Mechanical Engineering Graduated School of Korea Aerospace University, pp. 4~31.