공학석사학위 논문

스마트폰 기반 비행제어 컴퓨터를 위한 가변적 시간단계 제어기 연구

Development of Variable Time Step Controller for Smartphone-based Flight Control System

2021년 2월

인하대학교 대학원

항공우주공학과

박 승 현

공학석사학위 논문

스마트폰 기반 비행제어 컴퓨터를 위한 가변적 시간단계 제어기 연구

Development of Variable Time Step Controller for Smartphone-based Flight Control System

2021년 2월

지도교수 이 학 태

이 논문을 석사학위 논문으로 제출함

이 논문을 박승현의 석사학위논문으로 인정함.

2021년 1 월 6 일

주심 최 기 영 (인)

부심 유 창 경 (인)

위원 이 학 태 (인)

초 록

무인항공기의 자동비행을 위하여 꼭 필요한 비행제어컴퓨터는 여러 센서 데이터들을 획득하여, 제어시스템 개발자가 의도한대로 임무를 따르도록 운용 및 명령을 인가하는 역할을 한다. 하지만 기존의 비행제어 컴퓨터를 활용한 비행제어시스템은 하드웨어적으로 구성하기 복잡하고, 하드웨어에 따라 소프트웨어가 상이하여 개발하는 데에 어려움이 있다. 따라서 본 논문에서는 하드웨어가 모두 통합되어 있고, 표준적인 개발 시스템이 갖춰진 스마트폰 기반 비행제어시스템을 제안한다. 그러나 스마트폰은 기기의 제어를 위한 전용 플랫폼이 아니기 때문에, 제어 루프의 실시간성을 보장하지 않는다. 이 문제를 해결하기 위해 가변적 시간단계 제어기를 구성하였다. 5 자유도 시뮬레이션을 이용하여 가변적 시간단계 제어기를 검증하였다. 실제 스마트폰 기반 비행제어 하드웨어와 앱을 구성하여 무인선, 무인차량을 이용하여 단계적 개발을 진행하고, 무인항공기에 탑재하여 부분적 자동 비행을 수행하는 데에 성공하였다. 이러한 가변적 제어기를 탑재한 스마트폰 기반의 비행제어 시스템은, 특히 향후 영상과 통신 기능까지 사용하게 될 경우, 범용적인 무인이동체의 제어시스템으로 유용하게 활용될 수 있을 것이다.

ABSTRACT

A flight control system which is a crucial part of a UAV that enables automatic flights, it gathers data from various sensors then operates and commands to follow the specific missions as intended by the developer of the control system. However, flight control systems incorporating conventional flight controller is quite difficult to properly organize its hardware due to its complexity, and it is difficult to develop because of the differences in software of the hardware. Thus, in this paper, a Smartphone-based Flight Control System is proposed with integrated hardware and standardized development system. However, since smartphones are not a platform designed as a controller, it does not guarantee the real time operation of the control loop. To solve this problem, the Variable Time Step Controller was developed. The Variable Time Step Controller was tested using the 5 DOF Simulation. The smartphone-based flight control hardware and software were incrementally developed using an unmanned boat and an unmanned car, and the partial automatic flights were possible with an unmanned plane. The Smartphone-based Flight Controller with implementation of the Variable Time Step Controller, especially with the future development of video transmission and communication, will be useful as a universal control system of unmanned vehicles.

목차

표 목차

그림 목차

기호 목록

기호	설명
f	측정된 가속도 센서 데이터
ω	측정된 자이로 센서 데이터
f_{avg}	가속도 센서 데이터 평균값
$(p,q,r)_{avg}$	자이로 센서 데이터 평균값
f^b, ω^b	가속도/자이로 센서 바이어스
s_a	가속도 스케일
f_b	바이어스로 보정된 가속도 센서 데이터
ω_b	바이어스로 보정된 자이로 센서 데이터
C_n^b	n-frame to b-frame 방향코사인 행렬
e, R_0	지구의 이심률, 반지름
R_m, R_t	지구 장반경, 단반경
g^n	지구 중력가속도
R_{mm}, R_{tt}	지구 장반경, 단반경을 위도로 편미분한 값
Ω	지구 자전각속도
ω_{ie}^n	회전벡터 e-frame w.r.t i-frame, n-frame 사영
ω_{en}^n	회전벡터 n-frame w.r.t e-frame, n-frame 사영
ω_{in}^n	회전벡터 n-frame w.r.t i-frame, n-frame 사영 지구자전 각속도, 위도/경도 변화율
ω_{nb}^b	회전벡터 b-frame w.r.t n-frame, b-frame 사영
ω_{ib}^b	회전벡터 b-frame w.r.t i-frame, b-frame 사영 자이로 센서 측정치
$\dot{L}, \dot{l}, \dot{h}$	위도/경도/고도 변화율
$\dot{V^n}$	속도 벡터 변화율
$\dot{C_b^n}$	방향 코사인 변화율

δx	상태 (State) 오차
$\delta L, \delta l, \delta h$	위도/경도/경도 오차값
$\delta_N, \delta_E, \delta_D$	NED 방향 속도 오차값
ϕ_N, ϕ_E, ϕ_D	NED 방향 자세 오차값
$\nabla_x, \nabla_y, \nabla_z$	x, y, z 방향 가속도 센서 바이어스
ϵ_x	x, y, z 방향 자이로 센서 바이어스
P_k	공분산 (error covariance)
P_k^-	예측 공분산
F_k	시스템모델 행렬
Q	예측 노이즈 공분산
R	측정 노이즈 공분산
z_k	측정값
Llh^{INS}	INS로부터 계산된 위도/경도/고도
Llh^{GPS}	GPS로 측정된 위도/경도/고도
V_{NED}^{INS}	INS로부터 계산된 NED방향 속도
V_{NED}^{GPS}	GPS로부터 계산된 NED방향 속도
x_{INS}	INS 방정식 결과값
x	상태 (State) 벡터
H_{CMD}	고도 명령
P_{gain}	비례 게인
I_{gain}	적분 게인
D_{gain}	미분 게인
Err_{ROCD}	상승/하강률 오차
Err_{ROCD_I}	상승/하강률 적분항
Err_{ROCD_D}	상승/하강률 미분항
$dt_{variable}$	가변 시간단계
$ROCD$	상승/하강률
$ROCD_{pre}$	이전 단계 상승/하강률

1. 서론

1.1. 연구배경

최근 경비, 소방, 농업, 배송, 미디어 제작 등 여러 분야에서 드론이나 무인기의 수요가 늘어나고 있고, 이에 드론과 무인기에 대한 연구와 개발이 활발하게 이루어지고 있다.

드론과 무인기에는 자동비행을 위하여 비행제어 컴퓨터가 탑재되는데, 이 비행제어 컴퓨터는 GPS를 통한 위치 정보와 IMU를 통한 자세 정보를 토대로, 유저의 명령에 따라 자세를 제어하고 임무를 수행하게 된다. 하지만 일반적인 비행제어 컴퓨터가 포함된 시스템을 사용한다면, 부착된 각각의 센서가 필요하는 전압이 다른 경우도 있고, 센서데이터를 후처리 해야 하는 번거로움이 있으며, 공간적으로도 구성하기가 복잡하여 무인기 내부공간을 효율적으로 구성하기가 어려운 경우가 있다.

우리가 일상적으로 사용하고 있는 스마트폰에도 무인기에 필요한 기본적인 센서들이 탑재되어 완성품으로 구매할 수 있기 때문에 복잡한 시스템을 간단히 대체할 수 있고, 스마트폰을 교체하는 주기도 2년쯤으로 매우 짧아서 사용하지 않는 고성능의 스마트폰도 주변에서 쉽게 구할 수 있다. 따라서 무인기의 비행제어시스템으로써 스마트폰이 가능성이 있다고 생각되어 연구를 진행하였다.

1.2. 스마트폰 성능과 센서종류

스마트폰으로 비행제어기를 대체가능한지에 대해 가장 중요한 것은 스마트폰이 연산능력과, 비행에 필요한 센서의 유무이다.

스마트폰의 연산능력에 가장 큰 영향을 미치는 모바일 프로세서이다. 모바일 프로세서 제조사 중 하나인 퀄컴 (Qualcomm)사에 의하면[1], 모바일 프로세서는 컴퓨터와는 다르게 여러가지 유닛이 함께 구성되어 있다. CPU (Central Processing Unit)은 즉각적인 연산을 담당하고, 디바이스 전체에 신호를 보내고 받는 역할을 한다. GPU (Graphics Processing Unit)은 게임관련 연산을 담당하고, 비디오가 픽셀처럼 깨져서 보이지 않도록 돕는 역할을 한다. 이와 밖에도 오디오 관련 프로세싱을 하고, RF&LTE 등의 네트워크 관련 유닛도 포함되어 있다.

GadgetVerse 사이트에 의하면[2], 프로세서의 성능을 여러가지 방법으로 성능을 비교하는 프로그램인 Geekbench 4를 사용했을 때, 삼성 갤럭시 S20에 사용되는 Qualcomm SM8250 Snapdragon 865 모바일 프로세서는 2017에 나온 Intel사의 i5 프로세서인 Intel Core i5-7500과 싱글코어와 멀티코어 성능이 거의 비슷하다고 한다.

Android Developers에서는 안드로이드 스마트폰의 센서는 크게 3가지의 카테고리로 나뉜다고 명시했다[3]. 첫번째로 움직임 감지 센서는 3개의 축을 따라 가속력과 회전력을 측정한다. 움직임 감지 센서에는 가속도계, 중력 센서, 자이로스코프, 회전 벡터 센서가 포함된다. 두번째로 환경 센서는 주변 기온, 압력, 조도,

습도 같은 환경 매개변수를 측정하며, 기압계, 광도계 온도계가 포함된다. 마지막으로 위치 센서는 기기의 물리적 위치를 측정하며, 이 카테고리에는 방향 센서와 자기계가 포함된다.

1.3. 연구 목표 및 연구 진행방향

본 연구는 스마트폰을 비행제어시스템으로 사용할 때 발생할 수 있는 문제점들을 해결하는 대에 목표를 두었다. 스마트폰을 비행제어시스템으로 사용할 때 문제가 되는 점은 스마트폰이 직접적으로 모터 혹은 서보모터를 구동 시키는 PWM (Pulse Width Modulation)신호를 생성하지 못한다는 것이다. 또 다른 문제는 스마트폰의 GPS주기는 1Hz인데, 이것은 비교적 고속인 이동체에 탑재할 때 문제가 생길 수 있다. 그리고 저가의 계산능력이 좋지 않은 스마트폰의 경우, 앞서서 말한 GPS의 주기 문제를 해결하려고 할 때 필요한 필터 계산과, 전체적인 명령생성계산에 시간이 오래 걸릴 수 있다. 또한 어플의 기능이 많아질수록 어쩔 수 없이 어플 실행이 무거워지고, 이에 따라 제어 루프의 실시간성을 보장하기 어려워진다. 따라서 가변적 시간단계 제어시스템이 필요하다고 판단되어 연구하고, 앞서 말한 문제점들에 대하여 해결책을 제시하고 검증한다.

2. 스마트폰 기반 제어시스템 개발

2.1. 제어시스템 하드웨어 구성도[4]

스마트폰만으로는 비행 이동체의 조종면을 움직이는 PWM 신호를 만들어내지 못한다. 따라서 스마트폰의 명령을 받아서 모터, 서보모터와 연결되어 PWM 신호를 줄 수 있는 Sparkfun사의 IOIO 보드를 사용하였다. IOIO보드는 스마트폰과 USB-OTG (On the Go) 혹은 블루투스 동글을 부착하여 블루투스로 연결이 가능하다.

보통 자동비행 기능을 가지고 있는 항공기는 FCS (Flight Control System) 자체에서 EP(External Pilot)/Autopilot을 정하여, 문제가 생겼을 경우에 FCS로 명령을 주어 EP 비행을 한다. 이는 FCS가 어떠한 상황에서도 꺼지거나 문제가 안 생긴다는 가정하에 사용된다. 이 방법이 주로 쓰이는 이유는 EP 비행일 때도, 조종기의 명령에 Curve값이나 Gain값을 넣어 조종하기에 용이하기 때문이다. 하지만 스마트폰을 탑재하여 비행을 했을 때는 혹시 모를 어플리케이션 충돌이나, 외부적인 요인에 의해 문제가 생길 수 있다. 따라서 자동비행 중 비상사태를 대비하여 조종사가 직접 이동체에 명령을 줄 수 있도록 시스템을 구성하였다. 이를 위해 Multiplexer (MUX 혹인 MPX)를 사용하였다. RC 조종기에서 신호를 줘서 Multiplexer의 지정된 채널을 작동시면 완전 수동 혹은 자동 전환이 가능하다. 수동모드일때는 RC 리시버에서 생성된 PWM신호가 서보모터와 ESC에 직접 전달되어 조종면과 모터를 구동 시킨다. 자동모드 일때에는 IOIO보드에서 생성된

PWM값이 전달된다.

스마트폰과 IOIO보드는 블루투스를 통해 연결된다. IOIO보드 자체에는 블루투스 모듈이 없지만, 블루투스 동글을 통하여 부착하면 블루투스 지원이 가능하다. IOIO보드와 연결 방법에는 OTG 케이블로 직접 스마트폰과 연결하는 방법도 있지만, 최근 안드로이드 OS와 IOIO보드의 호환성 문제로 OTG케이블을 이용한 유선 연결이 불가하다. 필연적인 이유가 있긴 하지만 스마트폰과 IOIO보드가무선으로 연결되어, 이 덕분에 하드웨어 배치하기가 용이하다.

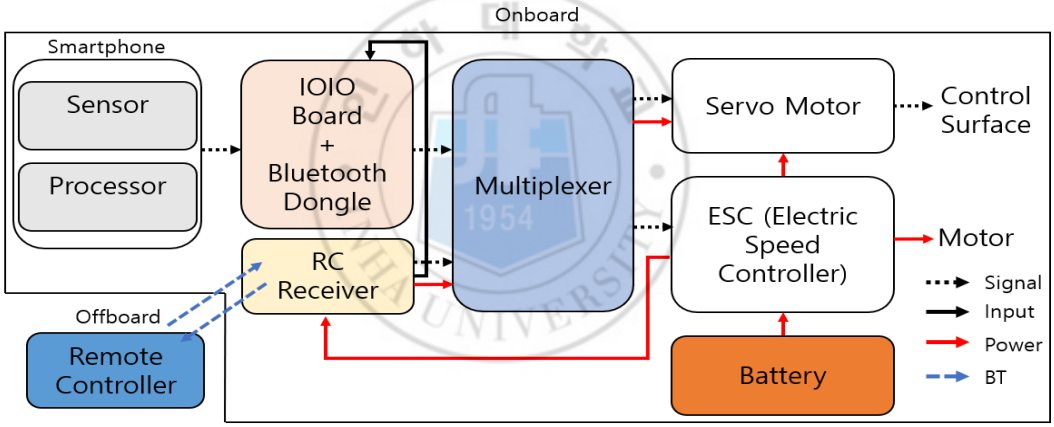

그림 1. 스마트폰 제어시스템 하드웨어 구성도

2.2. 어플리케이션

스마트폰에서 기본적인 계산과 제어명령을 하기위해서 안드로이드 어플리케이션을 개발하였다. 개발에는 Android Studio를 사용했고, 사용 언어는 JAVA를 사용했다.

일반적으로 이동체에 탑재되어 사용되는 센서들과 달리, 안드로이드에서는 Parsing 단계를 거치지 않아도 쉽게 각종 센서 데이터에 접근이 가능하다. 이에 대한 내용은 Android Developers에 자세하게 안내되어 있다. 간단히 원하는 센서들을 Manager 클래스에 등록(Register)하고, 원하는 센서값에 변화가 생기면 값을 받아오는 형식으로 데이터를 추출할 수 있다.

Location Manager 클래스에서는 위도, 경도, 진행방향, 지상속도 데이터를 불러올 수 있으며, Sensor Manager에서는 각속도, 가속도, 기압, 중력, 지자기장 데이터들을 불러올 수 있다.

본 논문에서 사용하는 스마트폰은 삼성 갤럭시 S10이며, 해당 스마트폰의 센서들의 주기는 다음과 같다.

표 1. 스마트폰 센서 주기

센서	GPS	Accel	Gyro	Mag	Baro
주기	1Hz	100Hz	100Hz	50Hz	10Hz

2.2.1. 안드로이드에서의 Body축 정의

Android Developers에 의하면[5], 안드로이드에서는 Body Frame의 X, Y, Z축을 다음과 같이 정의한다. 스마트폰 액정이 하늘을 바라보고, 핸드폰 상단이 앞을 가

리키게끔 바닥에 놓고 볼 때를 정방향이라고 할 때, X축은 오른쪽을 향하는 축, Y축은 핸드폰 상단을 향하는 축, Z축은 핸드폰 액정 뚫고 나오는 방향이다. 모든 센서값의 방향은 안드로이드에서 정의된 축과 방향에 따라 측정된다.

안드로이드에서 정의된 Body Frame은 항공기의 Principle Axes Body Frame과 방향이 다르다. 만약 스마트폰이 이동체에 정방향으로 탑재된다면 항공기에 사용되는 Body Frame과 달라지기 때문에 중요하다. 항공기에서 Principle Axes는 X축이 Roll Axis로 tail로부터 nose까지 이루어지는 축이고, Y축은 Pitch축으로 왼쪽 날개에서 오른쪽 날개를 향한 방향이다. Z축은 항공기 천장에서 배 밑 방향으로 이루어진 축이다[6].

축 방향이 다르기 때문에 안드로이드에서 측정되는 모든 센서의 축을 바꿔주어야 한다. 안드로이드에서 정의된 축에서 항공기의 Principle Axes로 바꾸기 위해서는 z방향으로 -90도, y방향으로 180도 회전을 면 된다. 스마트폰에서 측정된 데이터가 [A, B, C]일 때 [B, A, -C]로 바뀌준다.

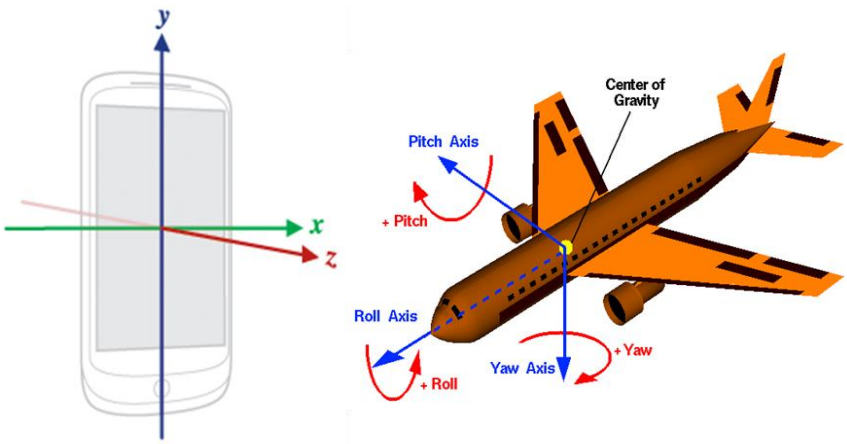

그림 2. 스마트폰의 축 정의와 비행기의 축 정의 비교

2.3. 개발 및 테스트

안드로이드 스마트폰을 제어시스템으로 사용가능성을 확인하기 위하여, 다양한 이동체에 탑재하여 테스트를 진행했다. 각 이동체 마다 필요한 기능이 무엇인지 알아보고, 개발 후 테스트하는 방식으로 진행하였다.

2.3.1. RC보트를 이용한 초기 개발

RC보트로는 경로점(Waypoint) 주행, 센서데이터와 계산된 데이터를 저장하는 로깅 기능, 현장에서 UI를 가지고 바로 PID 게인을 조절할 수 있는 어플리케이션 UI를 개발하는 것을 목적을 두고 개발을 진행하였다. 초기 테스트에 RC보트가 선택된 이유는 비행기보다 빠르지 않고, 추락하지도 않기 때문에 위험하지 않고, 비교적 가까운 거리에서 이동체가 어떻게 반응하는지 맨눈으로 확인 가능하기 때문이다.

경로점 주행을 위하여 구글 Maps API를 사용하여 경로점의 위치를 정한다. 그림 3에서 왼쪽 그림의 하단에 있는 Map 버튼을 누르게 되면 구글지도 창이 뜨게 된다. 구글지도에서 원하는 지점을 길게 누르면 경로점이 선택되고, 그림 3 오른쪽 그림 왼쪽 하단에 있는 Back 버튼을 누르게 되면 선택된 경로점들의 위도 경도는 intent기능을 사용하여 메인 엑티비티로 불러오게 된다. 메인 엑티비티에서 현재 GPS 위치와 경로점의 위치를 비교하여 명령을 계산하고 IOIO 보드로 명령을 인가한다.

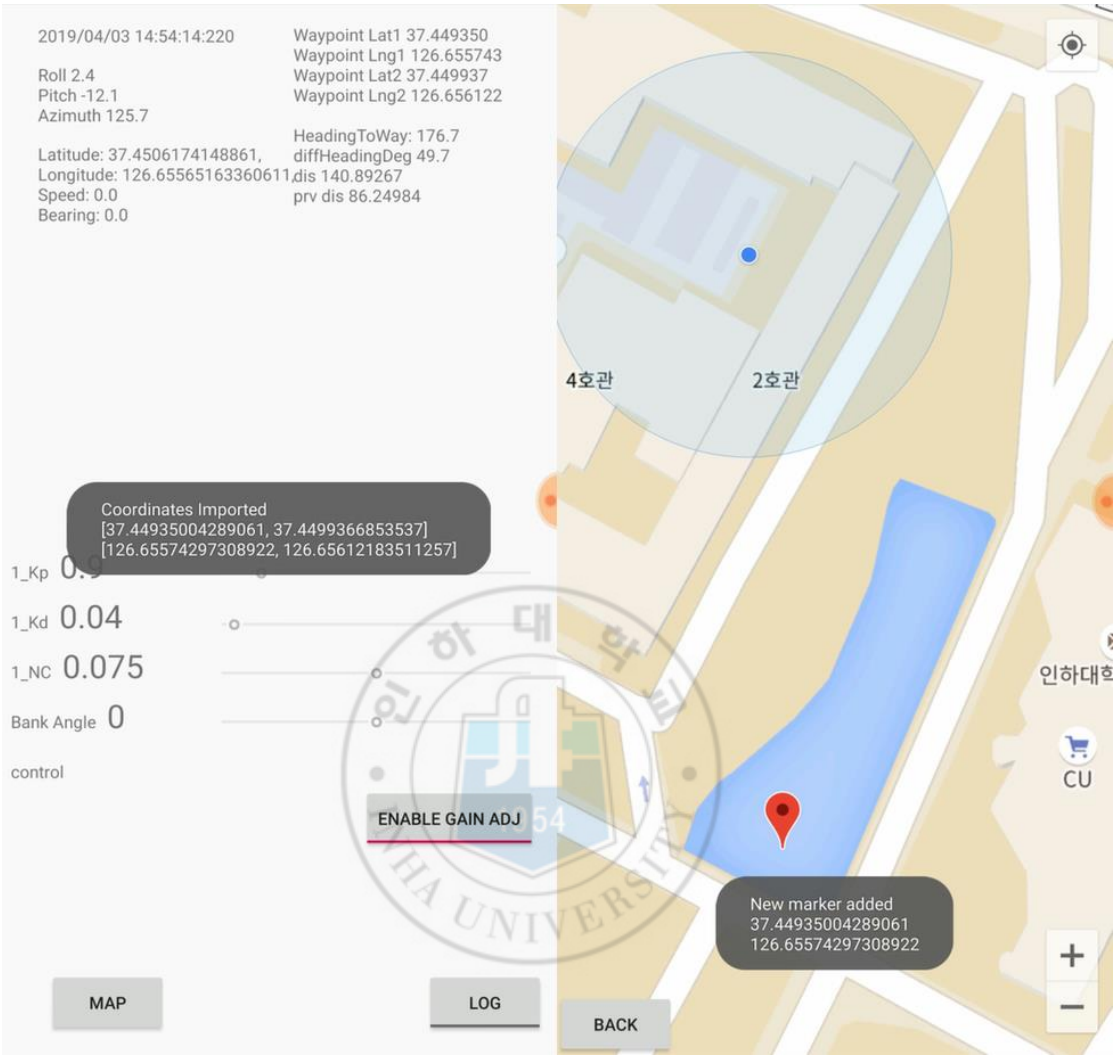

그림 3. 어플리케이션 화면 (좌), 구글지도 화면(우)

RC보트는 러더로 헤딩(방향)을 정한다. 이를 위하여 타겟 헤딩을 계산하고, 현재 헤딩과 비교가 필요하다. 타겟 헤딩은 현재 위치와 경로점 위치로 계산되고, 계산된 타겟 헤딩은 스마트폰의 지자계에서 나오는 헤딩과 비교하게 된다. 그 이후 비례제어기를 통해 명령을 생성한다. 이때 비례제어기의 게인은 어플리

케이션 UI상에서 조절 가능하다.

현재 위치와 경로점과의 거리는 안드로이드 Location Manager에서 distanceTo 메소드를 사용하여 계산한다. distanceTo를 사용하여 계산된 목표 경로점과의 거리가 5m보다 작을 경우, 다음 경로점을 목표로 삼는다.

실험 결과를 저장하기 위하여 로깅 기능을 추가하였다. UI에 지정된 버튼을 클릭하게 되면, 핸드폰의 내장 저장소에 폴더와 csv파일이 생성되며, 설정된 주기마다 데이터가 저장된다.

그림 4. 개발에 사용된 RC보트

테스트는 인하대학교 본관 앞 분수대에서 진행하였으며, 구글지도에서 두개의 경로점이 선택되었다. 그림 4와 같은 RC보트를 사용했으며 모터는 수동, 러더는 자동모드로 설정하였다.

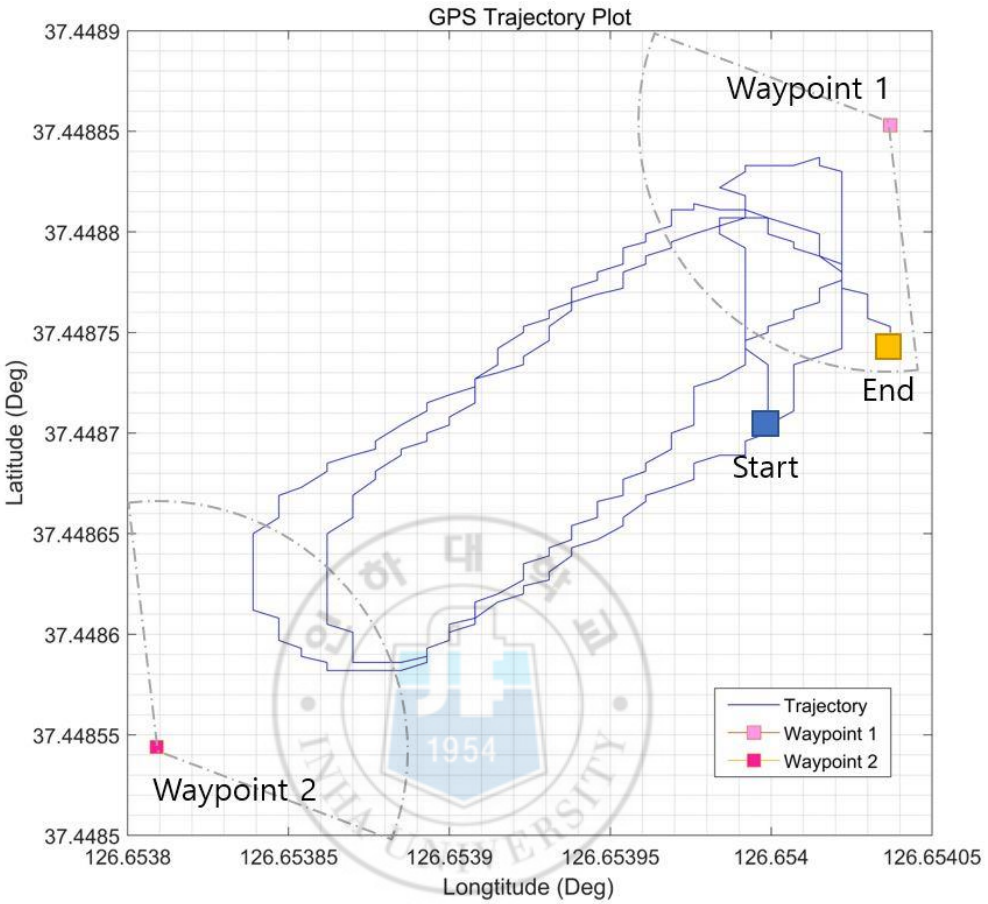

그림 5. RC보트 테스트 궤적

스마트폰 상의 로깅 기능을 사용하여 측정된 GPS 위치정보를 그린 그림은 그림 5와 같다. 그림을 보면 지정된 두개의 경로점 사이를 성공적으로 두차례 왕복하는 것을 확인할 수 있다. GPS센서의 주기가 1Hz이기 때문에 경로가 촘촘하지 않고, 계단형을 띄는 것을 확인할 수 있다.

2.3.2. RC비행기를 이용한 개발

RC비행기를 사용한 개발에서는 자동비행에 필요한 시스템/하드웨어 구성과 각종 센서를 사용하여 자동비행을 해보고 가능한지 확인해보는 것에 목적을 두었다.

추후에 System Identification을 위하여 IOIO보드의 라이브러리의 openPulseInput 함수를 사용하여 지정된 핀의 PWM 신호를 기록할 수 있는 기능을 추가하였다. openPulseInput 함수를 사용하면 0.001부터 0.002의 Pulse의 Duration이 측정된다. 서보모터와 모터에 전달되는 신호케이블을 IOIO보드에 연결하면 되고, 총 6개의 핀에서 PWM 신호를 읽어낼 수 있다. 이 기능으로 수동비행에서 서보모터와 모터에 인가한 PWM 신호를 확인할 수 있고, 인풋과 아웃풋을 비교하여 System Identification이 가능하다.

보트와 비교하여 사용하는 채널의 개수가 증가했다. 보트는 러더, 쓰로틀과 Multiplexer의 자동/수동을 선택하는 채널까지 3채널이였지만, 비행기는 에일러론, 엘레베이터, 쓰로틀, 러더에 Multiplexer까지 총 5채널이 필요하다.

그림 6. 개발에 사용된 RC비행기

2.3.3. 롤각 제어기

간단한 자동비행을 위하여 롤각 제어기로 일정한 각도를 유지하면서 간단한 선회 비행, Bank Angle Hold를 할 수 있도록 개발하였다. UI상에서 선회각도를 슬라이더로 설정하면, 설정각도와 현재 스마트폰의 각도와 비교하여 조종면 명령이 생생된다[7]. 롤각제어기의 구성은 다음과 같다.

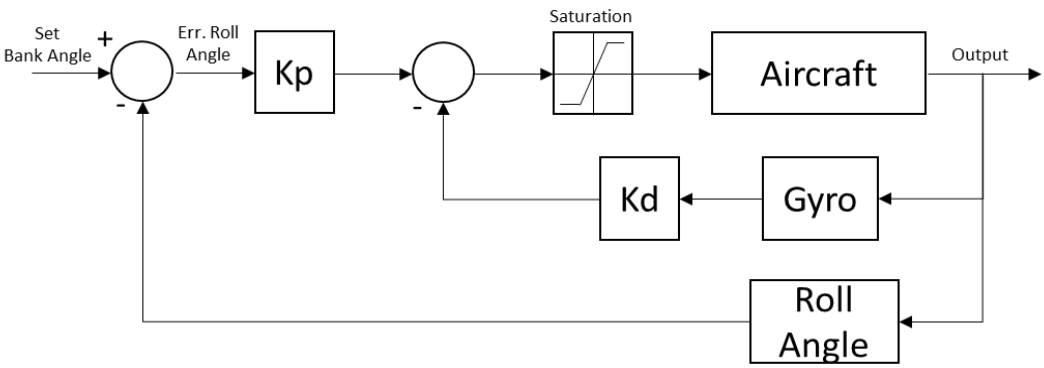

그림 7. 롤각제어기 구조

2.3.4. 비행 사고

롤각 제어기를 테스트하는 도중, 조종사가 수동 비행 중 의도치 않은 롤이 발생하여 비행기가 추락하는 경우가 있었다.

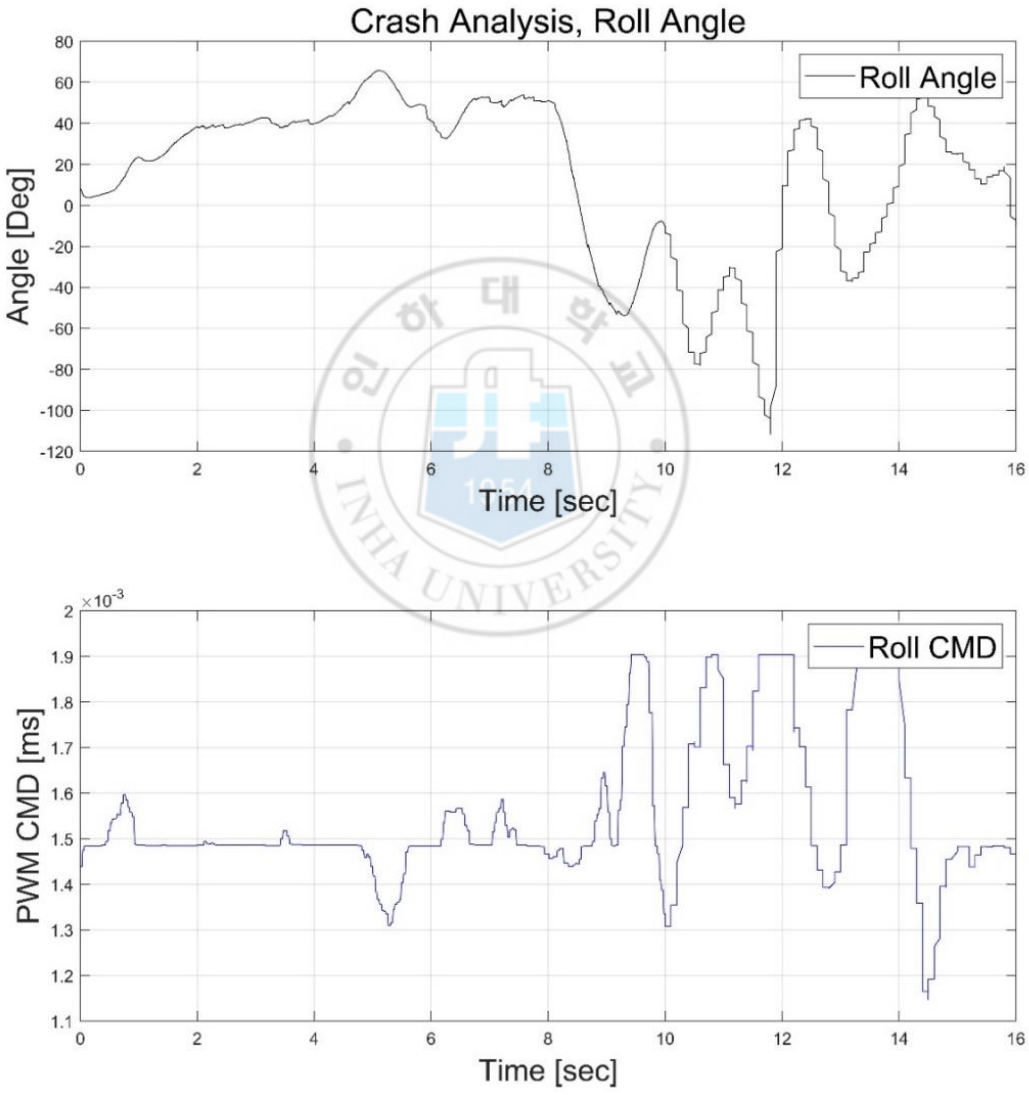

그림 8. 비행사고 데이터 분석

자동비행을 위하여 적정한 고도로 높이기 위하여 수동비행을 하다가 생긴 사고로, 그대로 자동비행을 했다가 같은 문제가 나타날 수 있다고 판단하여, 이로 인해 몇달간 비행테스트를 중단하고 원인규명을 하였다.

그림 8은 기록된 기체의 롤각과 에일러론의 입력을 나타낸 것이다. 약 8초에 인가되지 않은 급격한 롤이 생기고, 그 이후에 조종사가 다시 기체를 제어하려고 하지만 금세 다시 급격한 롤이 두번이나 더 생기는 것을 확인할 수 있다.

확실한 문제규명을 위해 외부 전문가를 초청하여 조언을 받았다. 먼저 의심되는 부분은 PWM로깅 기능 때문에 복잡해진 배선이다. 멀티플랙서와 서보모터로 연결된 케이블의 피복을 벗기고, 다른 케이블을 추가적으로 납땜하여 IOIO에 꽂아 신호를 측정했다. 이로 인해, 케이블이 매우 복잡하게 되었고, 합선이 일어나 이상 신호가 섞여 문제가 발생했다는 것이다. 이에 전문가가 내린 해결책은 케이블을 대체할 회로를 제작해서 문제를 해결하는 것이 였다.

2.3.5. TAIO 보드[4]

전문가의 자문을 받아 회로를 제작하였다. 전원 관련된 배선을 두껍게 하고, 다층 PCB 보드로 구성하여 내부에 전원과 그라운드층을 넣는 식으로 구성하였다. PWM 신호는 그라운드와 전압의 차이로 High와 Low를 구별하고, 이것으로 Duty Cycle을 나타낸다. 다층 PCB 보드로 확실하게 그라운드와 전압을 나누게 된다면, 노이즈를 줄일 수 있는 효과가 있다고 한다.

보드를 제작함으로써 케이블이 복잡하게 연결되었던 것을 모두 없어지고, 보

드에 부품을 부착하는 것으로 대체하였다. 따라서, IOIO보드, RC리시버, Multiplexer 두개를 꽂으면 작동 가능하다.

추가적인 기능으로는 점퍼로 하드웨어적으로 자동/수동과 완전수동을 고를 수 있게 되었다. Multiplexer에서 나오는 신호가 자동/수동 신호이며, RC리시버에서 곧바로 나오는 신호가 완전수동이다. 또한 점퍼로 보드의 전원으로 ESC전원을 사용할 것인지 아니면 외부 배터리 전원을 사용할 것인지도 고를 수 있다. IOIO 보드의 아날로그 인풋 기록 기능을 사용하여 향후에 외부센서를 부착하여 사용하기 위하여, 5개의 핀을 따로 부착할 수 있도록 구성하였다.

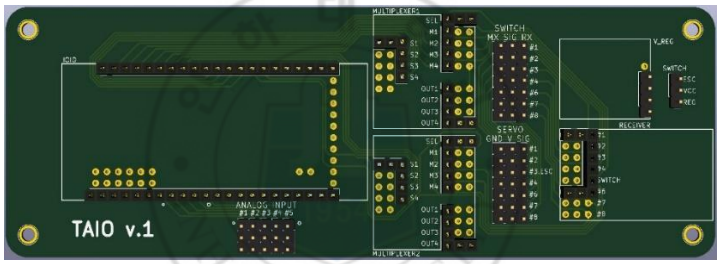

그림 9. TAIO 보드

2.3.6. 경로점 자동비행

TAIO 보드를 테스트하기 위하여, RC 비행기에 TAIO 보드와 스마트폰을 탑재하여 경로점 자동비행테스트를 하였다. 이때 자동인 것은 에일러론이였으며, 쓰로틀/엘리베이터/러더는 수동조작을 하였다. 이때 사용된 제어기는 그림 7와 비슷한 PD 제어기를 사용했다. RC보트 때와 같이 현재 위치와 경로점 위치로 타겟 헤딩을 계산하고, 타겟 헤딩과 스마트폰의 지자계 센서값과 비교하여 명령을 생성했다. 자동모드에서 에일러론의 최대명령은 $\pm 25°$로 설정하였다. 약 $160m^2$

운동장에서 비행시험을 진행하였으며, 경로점 만족조건은 30m로 설정하였다. 스마트폰에 구글지도를 사용하여 삼각형 모양 경로점 3개를 입력 후 RC 비행기에 탑재하였다. 먼저 수동비행으로 충분한 고도를 확보한 후에 경로점 자동비행을 활성화하였다. 그림 10은 경로점 자동비행을 했을 때 기록한 GPS 로깅데이터로 그린 그림이다. 실제로는 10바퀴 정도 선회했지만, 그림이 너무 복잡하게 보여 처음 자동모드를 켠 시점부터 두 바퀴 선회한 부분만 잘라서 그림을 그렸다. RC 비행기는 약 10m/s로 비행하였고, 지정된 2시, 6시, 10시 방향 경로점을 따라 성공적으로 선회하는 것을 확인할 수 있다. 이 그림에서 GPS가 1Hz이기 때문에 GPS신호 사이 간 거리가 큰 것을 확인할 수 있다.

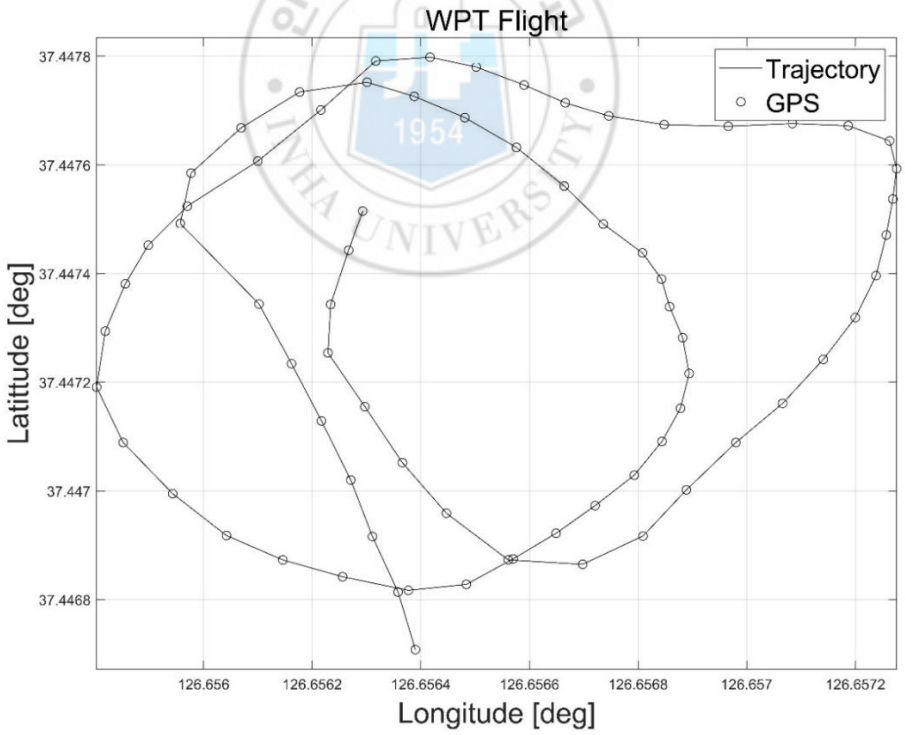

그림 10. 경로점 비행 로깅 데이터

3. GPS/INS 칼만필터

3.1. 간접 되먹임 칼만필터[8] [9]

스마트폰의 GPS 갱신주기가 1Hz이기 때문에, 고속으로 움직이는 비행체에 탑재하기에 어려움이 있다. 이를 해결하기 위하여 관성센서로 1Hz 주기 사이에 대하여 위치를 예측하는 방법이 사용된다. 관성센서는 가속도계와 자이로 센서를 의미한다. 이 관성센서는 갱신주기가 매우 빠르지만 바이어스 오차와 초기 정렬 오차가 있다. 오차값이 있는 데이터를 적분해서 항법정보를 얻을 때, 시간이 지남에 따라 오차가 기하급수적으로 증가하게 된다. 이를 해결하기 위하여 칼만필터를 사용한다.

칼만필터는 시스템 모델과 측정값을 이용하여 시스템의 상태 변수를 찾는 기법이다. 칼만필터는 여러가지 종류가 있는데, 본 논문에서 사용된 칼만필터는 간접 되먹임 (Indirect Feedback) 칼만필터이다. 간접 되먹임 칼만필터는 칼만필터를 통하여 얻어지는 추정치와 측정치의 오차를 이용하여 변수 추정치를 보정하는 방식이다. 칼만필터로 관성센서를 사용하여 계산한 위치, 속도, 자세, 가속도 센서 바이어스, 자이로 센서 바이어스를 보정한다.

본 논문에서는 가속도 센서의 고주파 노이즈를 줄이기 위하여 Low Pass 필터를 사용하였다. 또한 안드로이드 스튜디오에서 기본적으로 행렬 계산이 불가하기 때문에 Java Matrix Package, JAMA 라이브러리를 사용하여 행렬 계산을 하였다[10].

3.2. 간접 되먹임 칼만필터의 구성

3.2.1. 가속도계와 자이로센서의 초기보정

가속도계와 자이로센서의 초기보정을 위해 10초동안 가속도계의 데이터와 자이로센서의 데이터에서 출력되는 값을 기록하고 평균값을 측정한다. 이때의 가속도계의 평균값은 f_{avg}, 자이로센서의 평균값은 $(p,q,r)_{avg}$로 나타낸다.

가속도 바이어스 f^b, 자이로 바이어스 ω^b, 가속도 스케일 s_a은 다음과 같이 계산한다.

$$f^b = \begin{bmatrix} f_{x_{avg}} - sin(\theta_{avg}) \times 9.81 \\ f_{y_{avg}} - cos(\theta_{avg}) \times sin(\phi_{avg}) \times 9.81 \\ f_{z_{avg}} + cos(\theta_{avg}) \times cos(\phi_{avg}) \times 9.81 \end{bmatrix}$$

$$\omega^b = \begin{bmatrix} p_{avg} \\ q_{avg} \\ r_{avg} \end{bmatrix}$$

$$s_a = \frac{9.81}{\sqrt{f_{x_{avg}}^2 + f_{y}^2{}_{avg} + f_{z}^2{}_{avg}}}$$

위에서 계산된 바이어스과 스케일은 다음과 같이 가속도계와 자이로센서의 보정에 사용된다.

$$f = (f_x, f_y, f_z), \omega = (p,q,r)$$

$$f_b = s_a(f - f^b), \omega_b = \omega - \omega^b$$

f는 측정된 가속도계 데이터, f_b는 바이어스로 보정된 가속도계 데이터이다. 마찬가지로 ω가 자이로 센서로 측정된 데이터, ω_b가 바이어스로 보정된 값이다.

초기 방향 코사인 행렬을 만들기 위하여 롤, 피치, 요의 평균값을 측정한다. 여기서 롤, 피치, 요는 가속도센서와 지자계센서를 사용한 안드로이드 SensorManager의 내장함수인 getRotationMatrix과 orientation 함수를 사용해서 계산한다.

$$C_n^b = \begin{bmatrix} cos(\theta)sin(\psi) & cos(\theta)sin(\psi) & -sin(\theta) \\ sin(\phi)sin(\theta)cos(\psi) - cos(\phi)sin(\psi) & sin(\phi)sin(\theta)sin(\psi) + cos(\phi)cos(\psi) & sin(\phi)cos(\theta) \\ cos(\phi)sin(\theta)cos(\psi) + sin(\phi)sin(\psi) & cos(\phi)sin(\theta)sin(\psi) - sin(\phi)cos(\psi) & cos(\phi)cos(\theta) \end{bmatrix}$$

$$C_b^n = C_n^{b\prime}$$

3.2.2. INS 방정식

INS 방정식은 가속도계와 자이로 센서를 이용한 위치 및 자세의 미분 방정식이다. 입력은 가속도와 각속도이고 출력은 위치와 방향 별 속도이다. 가속도와 각속도로 미분 방정식을 구성하여 위도/경도/고도와 속도 방향코사인에 대한 미분 값을 계산하고, 이를 적분을 하여 현재 위치와 속도를 계산한다.

<지구 모델링>

$e = 0.0818191908426$

$R_0 = 6378137$

$g^n = [0\ 0\ 9.8]^T$

$\Omega = 15 \times \frac{\pi}{180} \times \frac{1}{3600}$ (지구 자전속도)

$R_m = \frac{R_0(1-e^2)}{(1-e^2 \sin^2 L)^{\frac{3}{2}}},\ R_t = \frac{R_0}{(1-e^2 \sin^2 L)^{\frac{1}{2}}}$

$\omega_{ie}^n = [\Omega\cos(L)\ \ 0\ \ -\Omega\sin(L)]^T = [\Omega_N\ \ 0\ \ -\Omega_D]^T$

$\omega_{en}^n = \left[\dot{l}\cos(L)\ \ -\dot{L}\ \ -\dot{l}\sin(L)\right]^T = [\rho_N\ \ \rho_E\ \ \rho_D]^T$

$\omega_{in}^n = \omega_{ie}^n + \omega_{en}^n$

$\omega_{nb}^b = \omega_{ib}^b - C_n^b \omega_{in}^n$

<위치, 속도, 방향코사인 미분방정식 계산>

$\dot{L} = \frac{V_N}{R_m+h}\ , \dot{l} = \frac{V_E}{(R_t+h)\ cosL}, \dot{h} = -V_D$

$\dot{V^n} = C_b^n\ f^b - (2\omega_{ie}^n + \omega_{en}^n) \times V^n + g^n$

$$\dot{C_b^n} = C_b^n \times \begin{bmatrix} 0 & -\omega_{nb}^b(3) & \omega_{nb}^b(2) \\ \omega_{nb}^b(3) & 0 & -\omega_{nb}^b(1) \\ -\omega_{nb}^b(2) & \omega_{nb}^b(1) & 0 \end{bmatrix}$$

<위치, 속도, 방향코사인 계산>

$$Llh^{INS} = Llh^{INS} + \dot{L}\dot{l}\dot{h} \times dt$$

$$V_{NED}^{INS} = V_{NED}^{INS} + \dot{V}^n \times dt$$

$$C_b^n = C_b^n + \dot{C}_b^n \times dt$$

3.2.3. 간접 되먹임 칼만 필터

본 논문에서 사용되는 간접 되먹임 칼만필터는 15개의 State로 구성되어 있다. 15개의 State는 위치, 속도, 자세, 가속도 센서 바이어스, 자이로 센서 바이어스로 이루어져 있으며, 이 State들의 보정 값으로 δx가 정의된다. δx는 위치 오차값 $\delta L, \delta l, \delta h$, 속도 오차값 $\delta_N, \delta_E, \delta_D$, 자세 오차값 ϕ_N, ϕ_E, ϕ_D, 가속도 센서 바이어스 $\nabla_x, \nabla_y, \nabla_z$, 자이로 센서 바이어스 $\epsilon_x, \epsilon_y, \epsilon_z$으로 이루어져 있다. 이 보정 값 δx로 다음 단계에서 상태 벡터 x를 보정한다.

아래의 그림 11은 자세하게 간접 되먹임 칼만필터의 구성을 나타낸 그림이다. 자세한 수식을 설명하기 전에 계산 순서에 대해서 설명하자면, 그림 11과 같이 간접 되먹임 칼만필터는 먼저 오차 공분산 초기화의 초기값을 설정하고 오차 공분산 예측을 계산한다. 이때 오차 공분산 예측 계산에 시스템 모델 F가 들어가는데, F에는 IMU에서 측정된 값이 들어간다. 오차 공분산 예측으로 칼만 이득을 계산하게 되고, Correction Calculation을 하게 된다. 여기서 Correction Calculation이란 앞에서 언급한 δx의 계산을 의미한다. 계산된 δx로 INS Equation으로 계산된 위치/속도/방향을 보정하게 되고, 센서들도 보정하게 된다. 그 이후

오차 공분산 보정을 하게 되고 한 Loop가 끝나게 되고 반복된다.

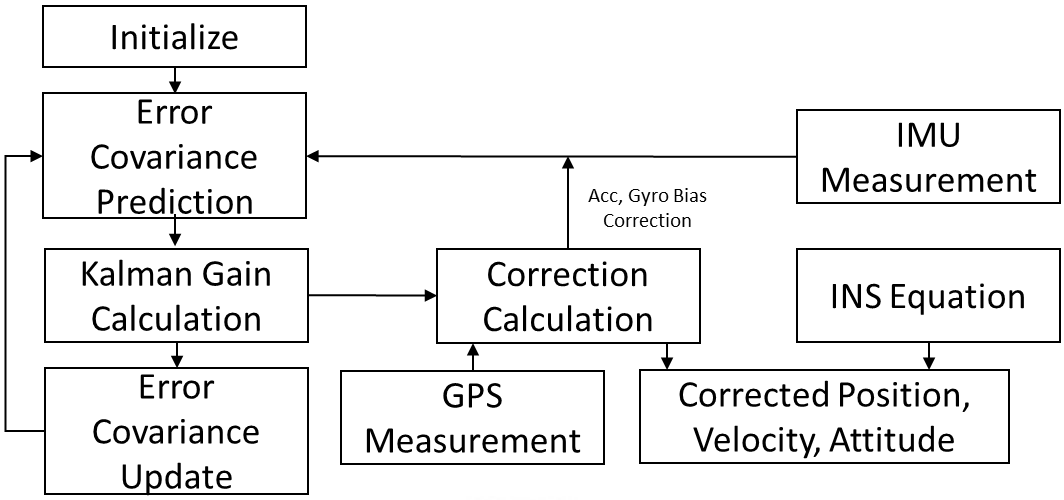

그림 11. 간접 되먹임 칼만필터의 구성

그림 11의 구성도에서 각각의 요소들의 계산을 순서대로 나열해보면 다음과 같다. 먼저 초기 오차 공분산 (Error Covariance) P_k는 다음과 같이 설정한다.

$$P_k = diag(1 \times 10^{-14}, 1 \times 10^{-15}, 1, 4.21 \times 10^{-5}, 2.73 \times 10^{-5}, 0.01, 0.01, 0.01, 0.28, 0.75, 0.8, 0.4, 1 \times 10^{-4}, 1 \times 10^{-4}, 3 \times 10^{-4})$$

다음 순서는 Error Covariance Prediction으로 $P_k^- = F_k P_k F_k^T + G_k Q G_k^T$로 계산된다. 이 식에서 칼만필터의 시스템 모델 F은 다음과 같이 구성된다. 시스템 모델 F를 구성하는 데에 IMU 측정값들이 포함되게 된다.

$$F = \begin{bmatrix} F_{11} & F_{12} & 0_{3x3} & 0_{3x3} & 0_{3x3} \\ F_{21} & F_{22} & F_{23} & F_{24} & 0_{3x3} \\ F_{31} & F_{32} & F_{33} & 0_{3x3} & F_{35} \\ 0_{3x3} & 0_{3x3} & 0_{3x3} & 0_{3x3} & 0_{3x3} \\ 0_{3x3} & 0_{3x3} & 0_{3x3} & 0_{3x3} & 0_{3x3} \end{bmatrix}$$

$R_{mm} = 6R_0 esin(L)\cos(L)$, $R_{tt} = 2R_0 esin(L)\cos(L)$

$$F_{11} = \begin{pmatrix} \frac{\rho_E \times R_{mm}}{R_m+h} & 0 & \frac{\rho_E}{R_m+h} \\ \rho_N secL(tanL - \frac{R_{tt}}{R_t+h} & 0 & -\frac{\rho_N secL}{R_t+h} \\ 0 & 0 & 0 \end{pmatrix}$$

$$F_{12} = \begin{pmatrix} \frac{1}{R_m+h} & 0 & 0 \\ 0 & \frac{secL}{R_t+h} & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

$$F_{21} = \begin{pmatrix} -(2\Omega_N + \rho_N \sec^2 L + \frac{\rho_D R_u}{R_t+h})V_E + \frac{\rho_E R_{mm}}{R_m+h} V_D & 0 & -\frac{\rho_D}{R_t+h}V_E + \frac{\rho_E}{R_m+h}V_D \\ (2\Omega_N + \rho_N \sec^2 L + \frac{\rho_D R_{tt}}{R_t+h})V_N + \left(2\Omega_D - \frac{\rho_N R_{tt}}{R_t+h}\right)V_D & 0 & \frac{\rho_D}{R_t+h}V_N - \frac{\rho_N}{R_t+h}V_D \\ -\frac{\rho_E R_{mm}}{R_m+h}V_N - \left(2\Omega_D - \frac{\rho_N R_{tt}}{R_t+h}\right)V_E & 0 & -\frac{\rho_E}{R_m+h}V_N + \frac{\rho_N}{R_t+h}V_E \end{pmatrix}$$

$$F_{22} = \begin{pmatrix} \frac{V_D}{R_m+h} & 2\Omega + 2\rho_D & -\rho_E \\ -2\Omega_D - \rho_D & \frac{V_N tanL + V_D}{R_t+h} & 2\Omega_N + \rho_N \\ 2\rho_E & -2\Omega_N - 2\rho_N & 0 \end{pmatrix}$$

$$F_{23} = \begin{pmatrix} 0 & -f_D & f_E \\ f_D & 0 & -f_N \\ -f_E & f_N & 0 \end{pmatrix}$$

$F_{24} = C_b^n$

$$F_{31} = \begin{pmatrix} \Omega_D - \frac{\rho_N R_{tt}}{R_t+h} & 0 & -\frac{\rho_N}{R_t+h} \\ -\frac{\rho_E R_{mm}}{R_m+h} & 0 & -\frac{\rho_E}{R_m+h} \\ -\Omega_N - \rho_N \sec^2 L - \frac{\rho_D R_{tt}}{R_t+h} & 0 & -\frac{\rho_D}{R_t+h} \end{pmatrix}$$

$$F_{32} = \begin{pmatrix} 0 & \frac{1}{R_t+h} & 0 \\ -\frac{1}{R_m+h} & 0 & 0 \\ 0 & -\frac{tanL}{R_t+h} & 0 \end{pmatrix}$$

$$F_{33} = \begin{pmatrix} 0 & \Omega_D + \rho_D & -\rho_E \\ -\Omega_D - \rho_D & 0 & \Omega_N + \rho_N \\ \rho_E & -\Omega_N - \rho_N & 0 \end{pmatrix}$$

$$F_{35} = -C_b^n$$

위의 식에서 P_k는 처음 칼만필터를 실행했을 때는 초기 P_k 값을 따르고, 다음 루프에서는 추후에 계산된 P_k값을 사용한다. 이어서 Error Covariance Prediction, P_k^-를 구성하는 G_k와 Q는 다음과 같다. 여기서 T는 시스템의 시간단계인 0.1초이다.

$$G_k = \begin{pmatrix} 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & C_b^n & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & -C_b^n & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \end{pmatrix} \times T$$

$$Q = diag(0,0,0,0.05^2,0.05^2,0.01^2,0.01^2,0.01^2,0,0,0,0,0,0)$$

다음으로는 칼만이득 (Kalman Gain)을 계산한다. 식은 다음과 같다.

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$$

칼만이득에 P_k^-는 앞서 계산한 Error Covariance Prediction를 사용하고, 나머지 H 행렬과 R 행렬은 아래와 같이 설정한다.

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$R = diag(1.1981 \times 10^{-14}, 1.0595 \times 10^{-15}, 1.1548, 4.2144 \times 10^{-5}, 2.733 \times 10^{-5})$$

그 다음 순서로 Correction Calculation δx를 계산한다.

$$\delta\widehat{x_k} = K_k z_k$$

위의 식에서 z_k는 칼만필터의 측정식이라고 하며, 다음 식과 같다.

$$z_k = \begin{pmatrix} Llh^{INS} - Llh^{GPS} \\ V_{NED}^{INS} - V_{NED}^{GPS} \end{pmatrix}$$

위의 측정값 계산식에서 Llh^{GPS}는 GPS에서 위도, 경도, 고도 정보가 1Hz로 갱신될 때 같이 갱신된다. 안드로이드 Location Manager를 사용하여 GPS에서 얻을 수 있는 것은 속력이기 때문에, V_{NED}^{GPS}는 다음과 같이 계산을 하여 대입한다. Time Elapsed는 GPS신호의 시간간격을 의미하며, GPS신호가 잡혔을 때의 시간을

기록하여 비교함으로써 계산된다. 이것으로 GPS 신호가 안 잡혀 갱신주기보다 더 늦게까지 신호가 안 들어올 것을 대비하였다.

$$R_m = \frac{R_0(1-e^2)}{(1-e^2 \sin^2 L)^{\frac{3}{2}}}$$

$$R_t = \frac{R_0}{(1-e^2 \sin^2 L)^{\frac{1}{2}}}$$

$$V_N^{GPS} = (R_m + h) \times \frac{L - L_{pre}}{Time\ Elapsed}$$

$$V_E^{GPS} = (R_t + h) \times \cos(L) \times \frac{l - l_{pre}}{Time\ Elapsed}$$

$$V_D^{GPS} = \frac{alt_{pre} - alt}{Time\ Elapsed}$$

3.2.4. 칼만필터 보정

그림 11에서 Calculate Correction에 해당되는 Update Estimate에서 계산되는 오차 추정값 δ_x는 다음과 같다.

$$\delta_x = [\delta L\ \delta l\ \delta h\ \delta V_N\ \delta V_E\ \delta V_D\ \phi_N\ \phi_E\ \phi_D\ \nabla_X\ \nabla_Y\ \nabla_Z\ \epsilon_X\ \epsilon_Y\ \epsilon_Z]^T$$

이 값들 중에 1~6번째 State에 해당하는 위도, 경도, 고도, N방향 속도, E방향 속도, D방향 속도는 INS 방정식 결과값에 더해져 다음과 같이 위치와 속도를 보정한다.

$$x = x_{INS} - \delta x$$

7~16번째에 해당되는 값들은 자세 업데이트, 가속도 센서 보정, 자이로 센서 보정에 사용된다. 다음 식을 통해 시스템에 반영된다.

$$C_b^n = \left(I - skew(\phi_N, \phi_E, \phi_D)\right)^{-1} C_b^n$$
$$f_b = s_a\left(f - f^b\right) - (\nabla_X \nabla_Y \nabla_Z)^T$$
$$\omega_b = \left(\omega - \omega^b\right) - (\epsilon_X \epsilon_Y \epsilon_Z)^T$$

3.3. 간접 되먹임 칼만필터 검증

간접 되먹임 칼만필터의 검증은 먼저 MATLAB 환경에서 진행되었다. 안드로이드 스마트폰을 RC자동차에 탑재하여 기록한 주행 로깅 데이터를 활용, 칼만필터가 성공적으로 작동하는지 검증 후 안드로이드 스튜디오 환경에 그대로 적용하였다. GPS/INS의 주기는 100Hz로 설정했는데, 그 이유는 필터의 한 루프를 실행하는 데에 2~3ms 밖에 걸리지 않았기 때문이다.

아래 그림 12은 송도 항공우주캠퍼스 옥상에서 안드로이드 스마트폰에 간접 되먹임 칼만필터를 적용하고, 그림 13의 RC 자동차에 탑재하여 진행한 결과이다. 조종은 수동으로 진행하였으며, 스마트폰 어플리케이션에서 로깅 기능을 활성화시켜 GPS 위치 데이터와 GPS/INS 간접 되먹임 칼만필터의 결과를 기록하였다.

그림 12에서 원으로 표시된 것이 GPS로부터 측정된 위치이고, 빨간색 점으로 보이는 것이 GPS/INS 간접 되먹임 칼만필터로 위치 예측을 한 것이다. GPS 위치와 GPS/INS필터의 값을 비교하기 위하여 축의 단위는 미터로, 초기 위치를 기준으로 정각원추도법, Lambert Conformal Conic Project을 사용하여 계산하였다. 그림을 확인해보면 GPS 위치와 GPS/INS로 계산/예측한 위치가 살짝 다른 것을 확인할 수 있는데, 이는 몇 센티미터 차이로 오차가 매우 적은 것을 확인할 수 있다.

그리고 칼만필터가 제대로 작동하는지는 공분산을 확인해보면 되는데, 공분산 데이터를 확인해본 결과, 15개의 State들의 공분산이 칼만필터 작동 직후 급격하게 떨어지고 수렴을 하는 것을 확인할 수 있었다.

최종적으로 이로 원래 목적이었던 1Hz의 GPS주기 사이를 GPS/INS가 성공적으로 잘 채워 주는 것을 확인할 수 있었고, 이를 사용한다면 고속으로 비행하는 기체의 비행테스트까지 원활하게 가능하다고 판단된다.

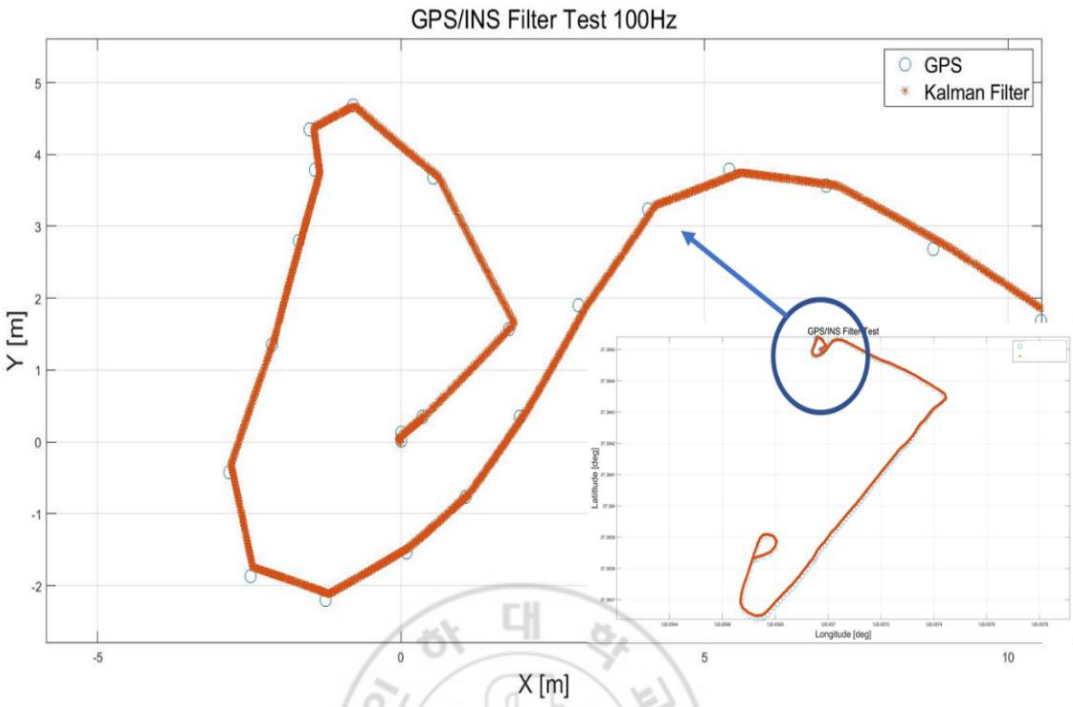

그림 12. 간접 되먹임 칼만필터 검증

그림 13. 검증에 사용한 RC 자동차

4. 가변적 시간단계 제어시스템

4.1. 사용 목적과 개념

보통 제어시스템은 시간단계(Time Step)가 일정하다고 가정하고 명령을 계산한다. 하지만 성능이 좋지 않은 스마트폰을 스마트폰 기반 제어시스템으로 사용했을 때, 시간단계를 지키지 못하는 증상을 발견했다. 이는 어플리케이션에 기능이 많아졌기 때문이다. 또한 추후에 영상과 통신 등 더 많은 기능이 추가될 예정이기 때문에 늘어난 계산 량을 최신 스마트폰도 감당해내지 못해 시간단계를 일정하게 유지하지 못하고, 제어 루프의 실시간성을 보장하지 못할 수도 있다. 따라서 정확한 명령생성을 위하여 스마트폰 기반 제어시스템에 가변적 시간단계 적용이 필요하다고 생각되었다.

가변적 시간단계 제어시스템은 매 스텝마다 시간단계를 계산하고 가변적인 시간단계로 제어명령을 계산한다.

4.2. 검증

가변적 시간단계 제어시스템의 검증은 관제 시뮬레이션을 위해 개발된 5 자유도 모델 시뮬레이션 시스템을 사용하였다[11]. 시간단계가 일정하지 않고, 딜레이가 존재한다고 가정을 하였다. 기존의 5 자유도 모델 시뮬레이션의 시간단계는 0.1초인데, 이 시간단계를 매번 랜덤으로 0.1~0.15초가 되도록 설정함으로써 시간단계가 밀리는 현상을 모사하였다. 시뮬레이션 상이 아닌, 실제로는 시간단

계가 바뀐다는 것은 센서 데이터가 빠른 주기로 갱신되고 있을 때, 원래 센서 데이터를 불러오는 시점보다 더 늦게 센서 데이터를 받아서 명령을 생성하는 것을 의미한다. 시뮬레이션 상의 Dynamics를 계산할 때는 이를 모사하기 위하여 시간 적분구간을 가변적으로 하여 계산하였다.

본 논문에 사용된 5 자유도 모델 시뮬레이션에는 고도 제어에 상승/하강률 제어를 사용한다. 다음은 상승/하강률 PID 제어어기에 가변적 시간단계 제어시스템이 적용된 예이다.

$$H_{CMD} = Err_{ROCD} \times P_{gain} + Err_{ROCD_I} + I_{gain} + Err_{ROCD_D} \times D_{gain}$$

$$Err_{ROCD_I} = Err_{ROCD_I} + Err_{ROCD} \times dt_{variable}$$

$$Err_{ROCD_D} = \frac{(ROCD - ROCD_{pre})}{dt_{variable}}$$

위의 식에서 Err_{ROCD}는 목표 상승/하강률에 현재 상승/하강률을 뺀 오차를 나낸다. Err_{ROCD_I}는 Err_{ROCD}에 시간단계인 dt를 곱하여 오일러 적분으로 상승/하강률 오차에 대한 적분값을 만들어낸 것이다. Err_{ROCD_D}는 Err_{ROCD}의 미분값으로, 현재 상승/하강률과 바로 전 스텝의 상승/하강률을 비교하고, dt로 나눈 것을 의미한다. 위에 식에서는 dt가 들어간 항이 적분기 I항과, 미분기 D항이다. 이곳에 적용되는 dt를 가변적으로 적용하였다. 마찬가지로 헤딩 제어와 속도 제어의 dt

들도 같은 가변적 시간단계를 적용한다.

가변적 시간단계 제어시스템을 검증하기 위하여 세가지를 비교해보았다. 그 세가지는 표2를 보면 확인할 수 있다. 첫번째는 Reference (기준)으로 시간단계가 일정하고, 제어기 시간단계도 일정하게 설정된 것이다. 두번째는 Delayed로 실제로는 시간단계에 랜덤하게 지연이 있지만, PID 제어기에 dt로 들어가는 시간단계는 일정하게 설정한 것이다. 마지막으로 Variable dt라고 명명한 것은 시간단계에 랜덤 지연이 생기고, 가변적으로 제어기가 실행될 때 이 지연된 시간단계를 계산하여 PID제어기에 사용한다.

표 2. 가변적 시간단계 제어시스템 검증

	Reference	Delayed	Variable dt
시간단계	일정	랜덤 지연	랜덤 지연
제어기 시간단계 (dt)	일정	일정	랜덤 지연

시뮬레이션을 실행하기 위하여 선택된 시나리오는 RNAV OLMEN 1N STAR 절차이고, 항공기 모델은 B737-800으로 진행하였다.

그림 14은 앞에서 말한 3가지의 경우를 모두 실행해보고, 기준과 비교하여 오차가 제일 크게 나타나는 구간을 위도, 경도, 고도별로 각각 나타낸 것이다. 그림 14에서 혼자 떨어져 있는 데이터 포인터가 Delayed 값이며, 아주 가까이 모여 있는 두개의 데이터 포인터 중에 왼쪽편에 있는 데이터가 Variable dt, 오른쪽

편에 있는 데이터가 Reference이다. 그림을 보면 Reference 값과 Variable dt 값은 오차가 거의 없다시피 비슷한 결과가 나왔고, 지연이 있으나 고정된 dt를 PID에 사용한 Delayed의 값은 큰 오차를 보였다. 표3은 그림 14의 결과를 표로 나타낸 것이다.

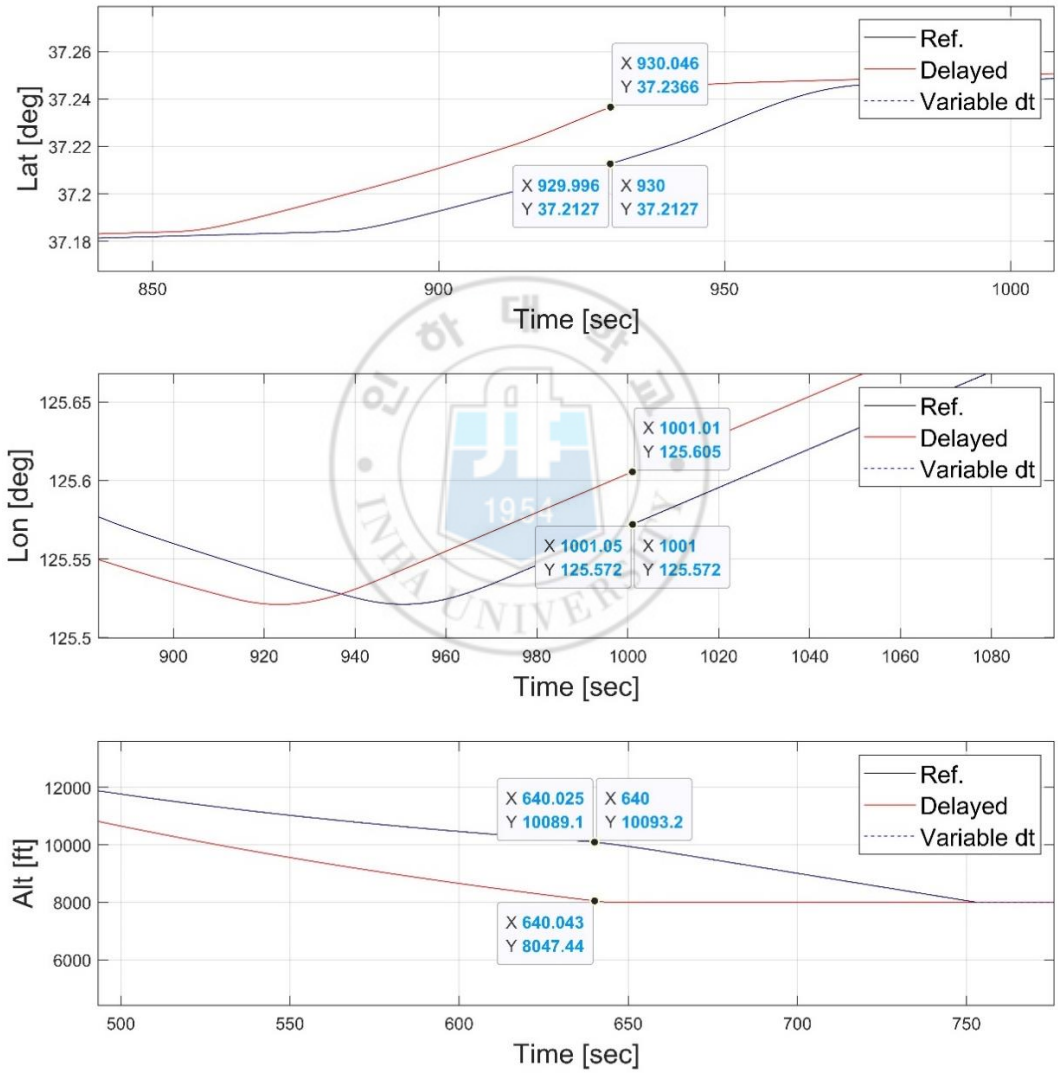

그림 14. 가변적 시간단계 제어기 검증 결과

표 3. 가변적 시간단계 제어시스템 검증 결과

	위도 [deg]	경도 [deg]	고도 [ft]
Reference	37.2126793	125.5720824	10093.16478
Delayed	-0.023892303	-0.033398652	+2045.726715
Variable dt	-1.1678E-05	+3.71288E-05	+4.100806025

그림 14과 표 3을 보면, 항공기 상태의 변화가 크고 시간단계에 지연이 생겼음에도 불구하고, 제어기에 사용하는 dt가 일정하다고 가정하게 되면 고도의 경우 크게는 2045ft까지 오차가 크게 생기는 것을 확인할 수 있다. 하지만 매 간격마다 시간간격을 계산하고 가변적 시간단계를 PID 제어기에 적용했을 때는 Reference 값과 매우 비슷한 결과를 도출할 수 있었다.

따라서 스마트폰 기반 제어시스템에 가변적 시간단계 제어시스템을 적용하였을 때, 시간단계에 지연이 생기는 상황이라면 더 정확한 자동비행을 위하여 유용하게 사용될 수 있다고 판단된다.

5. 결론

본 연구에서는 스마트폰 기반 제어컴퓨터를 위한 가변적 시간제어 시스템을 제시했다. 그 이전에 복잡한 기존의 제어시스템 대신, 스마트폰 기반 비행제어 시스템에 대한 연구를 진행했다.

제어시스템에서 시간단계에 지연이 생겨 제어 루프의 실시간성을 보장하지 않는 상황을 대비하여 가변적 시간제어 시스템을 적용해보고, 검증으로는 5 자유도 모델 시뮬레이션이 사용되었다. 그 결과, 이동체의 상태 변화가 클 때 시간단계 지연에 대한 영향이 컸고, 시간단계를 가변적으로 계산하여 PID 제어기에 사용한 경우 결과값이 기준값과 매우 비슷한 것을 확인할 수 있었다.

무인선과 무인차량을 사용하여 단계적 개발을 거쳐 스마트폰 기반의 비행제어시스템을 개발하였다. 개발 도중에 여러 문제점들을 발견하였고, 그것을 해결하는 방식으로 연구가 진행되었다. 전기신호 계열 이상은 TAIO 통합보드 제작, 느린 GPS 주기는 간접 되먹임 칼만필터 GPS/INS의 적용으로 해결하였다. 그리고 부분적인 자동 비행을 수행하였다.

본 논문에서 제시한 가변적 시간제어 시스템과 스마트폰 기반 제어시스템을 활용해 추후에는 [12]논문에서 제시한 방법으로 무인항공기의 시스템 식별을 해보고 PILS 시뮬레이션으로 제어기 튜닝을 진행하고 완전 자동비행테스트까지 진행할 예정이다.

6. 참고문헌

[1] “Mobile Processors 101: Why Smartphones Are Smarter with an All-in-One Processor.″ Qualcomm, 9 May 2019, www.qualcomm.com/news/onq/2013/06/13/mobile-processors-101-why-smartphones-are-smarter-all-one-processor.

[2] ″Qualcomm SM8250 Snapdragon 865 vs Intel Core i5-7500.″ GadgetVersus, gadgetversus.com/processor/qualcomm-sm8250-snapdragon-865-vs-intel-core-i5-7500/.

[3] ″센서 개요.” Android Developers, developer.android.com/guide/topics/sensors/sensors_overview?hl=ko.

[4] 박승현, 박정환, 정호준, 이학태. (2019). 스마트폰 기반 비행제어 시스템 초기 개발. 한국항공우주학회 학술발표회 초록집, (), 275-276.

[5] ″Sensor Event.″ Android Developers, developer.android.com/reference/android/hardware/SensorEvent.

[6] ″Aircraft Rotations.″ NASA, NASA, www.grc.nasa.gov/WWW/K-12/airplane/rotations.html.

[7] 박승현, 박정환, 김태영, 이학태. (2019) 스마트폰 기반 비행제어 시스템을 위한 롤각 제어기 개발. 한국항행학회 종합할술 대회 논문집.

[8] 김민지, 주문갑. (2015). INS/GPS와 간접 되먹임 칼만 필터를 사용하는 이동 로봇의 복합 항법 시스템의 구현. 대한임베디드공학회논문지 10(6), 373-379.

[9] 이종무, 이판묵, 성우제. (2003). 간접 되먹임 필터를 이용한 관성센서 및 초음파 속도센서 기반의 수중 복합항법 알고리듬. 한국해양공학회지, 17(6), 83-90.

[10] ″JAMA: A Java Matrix Package.″ NIST, math.nist.gov/javanumerics/jama/.

[11] 강지수, 오혜주, 최기영, 이학태. (2016). 항공교통관제 시뮬레이션을 위한 개선된 5 자유도 항공기 운동 모델 개발 및 검증방안 연구. 한국항행학회논문지, 20(5), 387-393.

[12] 이윤생, 석진영, 김태식. (2002). 무인항공기의 시스템 식별을 위한 비행시험기법. 한국항공우주학회지, 30(7), 130-136.