배재고등학교

2025학년도

1학년 자율탐구프로젝트 결과물

전기 덕트팬을 기반으로 한 VTVL의 개발과 GNC 알고리즘에 대한 탐구

지도교사 정 예 은

직상직하 팀

11309 박 원 비

10914 백 승 원

11006 김 민 호

초 록

본 연구는 전기 덕트팬(Electric Ducted Fan, EDF)을 기반으로 한 소형 VTVL(Vertical Take-off and Vertical Landing) 실증기를 제작하고, 해당 기체에 적용 가능한 저비용, 고확장성 GNC(Guidance, Navigation and Control) 알고리즘을 설계, 검증하는 것을 목표로 한다. 기존 재사용 발사체의 재착륙 제어 기술은 고가의 상용 비행제어기와 비공개 알고리즘에 크게 의존하고 있어, 학생 및 실험실 규모에서의 접근이 제한적이었다. 이에 본 연구에서는 상용 FC를 사용하지 않고, Teensy 4.1 기반의 자체 비행 컴퓨터와 IMU, GPS, ToF 센서를 활용한 상태 추정 시스템을 직접 구현하였다.

기체는 원통형 동체와 TVC(Thrust Vector Control)를 적용한 로켓형 구조로 설계되었으며, 메인 추진기로 90mm급 EDF를 사용하고 롤 축 제어를 위해 측면 보조 EDF를 추가하였다. 자세 표현에는 쿼터니언 체계를 적용하여 수직 착륙 과정에서 발생할 수 있는 짐벌락 문제를 회피하였고, IMU 기반 고속 자세 추정과 GNSS·고도계 기반 저속 위치 보정을 결합한 계층적 항법 구조를 구성하였다. 또한 TVC 짐벌은 4절 링크 기구를 이용한 회전형 서보 구동 방식으로 설계하고, 기구학 해석 및 구조 해석을 통해 타당성을 검증하였다.

이론적 동역학 모델링과 비행 시험을 통해 설계된 제어 구조가 저속, 저고도 VTVL 환경에서 안정적인 자세 유지와 제어가 가능함을 확인하였다. 본 연구는 학생 수준에서도 구현 가능한 VTVL 재착륙 실험 플랫폼과 GNC 설계 절차를 제시함으로써, 향후 실험실 규모 재사용 발사체 연구의 기초 자료로 활용될 수 있을 것으로 기대된다.

목 차

표 목 차

그 림 목 차

Nomenclature

Acronyms

- GNC: Guidance, Navigation and Control (유도・항법・제어)
- VTOL / VTVL: Vertical Take-Off and Landing / Vertical Take-Off Vertical Landing (수직이착륙 / 수직이착륙・수직착륙)
- AHRS: Attitude and Heading Reference System (자세 및 방위기준시스템)
- IMU: Inertial Measurement Unit (관성측정장치, 가속도계 + 자이로)
- GNSS: Global Navigation Satellite System (위성항법, GPS 등)
- TVC: Thrust Vector Control (추력 편향 제어)
- EDF: Electric Ducted Fan (전기 덕트팬)
- ESC: Electronic Speed Controller (모터 전자변속기)
- PWM: Pulse Width Modulation (펄스폭 변조)
- PID: Proportional-Integral-Derivative controller (비례-적분-미분 제어기)
- ENU: East-North-Up coordinate frame (동-북-상 좌표계)
- LPF: Low-Pass Filter (저역통과 필터)

Coordinate Frames and Indices

- {E}: 지구 고정 좌표계(ENU)
- {B}: 기체 좌표계(Body frame)
- 아래첨자 E: ENU 좌표계에서 표현된 벡터
- 아래첨자 B: 기체 좌표계에서 표현된 벡터
- (x_E, y_E, z_E): ENU에서의 위치 성분 $[m]$
- $(u_E, v_E, w_E,)$: ENU에서의 속도 성분 $[m/s]$
- $(p,\ q,\ r)$: 기체 좌표계 각속도(roll/pitch/yaw rate) [rad/s]

State, Attitude Representation

- x: 상태벡터(state vector)
- $p_E = [x_E, y_E, z_E]^T$: 위치벡터 [m]
- $v_E = [u_E, v_E, w_E]^T$: 속도벡터 [m/s]
- $\omega_E = [p, q, r]^T$: 기체 가속도 [rad/s]
- $q = [q_0, q_1, q_2, q_3]^T$: 자세 쿼터니언(스칼라-벡터 순서) [−]
- (ϕ, θ, ψ) : 오일러각(roll, pitch, yaw) [rad]
- ($\hat{-}$) : 추정치(estimated value)
- ($-_d$) : 목표치(desired/reference)
- ($e_{(-)}$) : 오차(error)

Ⅰ. 서론

1. 연구의 동기 및 목적

최근 몇 년간 스페이스X(SpaceX)를 비롯한 다양한 민간 우주 기업들이 재사용 가능한 발사체 기술을 상용화함에 따라, 기존의 일회용 발사체 중심에서 재사용 발사체로의 기술적 전환이 급속히 진행되고 있다. 특히, 재사용 발사체의 핵심 기술 중 하나인 정밀 재착륙 기술은 발사체의 운영 비용을 획기적으로 줄이고 발사 주기를 단축할 수 있는 잠재력을 지니고 있어, 학계와 산업계 모두에서 큰 관심을 받고 있다. 이러한 재착륙 기술을 구현하기 위해서는 고성능의 GNC(유도, 항법, 제어, Guidance, Navigation and Control) 알고리즘이 요구되며, 실제로 많은 기업들이 관련 기술을 자체 개발하여 적용하고 있다.
그러나 현재 사용되는 GNC 기술은 대부분 군사 혹은 기업 단위에서 천문학적인 비용을 들여 개발하였기에 관련 자료나 알고리즘이 외부에 공개되어 있지 않거나, 고가의 상용 비행 제어기(Commercial off-the-shelf, COTS Flight Controller)에 의존하는 경우가 많다. 이로 인해 일반 학생이나 실험실 규모에서는 GNC 기술을 기반으로 한 재착륙 제어 실험을 직접 수행하거나 재현하기 매우 어렵다.
이에 본 연구는 복잡한 상용 장비나 고가의 센서 없이, 학생 수준에서도 구현 가능한 재착륙용 GNC 시스템을 제안하고자 한다. 구체적으로는, 전통적인 멀티콥터 기반의 드론이 아닌, 원통형 직립 이착륙 형태(CVTVL, Cylindrical Vertical Take-off and Vertical Landing)의 기체를 제작하고, 그 추진 시스템에 TVC(Thrust Vector Control, 추력 편향 제어)를 적용할 수 있는 새로운 제어 체계를 직접 설계 및 구현하는 것을 목표로 한다.

이러한 시스템은 기존의 Pixhawk나 Ardupilot과 같은 상용 FC(Flight Controller)를 사용하지 않고, 자체 설계한 시스템을 통해 상태 추정과 제어 등

재착륙에 요구되는 전 과정을 통합 개발한다는 점에서 의의가 있다. 이를 통해, 일반 학생 및 연구자들도 실험실 규모에서 재사용 발사체의 핵심 기술 중 하나인 재착륙 제어를 직접 구현·검증해볼 수 있는 개방적이고 확장성 있는 실험 플랫폼을 제시하고자 한다.

본 논문에서는 해당 기체 및 제어 시스템의 설계 과정, 동역학 모델링, 상태 추정 및 제어기 설계, 시뮬레이션 및 실험 검증 과정을 단계적으로 제시함으로써, 향후 실험실 규모 재사용 발사체 연구의 기반 자료로 활용될 수 있도록 한다.

2. 선행연구 고찰

VTVL의 GNC 알고리즘을 연구하기 위해서 VTVL의 시험 비행이나 시뮬레이션은 필수적이다. 하지만 비행 제어 알고리즘을 연구하기 위해서 실규모 발사체를 제작하는 것은 비용과 안전성 측면 모두에서 비효율적이다 [1]. 따라서 실험실 규모에서 비행 제어 알고리즘을 연구하기 위해서 소형 VTVL 실증기를 제작하는 경우가 많다. 다음은 VTVL 실증기의 사례이다.

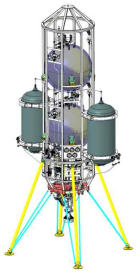
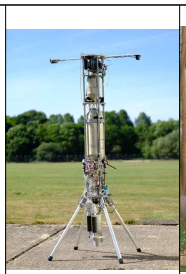
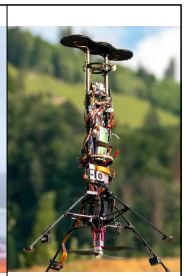
					
Armadillo Aerospace MOD -1	항공우주연구원 1톤급 수직이착륙 시연체	Airborne Engineering Gyroc	L Shang VTVL	guryere space program	guryere space program
LOX/IPA	LOX/LCH4	N2O/IPA	GOX/EtOH	N2O/IPA	Electric

Fig. 1. VTVL 실증기 사례 분석

위에서부터 순서대로 사진, 연구기관/기체이름, 추진체계이다. 이들 중 최 우측 1기를 제외하고 모두 화학 추진을 이용하는 시스템이다. 실제 우주 발사체와 동일한 시스템을 사용하기 때문에 추력 응답성, 연료 소모에 따른 질량 감소와 질량 중심점의 변화, 슬로싱등 다양한 환경을 동시에 실험할 수 있다는 장점이 있지만, VTVL에 적용하기 위해 높은 추력 조절 범위를 가지는 액체 연소기를 개발하기 위해서는 높은 비용과 위험성이 동반된다. 따라서 본 연구는 저비용과 구조적 단순함을 위해 최우측 사진과 같은 전동기를 이용하는 시스템을 구성한다. Guryere Space Program에서 Fig.xx의 5번째 기체를 시험하기전 GNC 알고리즘을 개발하기 위해 6번째 사진의 기체를 개발하였다. Contra-rotating propellers를 추진기관으로 채택하였기 때문에 롤 제어가 가능하다는 장점이 있다. 대부분의 우주발사체는 노즐 김벌링(Nozzle Gimbaling)을 통해 TVC를 구현하는 반면, 단일 추력기를 사용할 경우에 TVC를 통한 롤 제어는 불가능하다. 따라서 상용 우주 발사체는 별도의 롤 제어기로 RCS(Reaction control system)나 반작용 휠(Reaction wheel)을 체택한다. 본 연구에서 또한 RCS를 모사하기 위한 소형 팬을 측면에 부착하여 롤 모멘트를 발생시킬 수 있도록 하였다.

별도의 롤 제어기로 인한 롤 제어가 가능해졌기 때문에 프로펠러의 무게나 부피의 대비 추력 효율이 높은 덕트팬(EDF, Electric Ducted Fan)을 채택하였다. 또한 한 방향으로 지속적으로 회전 하는 덕트팬은 회전의 반대 방향으로 지속적인 롤 모멘트를 발생시킨다. 롤 제어기를 기체의 회전방향을 상쇄시키는 방향으로만 설치하여 그 추력을 증가/감소시키며 롤 모멘트을 제어 하도록 하였다.

무인 멀티콥터, 회전익을 운용하기 위한 상용 비행 제어 소프트웨어들은 과거부터 많은 기업들로부터 오픈소스로 제공되어 왔다. PX4, Ardupilot, iNAV 등이 그 대표적인 예시로써 PX4의 경우 고정익, 회전익, 멀티콥터, 잠수함,

VTOL 비행기등 다양한 운송수단에 맞춰 사용될 수 있다. 하지만 TVC 시스템을 사용하는 로켓의 경우 오픈소스로 유통되는 비행 제어 소프트웨어가 잘 알려지지 않았고, 개발 상황에 따른 요구조건이 크게 달라질 수 있다는 점, 높은 확장성을 요구한다는 점에서 직접 구현하기로 하였다.

3. 연구 방법

본 연구에서는 수직 이착륙(VTVL) 기체의 자세 안정화 및 제어 기술을 확보하기 위해 이론 연구 – 시뮬레이션 – 실 기체 제작– 비행시험으로 이어지는 단계적 접근 방식을 취하였다. 먼저, 일반적인 임베디드 환경에서도 구현이 가능하도록 보편적인 MCU 기반 비행 컴퓨터 구조를 설계하였으며, Arduino IDE 기반의 개발 환경에서도 접근이 가능하도록 초기 시스템을 구성하였다. 이후 실험용 기체를 제작하고 반복적인 비행 시험을 하면서 GNC 알고리즘을 보완하고 제어기 파라미터를 직접 튜닝하는 과정을 통해 성능을 개선하였다.

우주발사체의 비행 특성을 모사하기 위해, 기존 연구에서 흔히 사용되는 COTS 멀티콥터 기반 플랫폼 대신 원통형 동체와 추력편향(Thrust Vector Control, TVC) 기구를 갖춘 로켓형 구조를 채택하였다. 비용 및 안전성을 고려하여 메인 추진기는 90mm 전기 덕트 팬으로 구성하고, 전체 질량을 3kg 미만, 길이를 약 1200mm, 직경을 180mm로 설계하였다. 또한 롤 축 제어를 위해 동체 측면에 30mm급 소형 덕트팬 2기를 추가로 장착하여 3축 제어가 가능하도록 하였다. 기체 설계의 타당성은 MATLAB 기반 동역학 모델 시뮬레이션을 통해 먼저 검증한 후, 실제 비행 데이터를 이용해 추가적으로 보정 하였다.

기존 연구 사례로는 NASA, Armadillo Aerospace, SpaceX 등의 Lander 및 Hopper 계열 기체가 있으나, 이들은 대부분 액체 추진 기반의 고가・대형 플랫폼으로 초기 개발 단계에서의 위험도가 높다. 이에 본 연구에서는 제어 실패 및 추락 시에도 안전성을 확보할 수 있도록 소형 전기 추진 기반의 실험

플랫폼을 제작하였다. 유사한 접근 방식은 L. Xiang 및 Greener Space Program의 연구 사례에서도 확인할 수 있다. 또한 초기에는 동축 반전 프로펠러 기반 플랫폼도 고려하였으나, 충분한 추력 여유를 확보하기 어렵다는 한계로 인해 전기 덕트 팬 기반 설계를 최종적으로 채택하였다.

비행 컴퓨터는 기존에 널리 사용되는 Arduino 계열 MCU는 연산 성능이 부족하다고 판단하여, 고속 연산에 적합한 Teensy 4.1 계열 MCU를 사용하였다. IMU는 MPU-9250을 기반으로 하고, 고도, 위치 정보를 보완하기 위해 ToF 센서 및 GPS 모듈을 추가하였다 [2].

소프트웨어 구조는 일반적인 GNC 알고리즘 체계를 따르되, 본 연구에서는 유도(Guidance) 단계는 제외하였다. 항법(Navigation) 기능은 일반 UAV에서 널리 사용하는 알고리즘을 기반으로 구현하였으며, 관련 공개 소스들을 참고하여 필요한 모듈을 조합하였다. 제어(Control) 부분은 본 연구의 핵심으로, 추진기의 추력 크기를 조절하는 기존 멀티콥터 제어 방식과 달리, 추력 방향을 직접 조향하는 TVC 기반 제어 방법을 적용하였다. 이 기법은 실제 로켓에서 널리 사용되며 기존 문헌에서도 다수의 제어 알고리즘이 제안되어 있으나, 본 연구에서는 이를 실험 플랫폼에 맞게 수정 및 재구성하여 구현하였다 [3]. FixHawk, Dream Flight VTOL 등 기존 VTOL 제어 소프트웨어는 일부 기능만 참고하였으며, 전체적인 알고리즘 구조는 독자적으로 설계하였다.

Ⅱ. 이론적 배경

2.1 VTVL의 개념 및 안정성 특성

VTVL(Vertical Take-off and Vertical Landing) 기체란 추진기관으로 로켓 엔진을 사용하여 수직으로 이륙 및 착륙 기동을 수행할 수 있는 비행체를 의미한다. 이는 대기 흡입을 기반으로 양력 또는 추력을 생성하는 항공기형 VTOL(Vertical Take-off and Landing) 시스템과 구분되어 사용된다. VTVL은 자체 추진력만을 이용하여 비행 전 구간을 운용한다는 점에서 로켓형 비행체에 속한다.

대부분의 우주 발사체는 이륙 과정에서 수직으로 상승하는 형태를 가진다. 이때 추진축은 기체의 질량 중심선과 정렬되어 최대 추진 효율을 발휘한다. 비행 중 이를 안정적으로 유지하기 위해서는 정적 안정성(Static Stability)이 요구된다. 비행체의 질량 중심점이 압력 중심점보다 비행 방향으로 충분히 멀어질 때 정적 안정성은 확보되며 일반적으로 동체 직경을 기준으로 한다. 무게 중심점과 압력 중심점 사이의 거리를 동체의 평균 직경으로 나눈 값(무차원)을 정적 안정 여유(Static Margin)이라 하며, 그 값이 1 이상일 때 비행체가 불안정한 상태에 돌입하더라도 다시 안정적으로 되돌아올 수 있는 복원력(Restoring Force)을 가진다. 능동적인 제어수단이 없는 로켓에서는 일반적으로 정적 안정성을 1.5-2.5 사이로 권장한다 [2]. 복원력은 비행체의 항력에 의해 생기는 힘으로 동압(dynamic pressure)에 비례하며, 비행체의 동압은 속도 제곱에 비례하기 때문에 착륙과 같이 저속 기동이 요구되는 상황에서 복원력의 효과는 거의 미미하다.

따라서 수직으로 착륙을 하기 위해서는 능동적으로 안정성을 유지해야한다. 이를 동적 안정성(Dynamic Stability)이라 부르며 추력 벡터 제어(TVC, Thrust

Vector Control), 반작용 제어 시스템(RCS, Reaction Control System), 자세 제어 알고리즘(Attitude Control Algorithm) 등을 활용하여 동적 안정성 여유(Dynamic Stability Margin)를 확보해야 한다.

2.2 GNC 시스템의 구조

비행체를 능동적으로 안정화하고 원하는 궤적을 따라가도록 만드는 전체 알고리즘을 GNC(Guidance, Navigation and Control) 시스템이라고 한다. GNC는 유도(Guidance), 항법(Navigation), 제어(Control)의 세 부분으로 구분되며, 각 부분은 다음과 같은 역할을 수행한다.

유도는 시간에 따라 비행체가 어떤 상태(state)를 가져야 하는지를 결정하는 부분이다. 목표 착륙 지점, 목표 고도, 목표 자세 등의 기준 궤적(reference trajectory)을 생성하는 역할을 한다. 대형 발사체에서는 연료소모, 공력, 요구 제한 조건등을 고려하여 최적 궤적을 계산하는 복잡한 유도 알고리즘이 이용된다. 본 연구에서는 저속의 수직 이착륙을 수행하는 것이 목적이기 때문에 별도의 공력특성과 최적 궤적을 생성하는 알고리즘을 생략했고, 무선 조종기 혹은 컴퓨터를 통해 전송되는 목표 고도와 자세를 목표 상태(desired state)로 사용하는 수준의 유도만을 적용하였다.

항법(Navigation)은 시스템의 위치, 속도, 자세각, 시간등을 추정하는 것을 말한다. 비행체를 제어하기 위해서는 현재 비행체의 상태를 아는 것이 중요하다. 하지만 현실에서 비행체의 절대적인 상태를 아는 것은 불가능하기 때문에, 일반적으로 센서(sensor)를 이용하여 추정한다. 비행체와 같이 상태가 복잡한 시스템의 상태를 보다 정밀하게 추정하기 위해서는 여러개의 다양한 센서를 이용 할 수 있다. 본 연구에서 IMU, GNSS, ToF, 타이머는 항법을 위해 사용되는 센서이다. IMU는 3축 자이로, 3축 가속도, 3축 자기장을 측정하며, GNSS는 위도, 경도, ToF는 지면까지의 수직거리인 고도를 측정하며 타이머는 센서들의

시간을 동기화하는 역할을 한다. 각 센서는 샘플 레이트(Sample Rate), 측정 잡음 및 바이어스, 상대량 측정과 절대량 측정인지가 서로 다르다. 따라서 여러개의 센서를 종합하여 하나의 보다 정확한 추정값를 얻기 위해서는 센서 퓨전(sensor fusion) 알고리즘이 요구된다. 본 연구에서는 항법 알고리즘을 크게 두가지로 분류하였다. IMU를 이용한 수 kHz 수준의 고속 자세 추정값과, GNSS, 고도계를 종합한 수십 Hz 수준의 저속 위치, 속도 보정이다. VTVL의 상태 벡터는 ENU와 Body 좌표계에서 다음과 같이 정의한다. 여기서 좌표 체계는 2.3.1 좌표계 정의 및 기준 좌표 체계 설정을 따른다.

$$x = \begin{bmatrix} p^E \\ v^E \\ q_E^B \\ \omega^B \end{bmatrix} = \begin{bmatrix} x_E \\ y_E \\ z_E \\ u_E \\ v_E \\ w_E \\ q_0 \\ q_1 \\ q_2 \\ q_3 \\ p \\ q \\ r \end{bmatrix}$$

여기서 q_E^B, ω^B는 고속 자세 추정을 통해 추정하며, 위치인 p^E와 속도인 v^E는 저속 위치, 속도 보정을 통해 추정한다.

유도에서 도출된 desired state로부터 추정된 자세를 뺀 값을 상태 오차(State Error)라고 하며 이를 최소화하기 위한 액추에이터의 입력 명령을 계산하는 알고리즘을 제어(Control)라고 한다. 제어 가능한 액추에이터의 요소들은 메인 팬의 추력, TVC의 짐벌 각($\delta_{\theta,}\delta_{\psi}$), 롤 제어기의 추력이다.

3차원 상에서 이동하는 VTVL은 3개의 회전운동과 3개의 선형운동을 종합해, 총 6자유도를 가지고 있으나 4개의 제어 요소를 통해 6개의 자유도를 제어해야 하고, 이는 한 개의 제어 입력이 한 개 이상의 자유도에 관여해야 제어가 가능함을 알려준다. 롤 제어기의 경우 롤축의 회전운동에만 관여하지만, TVC의 각도와 메인팬의 스로틀링은 자세와 위치 등 다양한 요소에 관여한다. 따라서 제어기를 계층 구조로 구성함으로써 자세와 위치의 제어를 분리시켰다. 내부 루프는 자세 제어를 담당하며, 외부 루프는 위치와 고도에 관여한다.

2.3 VTVL의 수학적 모델링

2.3.1 좌표계 정의 및 기준 좌표 체계 설정

VTVL 기체의 위치・속도・자세를 수학적으로 기술하기 위해서는 먼저 기준 좌표계를 명확히 정의해야 한다. 이후에 나오는 수식들은 모두 기준 좌표계에 대해 유도, 전개되기 때문에 좌표계에 대한 정의와 표기가 중요하다. 일반적으로 항공우주에서는 다음과 같은 좌표계를 대표적으로 사용한다 [3].

- 지구중심 관성 좌표계(ECI, Earth centered inertial reference frame)
- 지구중심 지구고정 좌표계(ECEF, Earth centered earth fixed frame)
- 지역 수평 ENU 좌표계(Local-level ENU, East-North-Up)
- 동체 좌표계 (Body frame)

본 연구의 VTVL 기체는 비행 영역이 수십 m, 비행시간이 수십 초 이하의 실험 규모 가지고 있기 때문에, ECI/ECEF는 GNSS 데이터 해석의 배경으로 간단히 소개하며, 실제 동역학 및 제어 해석에는 지역 ENU 좌표계와 동체 좌표계만 사용한다.

지구중심 관성 좌표계(ECI, Earth centered inertial reference frame)이란 가속, 회전이 없는 좌표계를 의미한다. 실제 환경에서는 이러한 좌표계가 존재하지 않지만, 좌표계 간의 관계를 정의하기 위해 사용된다. 일반적으로 지구 중심을 좌표계의 중심으로 설정하고 Z축을 지구 자전축 방향으로 설정한다.

지구중심 지구고정 좌표계(ECEF, Earth centered earth fixed frame)는 관성 좌표계에 지구의 자전이 고려된 좌표계이다. 따라서 축은 ECI와 동일하며 t=0 인 지점에서 ECI와 일치한다. GNSS 수신기가 제공하는 위도・경도・고도는 WGS-84 타원체 기준의 ECEF 좌표를 기준으로 한다.

대형 발사체에서는 일반적으로 ECI,ECEF를 통해 항법 좌표계를 정의하는 방식을 이용하고 있으나, 본 연구에서는 이착륙점 주변의 작은 공간만을 고려하기 때문에 GNSS와 같이 ECEF를 이용하는 경우가 발생 할때에는 지역 수평 ENU 좌표계로 바로 변환한 뒤 사용한다.

본 연구에서는 항법 좌표계로 이착륙점을 기준으로 하는 지역 수평 ENU 좌표계(Local-level ENU, East-North-Up)를 사용한다. 원점은 이착륙 지점의 GNSS 기준위치이며, 오른손 좌표계에 따라 x_E: 동쪽, y_E 북쪽, z_E 지표를 기준으로 위를 향하는 방향으로 축이 정의된다.

GNSS 수신기는 위치를 위도・경도・고도 (φ, λ,h) 형태로 제공한다. 이때 이착륙점 기준의 상대 위치를 ENU로 선형 근사하면, 라디안 단위의 각 차이를 $\Delta\varphi = \varphi - \varphi_0, \Delta\lambda = \lambda - \lambda_0$라 할 때, 지구 반경을 R_E로 근사하면 다음과 같이 표현된다.

$$
\begin{aligned}
x_E &\approx R_E \cos\varphi_0 \, \Delta\lambda \\
y_E &\approx R_E \, \Delta\varphi \\
z_E &\approx h - h_0
\end{aligned}
$$

여기서 x_E가 동쪽, y_E는 북쪽 방향을 나타낸다. 수십 미터 이내에서 시험되는 실험 특성상 위도와 경도의 차이는 매우 작기 때문에 선형으로 근사하여도 충분한 정확도를 얻을 수 있다. 결과적으로 본 연구에서는 ENU 좌표계(x_E, y_E, z_E)를 항법 좌표계로 사용하며, 이후 동역학과 제어의 모든 수식은 이 ENU를 기준으로 한다.

동체 좌표계 (B, Body frame)는 기체에 고정되어있는 좌표계이다. VTVL이 비행중 거의 수직으로 유지된다는 점을 고려하여 원점은 기체의 질량중심(CG), 축은 오른손 좌표계를 사용하며, x_B가 비행체의 앞부분인 상단방향을 지칭한다. 이는 수직상태에서 $+z_E$와 일치한다. y_B는 오른쪽 방향이며 z_B는 나머지 한 방향이다. 자세를 표현하는 각도는 이에 따라 x_B를 중심으로 회전하는 롤(ϕ), y_B의 피치(θ), z_B의 요(ψ)이다. 일반적인 항공이나 발사체의 좌표체계와 비슷하나, 동체가 대부분 수직으로 서 있다는 점에서 일반적으로 전방인 x_B가 수직인 차이가 있다 [2].

2.3.2 자세 표현 체계 (Attitude Representation)

비행체의 자세를 표현할 때는 기본적으로 두 기준 좌표계 사이의 상대적인 방향(orientation)을 표시한다. 이는 일반적으로 행렬을 통해 표현되며 행렬 사이의 변환에 관한 자세한 전개, 유도는 생략한다. 기체를 기준으로 하는 동체 좌표계의 각 축을 관성 좌표계의 단위 벡터로 나타내면 총 9개의 스칼라 성분이 나온다. 이것을 행렬로 표현하면 다음과 같이 표현된다.

$$C_I^B = \begin{bmatrix} i_1 & i_2 & i_3 \\ j_1 & j_2 & j_3 \\ k_1 & k_2 & k_3 \end{bmatrix}$$

이 행렬은 방향 코사인 행렬(DCM, Direction Cosine Matrix)이라 부르며 기호로는 C를 사용한다. 위 첨자의 좌표계에서 아래첨자의 좌표계로의 회전 변환을 나타낸다. 이 행렬은 각 벡터들이 수직이며, 크기가 1로 정규화 된 정사각 행렬이기에 전치행렬이 역행렬과 같다.($C^{-1} = C^T$, 정규 직교 행렬) 또한 연속적인 회전을 표현할 때 각 회전 행렬의 곱으로 표현할 수 있다. 관성 좌표계에서의 임의의 벡터 $\hat{v_I}$를 기체 좌표계로 변환할 때 방향 코사인 행렬에 $\hat{v_I}$를 곱함으로써 기체 좌표계로 표현된 $\hat{v_I}$인 $\hat{v_B}$구할 수 있다.

$$C\hat{v_I} = \begin{bmatrix} i_1 & i_2 & i_3 \\ j_1 & j_2 & j_3 \\ k_1 & k_2 & k_3 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} v_i \\ v_j \\ v_k \end{bmatrix} = \hat{v_B}$$

이로써 방향 코사인 행렬은 정해진 값이 아니기 때문에 적용에 따라 매개변수화 되어 표현된다. 그중 가장 대표적인 방식이 오일러 각(Euler angle)과 쿼터니언(Quarternion)이다.

2.3.2.1 오일러 각(Euler angle) 체계

오일러 각(Euler angle)은 세 번의 축 회전으로 기준 좌표계에서 동체 좌표계로의 회전을 표현하는 방법이다. 본 연구에서는 Z-Y-X(yaw-pitch-roll) 순서를 사용한다. EMU에서 Body로의 회전 행렬은 다음과 같이 표현된다.

$$C_E^B = R_x(\phi) R_y(\theta) R_z(\psi)$$

각 단일 회전 행렬은 다음과 같다.

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

이를 곱하면 C_E^B의 각 성분을 ϕ, θ, ψ의 삼각함수로 전개할 수 있다. IMU 센서의 자이로는 동체 좌표계 각속도 $\omega^B = [p, q, r]^T$를 측정한다. 여기서 p,q,r은 각각 롤, 피치, 요에 대한 회전 속도이다. Z-Y-X의 순으로 오일러 각에 대해 다음 관계가 성립한다.

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \cos\theta\sin\phi \\ 0 & -\sin\phi & \cos\theta\sin\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

이 식을 통해 롤, 피치, 요 회전의 미소 회전축을 모두 동체 좌표계에서 표현한 뒤 더 구하는 방법으로 유도할 수 있다. 실제로는 이 행렬을 역행렬로 나타내어 반대를 구하고, 이를 적분하여 오일러 각을 구할 수 있다.

오일러 각은 직관적인 장점이 있으나 θ =±90 부근에서 행렬중 축이 겹쳐 차원을 상실하는 짐벌락(gimbal lock) 문제가 있다. 따라서 내부 계산은 쿼터니언 체계를 이용하여 계산하고, 오일러 각은 추후 분석이나 표시용으로만 사용한다.

2.3.2.2 쿼터니언(Quaternion) 체계

쿼터니언(quaternion)은 3차원 회전을 4개의 스칼라 성분으로 표현하는 자세 표현 방법으로, 오일러 각과 달리 짐벌락(gimbal lock)이 발생하지 않으며 수치적 안정성이 우수하다는 장점이 있다. 쿼터니언은 하나의 스칼라 성분과 3차원 벡터 성분으로 구성되며 다음과 같이 정의된다.

$$q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos\frac{\alpha}{2} \\ e\sin\frac{\alpha}{2} \end{bmatrix}$$

여기서 q_0은 회전 각도 정보를 가지는 스칼라 성분이며, $q_1 q_2 q_3$은 축의 방향과 각도를 함께 가지는 벡터이다. 따라서 이를 회전축-회전각으로 표현할 경우 뒤의 행렬과 같이 표현되며, 각도 체계의 특성상 반각으로 표현된다. α는 회전 각도, $e = \begin{bmatrix} e_x e_y e_z \end{bmatrix}^T$는 회전축을 나타내는 단위벡터이다. 단위 쿼터니언의 경우 항상 다음의 정규화 조건을 만족한다. 이 조건으로 인해 사원수인 쿼터니언의 자유도 한 개가 제거되며, 3개의 독립적인 자유도로 짐벌락 없이 자세 표현이 가능하다.

$$\| q \|^2 = q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$$

쿼터니언을 이용하면 동체 좌표계와 관성 좌표계 사이의 회전 변환을 간결하게 표현할 수 있다. ENU 좌표계 상의 벡터를 동체 좌표계 상의 벡터로 변환하기 위해서는 ENU 좌표계에 쿼터니언으로부터 유도된 방향 코사인 행렬(Direction Cosine Matrix, DCM)을 곱해 변환할 수 있다. 회전 쿼터니언과 회전 쿼터니언의 켤래를 곱함으로써 쿼터니언의 회전을 표현할 수 있으며, 이를 3개의 축에 적용하여 표현하면 각 열이 동체 자표계의 x,y,z가 되는 DCM이 유

도된다.

$$C_E^B = \begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2) \\ 2(q_1q_2+q_0q_3) & 1-2(q_1^2+q_3^2) & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & 1-2(q_1^2+q_2^2) \end{bmatrix}$$

IMU의 자이로 센서는 동체 좌표계 기준의 각속도를 측정한다. 순간 각속도는 회전 행렬의 시간 변화율에 해당함으로 측정값을 회전 행렬로 변환하는 미분 방정식을 다음과 같이 표현할 수 있다. 쿼터니언은 반각으로 표현되는 체계이므로 1/2을 곱하고, 각 속도 행렬에 $\Delta t \rightarrow 0$을 곱한 미소 회전을 각도 쿼터니언에 곱하여 각도 변화율을 구할 수 있다.

$$\dot{q} = \frac{1}{2} \begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix} q$$

이 미분 방정식을 수치적으로 적분함으로써 연속적인 자세 추정이 가능하다. 다만 적분 과정에서 수치 오차로 인해 쿼터니언이 정규화 조건를 벗어날 수 있다. 따라서 매 연산마다 정규화(normalization)를 수행하여 다음 식이 성립하는 조건 찾아 수치 오차를 최소화한다.

$$q \leftarrow \frac{q}{\| q \|}$$

본 연구에서 자세 추정 및 제어 연산 알고리즘에서는 쿼터니언 체계를 사용하였다. 고속 회전, 큰 각도 기동, 수직 착륙 과정에서 오일러각 체계 대비 수치적으로 안정적이며 특이점을 방지할 수 있기 때문이다. 반면, 이용자가 직관적으로 이해하기 쉬운 오일러 각(롤-피치-요)은 쿼터니언으로부터 변환하여 로그 기록 및 시각화 용도로만 사용하였다 [3].

2.3.3 VTVL의 동역학 모델링

본 연구의 VTVL은 비행 중 구조적 변형이 거의 없으며, 계산의 편리성을 위해 기체를 강체(rigid body)로 가정할 수 있다 [2]. 따라서 기체의 운동은 강체의 병진 운동과 회전 운동을 결합한 6자유도(6-DOF) 운동 방정식으로 모델링된다. 본 절에서는 ENU 좌표계를 기준으로 한 병진 운동 방정식과, 동체 좌표계를 기준으로 한 회전 운동 방정식을 각각 유도한다.

2.3.3.1 병진 운동 방정식 (Translational Dynamics)

기체의 질량 중심(CG)의 위치를 $p_{E=}\left[x_E y_E z_E\right]^T$, 속도를 $V_E = \dot{P}_E$라 하면, 뉴턴의 제2법칙에 따라 병진 운동 방정식은 다음과 같이 표현된다.

$$m\dot{V}_E = \sum F_E$$

여기서 m은 기체의 질량, $\sum F_E$는 기준 좌표계(ENU)에서 작용하는 외력의 합이다. 본 연구에서는 저속, 저고도 실험 환경을 가정하므로 공력적 항력 및 양력은 1차 모델에서는 무시하고, 주요 외력으로 중력과 추력만을 고려한다.
중력은 ENU 좌표계에서 다음과 같이 표현된다.

$$F_g = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix}$$

메인 덕트팬이 발생시키는 추력은 동체 좌표계의 $+x_B$ 방향으로 작용하고 이를 ENU 좌표계로 변환하면 다음과 같다.

$$F_T^E = C_E^B \begin{bmatrix} T \\ 0 \\ 0 \end{bmatrix}$$

최종적으로 VTVL의 병진 운동은 다음과 같이 정리된다.

$$m\dot{v_E} = F_T^E + F_g$$

기체의 자세와 추력의 크기에 따라 기의 위치와 속도가 결정될 것이다.

2.3.3.2 회전 운동 방정식 (Rotational Dynamics)

회전 운동은 동체 좌표계 기준의 각속도 $\omega_B = [p\ q\ r]^T$와 관성 모멘트 행렬 I_B를 이용해 표현한다. 강체의 회전 운동 방정식은 오일러 방정식(Euler's equation)으로 주어진다.

$$I_B\dot{\omega}_B + \omega_B \times (I_B\omega_B) = \tau_B$$

여기서 τ_B는 동체 좌표계에서 작용하는 총 모멘트 벡터이다. 본 연구의 VTVL 기체는 질량 분포가 축대칭에 가깝기 때문에, 관성 모멘트 행렬은 다음과 같은 대각 행렬로 근사할 수 있다.

$$I_B = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix}$$

회전 운동 방정식은 제어 입력이 실제 기체의 각속도 변화에 영향을 주는 것을 나타낸다.

2.3.4 추력 벡터 제어가 적용된 비행체의 동역학

VTVL 기체의 자세 제어는 메인 추진기의 추력 방향을 편향시키는 추력 벡터 제어(Thrust Vector Control, TVC)를 통해 이루어진다. TVC는 추력의 크기는 유지한 채 방향을 변화시켜 모멘트를 발생시키는 방식으로, 저속에서도 안정적인 제어가 가능하다는 장점을 가진다.

2.3.4.1 추력 편향에 따른 힘 벡터

메인 덕트팬의 추력이 피치 및 요 방향으로 각각 $\delta_{\theta,}\delta_{\psi}$만큼 편향되었다고 가정하면, 동체 좌표계에서의 추력 벡터는 다음과 같이 표현된다.

$$F_T^B = T\begin{bmatrix}\cos\delta_\theta\cos\delta_\psi \\ \sin\delta_\psi \\ -\sin\delta_\theta\end{bmatrix}$$

또한 편향각이 충분히 작을 경우 계산의 편의성을 위해 다음과 같이 선형 근사가 가능하다. VTVL TVC 짐벌의 가동 범위는 30도를 넘지 않는 것이 일반적이므로, 실제로는 아래와 같이 연산한다.

$$F_T^B \approx T\begin{bmatrix}1 \\ \delta_\psi \\ -\delta_\theta\end{bmatrix}$$

2.3.4.2 TVC에 의해 발생하는 모멘트

추력 벡터 제어(TVC)를 통해 자세를 제어하기 위해서는 추력의 작용선(line of action)이 기체의 질량 중심(CG)을 통과하지 않아야 한다. 만약 추력 작용선

이 질량 중심점을 통과한다면, 추력 편향은 다른 방향으로의 병진력만을 발생시킨다. 따라서 추력기는 짐벌에 장착되어 구동되는 짐벌 점(gimbal pivot)을 중심으로 구동된다. 따라서 TVC는 기체의 CG와 짐벌 점과의 거리를 레버로 하여 회전 모멘트를 생성한다.

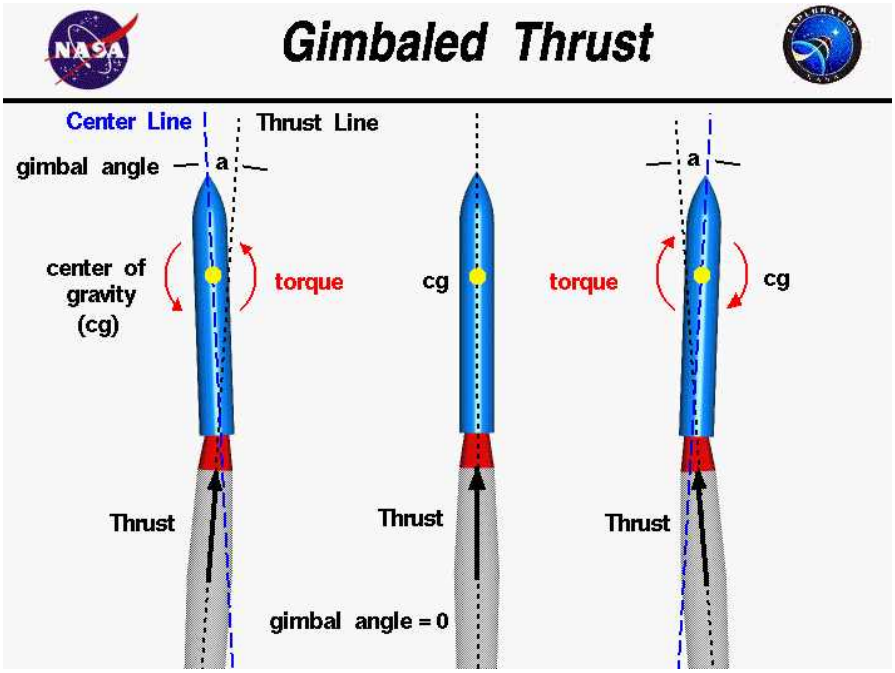


Fig. 2. TVC와 CG의 관계

추력 작용점과 질량 중심 사이의 거리 벡터를 $r_T = [0\,0\,-l]^T$라 하면, 추력에 의해 발생하는 모멘트는 다음과 같다.

$$\tau_{TVC} = r_T \times F_T^B$$

이를 전개하면, 피치 및 요 축에 대한 제어 모멘트가 추력 편향각에 비례함을 확인할 수 있다.

$$\tau_y \approx l T \delta_\theta, \quad \tau_z \approx l T \delta_\psi$$

제어시 추력기의 짐벌 각도와 실제 기체 각도가 선형적으로 비례하는 것을 의미한다. 따라서 제어시 짐벌의 각도와 기체 각도사이의 비례관계 인자 하나만 찾을 경우 제어가 가능하다.

2.3.4.3 롤 제어

추력 벡터 제어(TVC)는 추력의 방향을 편향시켜 질량 중심에 대해 모멘트를 발생시키는 방식이지만, 단일 추진기를 사용하는 축대칭 구조의 VTVL 기체에서는 TVC만으로는 추력 작용선과 동일한 선상에 위치하는 롤(roll) 축에 대한 제어가 불가능하다. 앞선 2.5.2절에서 유도한 바와 같이, TVC에 의해 발생하는 모멘트는 추력 작용점과 질량중심 사이의 레버 암에 의해 결정된다. 그러나 롤 축의 경우, 추력 벡터를 편향하더라도 그 작용선은 여전히 동체의 종방향 축과 평행하게 유지된다. 결과적으로 TVC로 인한 롤 방향의 모멘트는 생성되지 않는다.
따라서 본연구에서는 롤 축을 효과적으로 제어하기 위해서 추가적인 롤 축 제어기를 부착할 것이다. 덕트팬을 사용하는 기체에서는 팬이 구동 중 일시 팬의 회전에 의해 그 반대 방향으로 롤 모멘트가 발생한다. 따라서 롤 제어기는 이를 상쇄 시키는 방향으로 같은 크기의 모멘트를 발생시키면 기체의 롤 축은 정적일 것이다. 또한 롤 축 제어기의 모멘트 크기를 높이거나 낮춤으로써 양방향에 대한 제어가 가능하다.

이와 같이 제어 입력을 축별로 분리함으로써, 각 축에 따른 별도의 자세 제어기를 구성하고 운영할 수 있다. 각 축 간의 상호작용만 고려하여 계층적인 제어기를 설계할 시 전체 역학을 동시에 고려하는 것 대비 비교적 낮은 난이도로 구현할 수 있다는 장점이 있다.

Ⅲ . 하드웨어 모델링

3.1 추진체계의 선정

RC(Remote Control)용 비행체에 적용되는 덕트팬은 높은 추력을 내면서도 기타 연구용/산업용에 적용되는 덕트팬 대비 저렴하다. 그중 QX-motor의 제품으로 선정하였다. EDF는 흡입구의 직경을 기준으로 제품이 분류되었으며, 30mm, 50mm, 64mm, 70mm, 80mm, 90mm의 크기로 판매되었다. 본 연구에서는 배터리, BLDC(브러시리스다이렉트커런트) 모터 구동용 인버터, 비행제어 컴퓨터 등이 탑재 되어야 하며, 추후 연구용 장비의 탑재를 고려하여 여유를 두는 것이 적합하다고 판단하였고 따라서 40N 이상의 추력을 발휘하는 90mm 버전으로 선정하였다. 자세한 제원은 다음과 같다. [3]

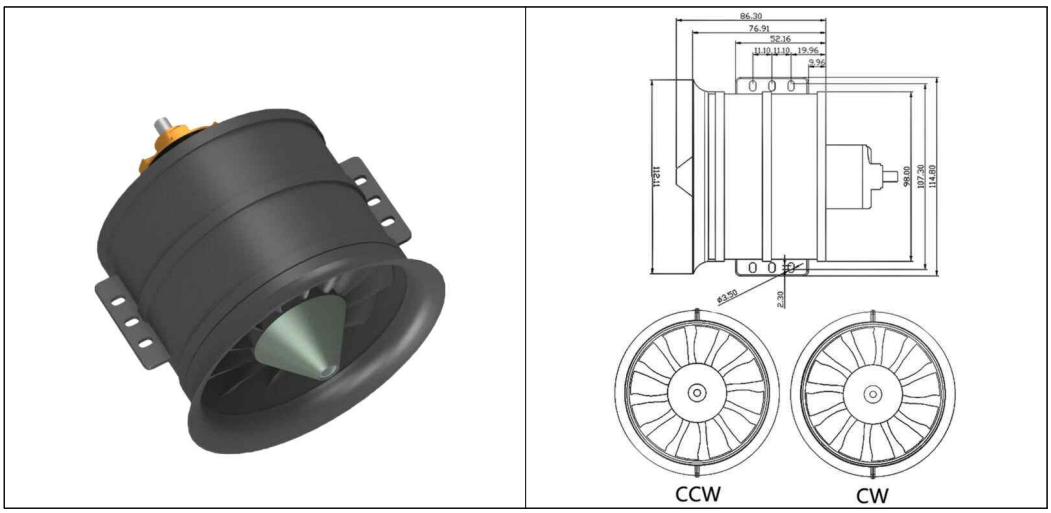


Fig. 3. QX-motor 90mm EDF

모델 명	QX-motor 90mm EDF
모터 모델명	QF3758-1200KV
무게 (g)	430
블레이드 수 (개)	12
전압 (V)	29.6
전류 (A)	93.0
전력 (W)	2752.8
최대 추력 (N)	44.13
회전수 (RPM)	35,520
효율 (g/W)	1.63

Table 1. QX-motor 90mm EDF 제원

메인팬이 생성하는 토크를 상쇄하여 롤 축을 제어하기 위한 보조 EDF를 선정해야 한다. 메인팬이 생성하는 토크는 명시되어있지 않고, 측정할 수 없기 때문에 명시되어있는 값을 통해 추정할 수 있다. 메인팬이 생성하는 최대 토크는 정상 회전시 발생하는 부하 토크와 가속시 발생하는 가속 토크의 합으로 표현될 수 있다. 계산을 위해 최대 속도까지의 가속시간은 0.5초로 추정하였으며, 회전자의 질량은 100g, 모터의 직경은 37mm, 모터의 기계 효율은 0.8로 가정하였다. 롤 제어기가 발생시키는 토크는 상쇄시켜야 하는 토크 이상이 되어야 제어가 가능하며, 상쇄키여야 하는 토크의 2배가 되었을 때 제어축의 양측 회전 방향에 대한 가속/감속률이 동일해지기 때문에 약 2배를 목표치로 선정했다. 롤 제어기의 요구 토크는 다음과 같다.

$$T_d = 2(\frac{P}{\omega_r} \cdot \eta + (\frac{1}{2} m_r r_r^2) \cdot \frac{\omega_r}{\Delta t_{accel}})$$

(여기서 P는 모터 출력W, T_d는 요구토크 Nm, ω_r는 각속도 rad/s, η는 모터 효율, m_r은 모터 질량 kg, r_r은 로터 반지름 m, Δt는 가속시간 s)

위 수식에 따라 롤 제어기는 0.425 Nm 이상의 추력을 낼 수 있어야 한다. 동체 직경이 180mm임에 따라 보조 EDF가 중심축으로부터 100mm 떨어져 있다

고 가정하였고, 이때 두 개 중 하나의 EDF는 2.125 N 이상의 추력을 내야한다. 따라서 2.16 N의 최대추력을 가지는 Qx-motor의 30mm 3S 7000KV EDF를 보조 EDF로 선정하였다.

Fig. 4. Qx-motor의 30mm 3S 7000KV EDF

3.2 TVC 하드웨어 디자인

TVC란 추력을 편향하여 롤, 피치 모멘트를 만들어 내는 장치로 고정노즐의 기계식 편향, 2차 분사 방식과 가동 노즐로 나누어진다. 그중 액체 로켓에 대표적으로 사용되는 방식은 짐벌(Gimbal)로, 추력기 전체를 볼 조인트(ball-joint)나 베어링을 중심으로 회전시키는 구조이다. 구조적 단순함과 제어의 신뢰성 등 다양한 이유로 현재 대부분의 액체 연소기가 채택한 방식이다.

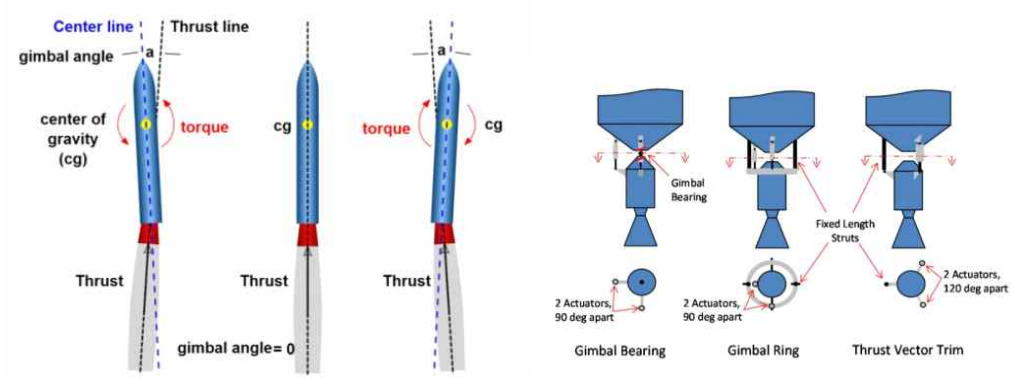


Fig. 5. TVC 짐벌의 구조

본 연구는 액체 로켓 추진 기관에서의 적용을 염두하고 있기 때문에 짐발 방식을 채택하였다. 그중 연소실이 하나의 점에 베어링을 통해 연결되는 방식으로 하였으며, 액추에이터는 서로 90도를 이루어 2개의 축에 대한 독립적인 제어가 가능하고록 하였다.

엔진을 실질적으로 짐벌링 시키는 액추에이터는 일반적으로 선형 운동을 하는 액추체이터가 사용된다. 하지만 리니어 서보 모터는 높은 가격을 형성하고 있으며, 본 연구에 적용될 수 있는 크기의 제품은 많지 않았다. 비교적 낮은 가격과 함께 소형으로 판매되는 제품을 찾을 수는 있었으나, 브러쉬드 DC모터를 사용함에 따라 내구성, 신뢰성의 문제가 있었고 무엇보다 현재 상태를 알 수 있는 엔코더와 같은 장치가 없어 오픈루프의 형태로 사용되어야 했다. 따라서 저렴한 가격과 크기, Closed loop를 형성하고 있는 회전형 서보 모터를 사용하였다.

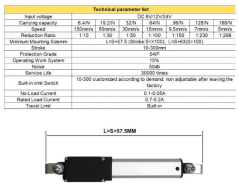
사진			
명칭	Ultra Motion AM Series	중국제요	DS 3230 PRO
가격대	KRW 2,000,000~	KRW 12,000~	KRW 14,000~

Fig. 6. TVC 구동 액추에이터 비교

3.2.1 TVC 기구학

회전형 액추에이터의 회전 운동을 기계적 변환을 통해 선형 운동으로 치환하여 적용하는 방식은 실제 짐벌(gimbal)과 유사한 운동 특성을 구현할 수 있다는 장점을 지닌다. 그러나 이와 같은 변환 기구는 필연적으로 추가적인 부품, 질량 증가, 그리고 구조적 복잡성을 수반한다. 본 연구에서는 이러한 문제를 최소화하고, 구조적 단순성과 경량화를 동시에 달성하기 위해 회전 운동을 직접적으로 추력기 편향에 활용하는 방식을 채택하였다.

구조적으로는 서보모터에 부착된 서보혼이 푸시로드 링크를 통해 EDF 하우징을 직접 밀고 당기는 형태로써, EDF는 상단부에 설치된 볼 조인트를 중심으로 회전하도록 설계되었다. 이러한 형태는 운동학적으로 Crank-Rocker 형상의 4절 링크(Crank-Rocker Four-bar Mechanism)로 분류된다.

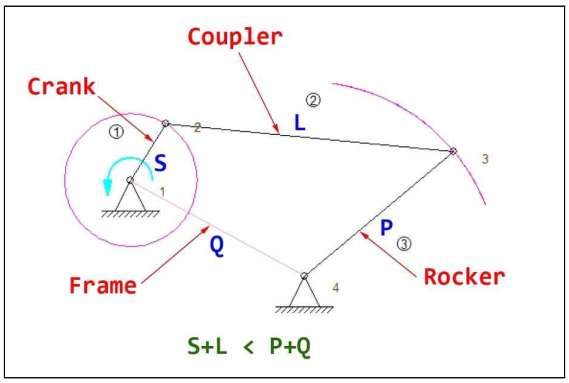


Fig. 7. Crank-Rocker 4절 링크 구조

본 연구에 사용된 링크의 기하학적 파라미터는 서보혼 길이(크랭크 길이) 24mm와 동체의 끝을 기준으로 로커의 회전중심인 볼 조인트는 하향으로 20mm에 위치해있으며, 서보모터의 회전축은 상단으로 50mm, 좌측으로 40mm에 위치해있다. 덕트팬 형상에 따라 로커의 길이는 100mm, 초기 위치가 중심선을 기준으로 -37.9도 기울어진 형상을 가진다. 초기 파라미터는 Fig. 8. 과 같다.

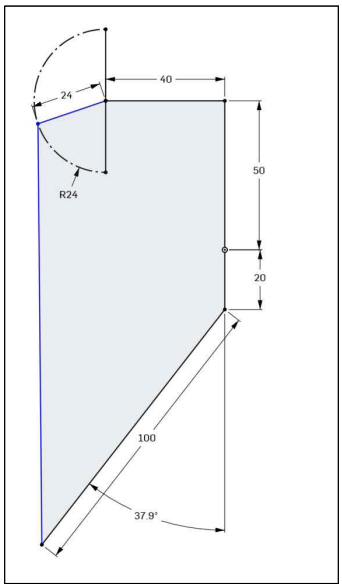


Fig. 8. TVC 짐벌의 초기 파라미터

일반적으로 TVC는 ±15° 에서 ±20° 정도의 편향각을 가진다. 본 연구는 구조적인 간섭 없이 충분한 피치, 요 토크를 확보하기 위해 ±20° 를 편향 각으로 지정했다. 서보모터는 최대 180° 의 운동을 할 수 있으며, 크랭크와 로커 각도 사이의 선형성이 크게 감소되지 않으면서 충분한 토크를 전달할 수 있도록 서보모터는 ± 60° 의 120도 유효 스트로크를 가지도록 하였다.

4절 링크는 크랭크의 입력 각도와 로커의 출력 각도가 비선형적으로 비례하기에 단순 수치 대입을 통한 최적의 커플러 길이 특정은 어렵다. 링크의 길이와 초기 위치에 따라 해가 존재하지 않거나, 예상하는 조립체계가 변형되는 문제가 발생할 수 있기에 링크 길이와 크랭크 중심각을 변수로 둔 Nelder-Mead 기반 비선형 수치 옵티마이저를 설계하였다.

목적함수는 다음과 같다.

$$f(L_2, \phi_0) = (\theta_{center} - \theta_{target})^2 + (|\Delta\theta_{up}| - 20^\circ)^2 + (|\Delta\theta_{down}| - 20^\circ)^2$$

여기서 세타_센터는 크랭크 극단 위치에서 로커각의 평균이며, 델타세타업, 델타세타다운은 중심각 대비 최대, 최소의 편차이다. 따라서 목접 함수가 최소화되는 설계 변수를 찾으면 된다. Python에서 기본 제공되는 scipy.optimize 라이브러리를 통해 Nelder-Mead를 구현했으며 이는 부록 1에 첨부되어 있다. 결과는 다음과 같다.

```
=== Optimization result (L4 = 100 mm, theta_center = -127.9 deg) ===
success            : True
L2 (coupler)       : 142.808800 mm
phi0 (deg)         : -164.937917
rocker center      : -127.900000 deg (target = -127.9)
rocker up dev      : +20.000000 deg (target ≈ +20.0)
rocker down dev    : -20.000000 deg (target ≈ -20.0)
```

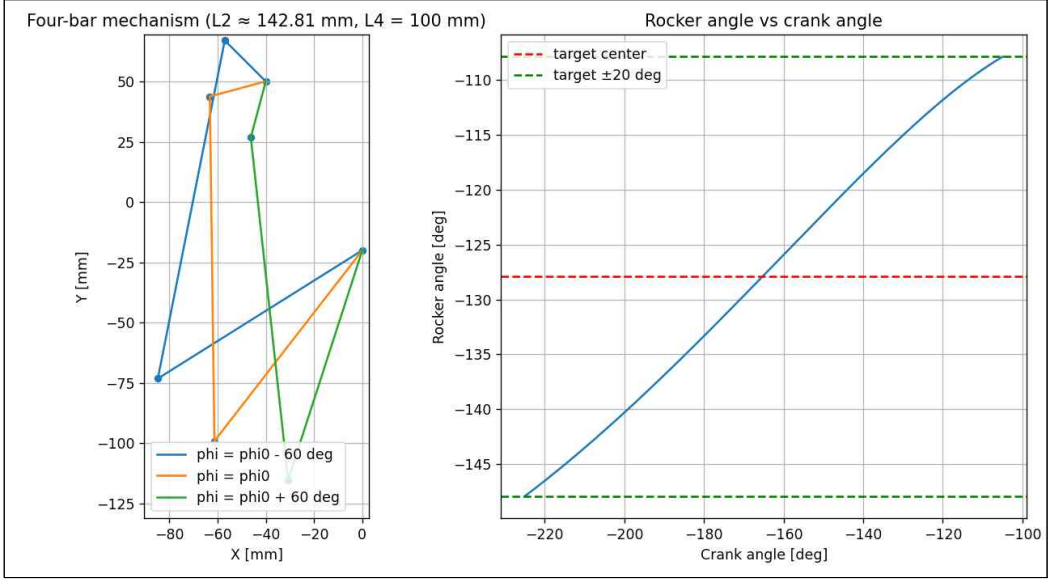


Fig. 9. 파이썬 기반 링크 길이 옵티마이저 결과 값

최적화된 수치를 분석한 결과 커플러의 길이는 142.8088mm가 나왔으며, 크랭크의 초기 각도는 -164.94도임을 확인하였다. 또한 크랭크의 각도와 로커의 각도가 선형에 가깝게 나왔음을 확인하였다. 따라서 추후 제어 시 서보모터의 각도와 메인팬의 각도가 비례한다고 가정하여 제어할 수 있다. 최종 푸쉬로드의 길이는 계산값을 반올림한 143mm로 선정하였다. 앞선 결과를 종합하여 설계한 TVC 마운트의 CAD(Computer-Aided Design)는 Fig. 10.과 같다.

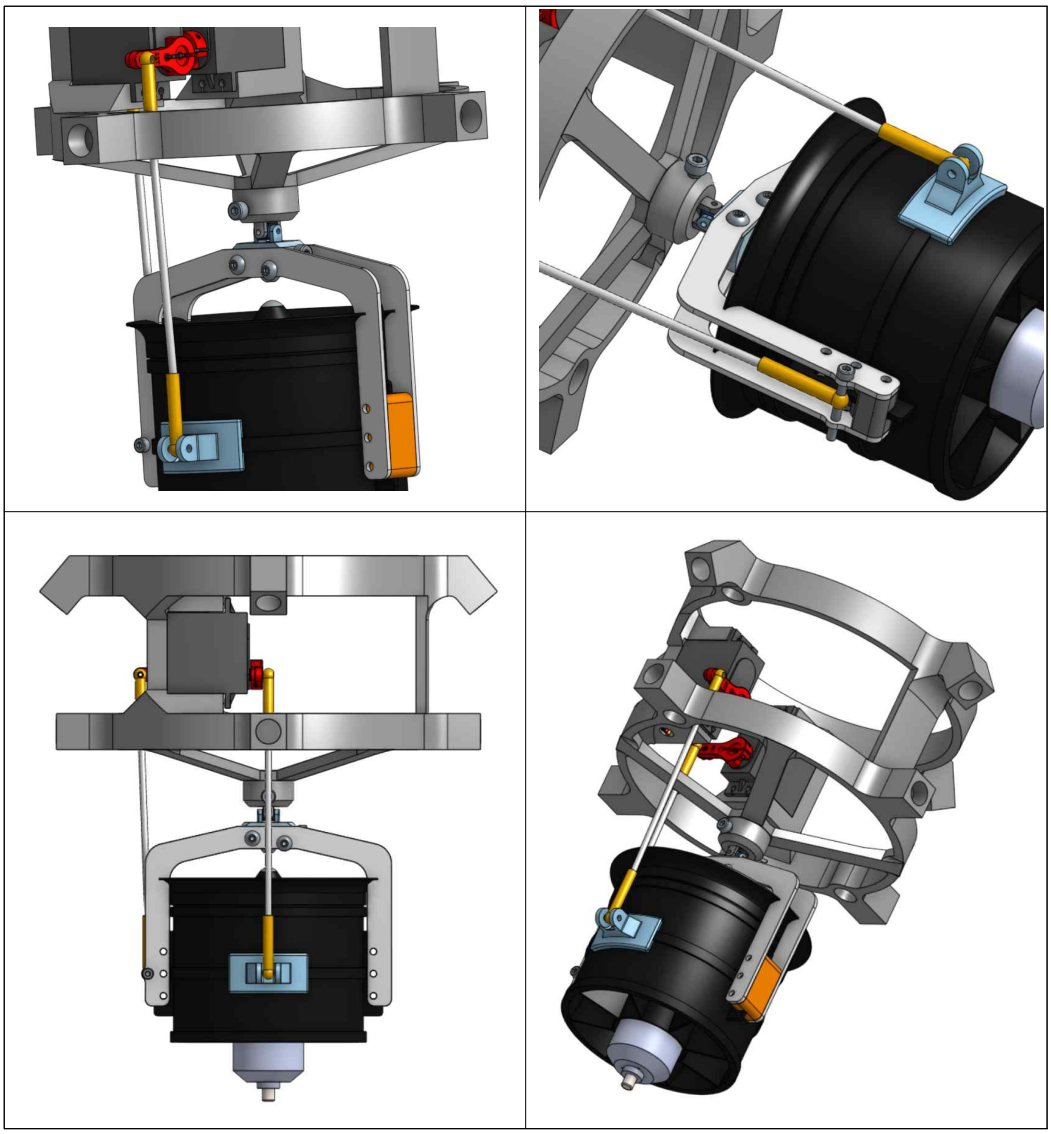


Fig. 10. 최종 TVC 짐벌 액추에이터 CAD

동체는 짐벌 구동부 이상의 직경으로, 180mm로 설계하였으며 짐벌 마운트에 착륙을 위한 다리와 상단에 제어기를 장착 할 수 있도록 파이프를 부착할 수 있는 구멍을 추가하였다. 경량성을 위해 다리와 동체는 12mm 탄소섬유 파이프로 구성하였다.

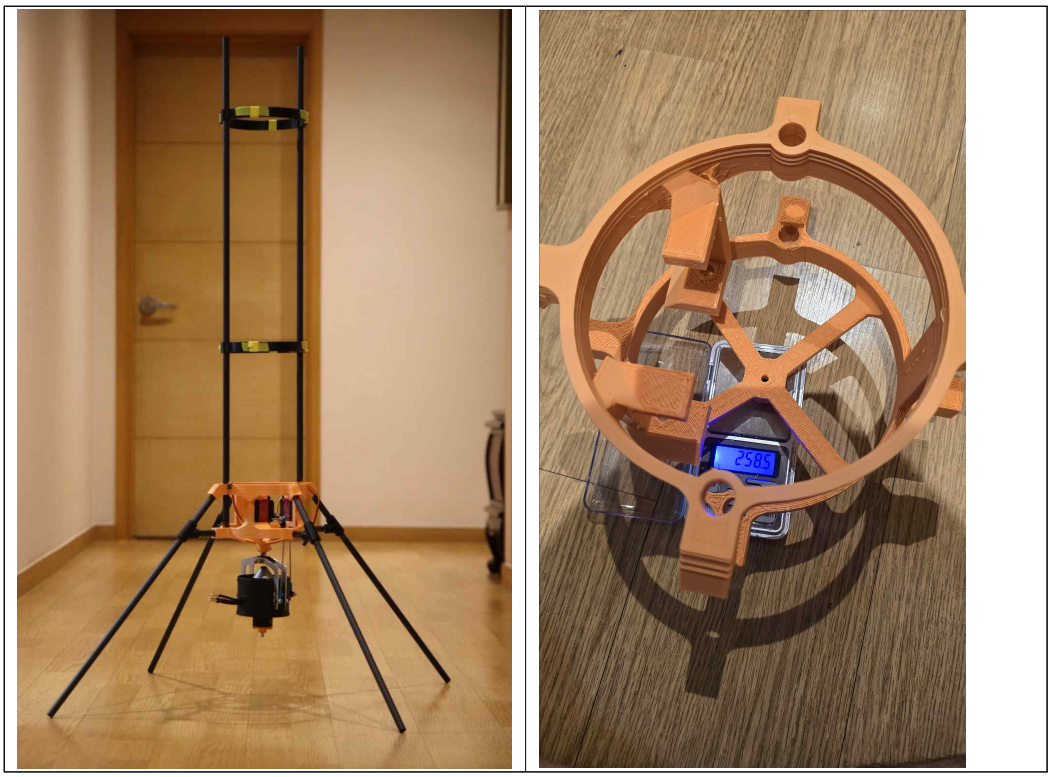


Fig. 11. 완성된 TVC 마운트 사진

3.2.2 마운트 FEA

완성된 CAD(Computer-Aided Design)가 충분한 강도를 가지는지에 대하여 응력을 FEA(Finite element analysis) 해석하였다. Simscale을 이용하여 Dynamic 해석을 진행하였으며, 물체는 기본 제공된 PLA, 솔버로는 MUMPS (MUltifrontal Massively Parallel sparse direct Solver)가 이용되었다. 상단 면을 Fixed support로 지정하고 하단 볼 조인트 마운트에 50N의 Force를 부여하였다. FEA의 결과는 다음과 같다.

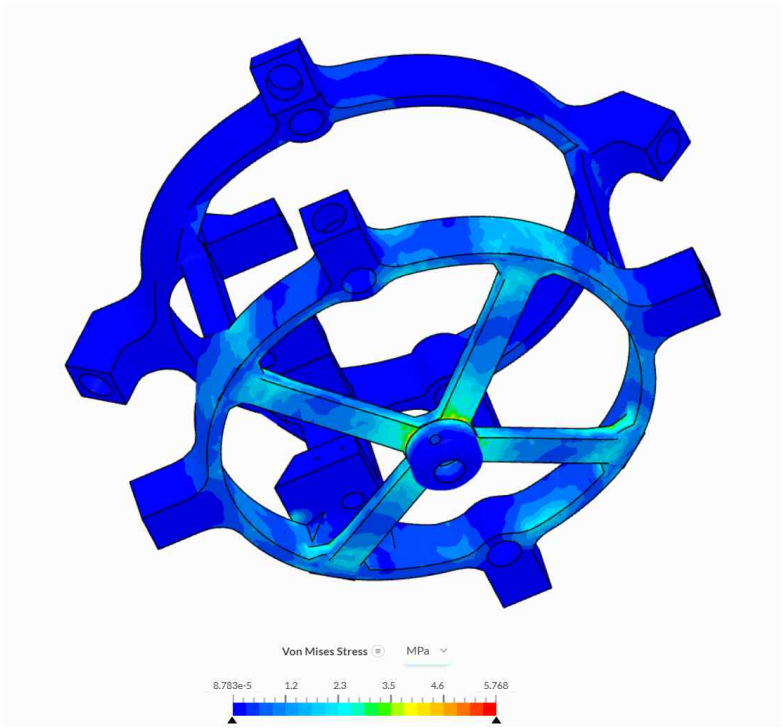


Fig. 12. FEA의 Von Mises Stress 분포

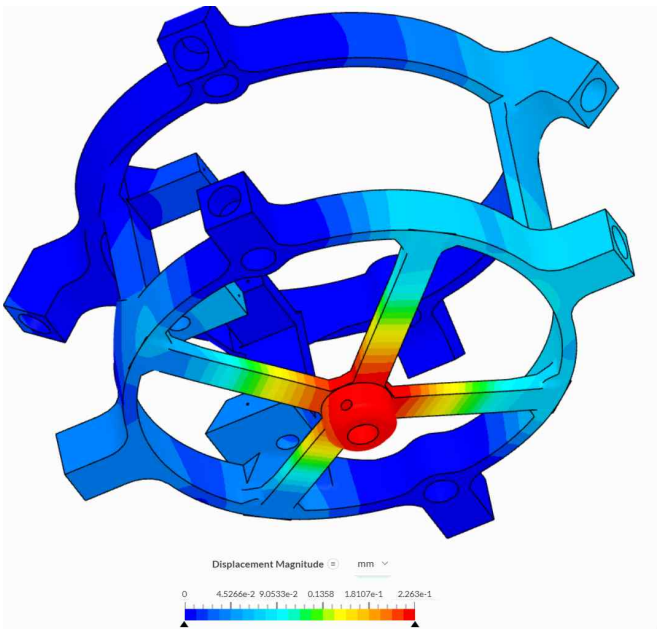


Fig. 13. FEA의 변위 분포

FEA의 결과, Von Mises Stress는 평균적으로 0.25MPa, 최대 5.768 MPa를 보였다. PLA의 항복강도는 상온에서 약 65MPa으로 알려져 있으며, 적층 제조임을 감안하여도 충분한 강도가 확보되었음을 확인하였다. 또한 최대응력이 볼조인트 마운트와 이를 고정하는 지지대 사이의 모서리에 작용하고 있다. 이는 모서리에 곡률을 주는 것으로 해결이 가능하며, 우려됐던 지지대에는 큰 응력이 작용하지 않았기에 추가적인 보강없이 부품을 사용하기로 결정하였다. 힘이 작용할 때 변위는 평균 0.03mm, 최대 0.23mm를 보였으며, 대부분 지지대 이후에 발생하기에 파손 시 지지대가 먼저 파손되어 상단 구조물에 큰 손상을 주는 것을 막을 수 있을 것으로 생각된다.

3.3 전자제어 하드웨어 선정

VTVL 기체의 자세 추정 및 제어 알고리즘을 안정적으로 수행하기 위해, 연산 성능과 확장성을 기준으로 비행 컴퓨터를 선정하였다 [4]. Teensy 4.1은 ARM Cortex-M7 기반의 고속 MCU로, 고속 제어 루프와 센서 융합 알고리즘을 실시간으로 처리하기에 충분한 연산 성능을 가지고 있으며, 타 보드 대비 다양한 I/O 핀을 가지고 있어 비행 컴퓨터로써 적합하다고 생각하였다.

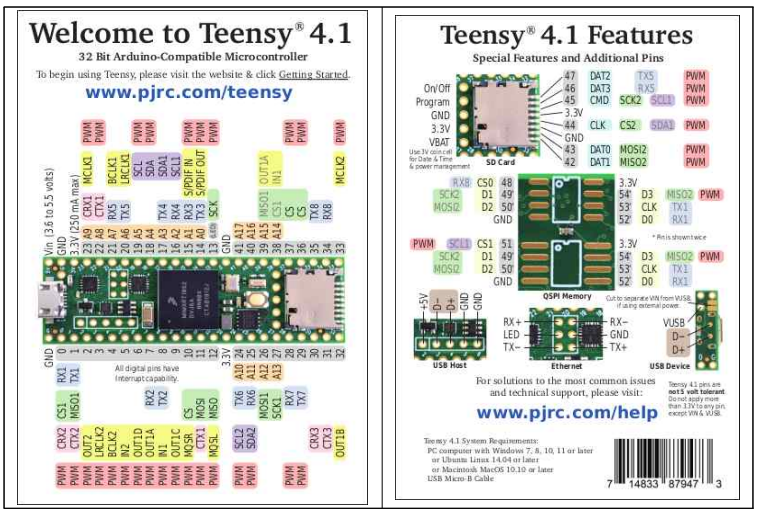


Fig. 14. Teensy 4.1 MCU

자세 측정을 위한 IMU로는 MPU-9250을 채택하였다. MPU-9250은 3축 자이로스코프, 3축 가속도계, 3축 지자기를 측정 할 수 있는 보편적인 9축 센서이다. 또한 절대적인 위치를 알아내기 위한 BK-252Q GNSS 모듈과 VL53L0X TOF(Time of flight)센서를 추가하여 각각 절대 위치와 고도를 알아낼 수 있도록 하였다 [5].

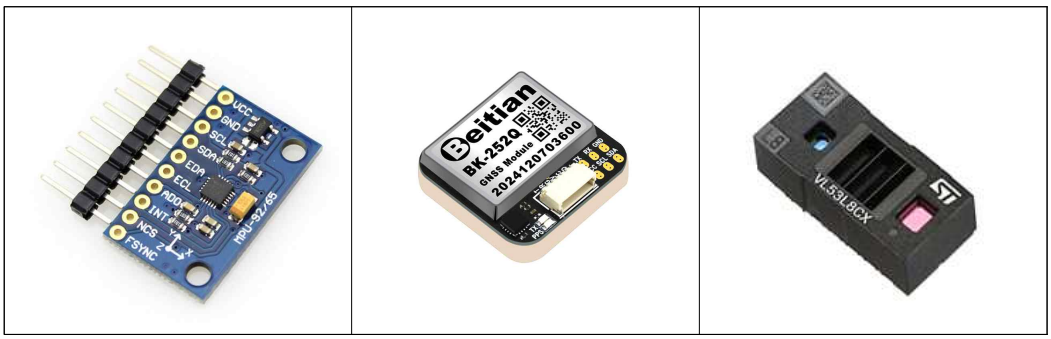


Fig. 15. MPU9250, BK-252Q, VL53L0X 센서

Teensy와 MPU9250은 SPI 프로토콜을 통해 통신하며, BK-252Q와는 UART, VL53L0X와는 I2c 프로토콜로 통신한다. BK-252Q와 VL53L0X는 5V 이상의 전압을 인가해야 구동될 수 있다. 반면 Teensy의 경우 로직레벨 전압이 3.3V이기 때문에 5V 로직레벨을 입력받을 경우 손상을 입을 수 있다. 따라서 각 센서의 출력을 오실로스코프를 통해 확인한 결과 MPU 9250을 제외한 나머지 센서들은 5V를 인가하여도 로직레벨은 3.3V로 출력됨을 확인하였다. 하지만 MPU 9250은 입력전압과 동일한 전압으로 신호를 송신하기 때문에 입력 전압을 3.3V로 구동하였다.

Fig. 16. 오실로스코프를 이용한 로직 전압 확인 BK-252Q(좌) ESC(우)

전기로 구동되는 덕트팬의 경우 BLDC 모터를 가지고 있어, 배터리의 직류 전원을 3상 교류로 변환해야 한다. 취미용 RC 항공기에 사용되는 상용 ESC를 사용하였다. 메인팬의 경우 한 개의 120A, 측 추력기는 각각 한 개씩 30A ESC를 사용하였다. ESC는 PWM(Pulse Width Modulation) 신호를 입력받는다. 50Hz를 주기로 펄스의 폭이 1ms일 때 0%, 2ms 일 때를 100%로 하여 구동된다. 일반적으로 5V의 로직레벨로 구동되기 때문에 3.3V 전압의 신호에서도 구동이 가능한지 확인하였고, 2.5V 이상의 로직 전압에는 구동이 가능함을 확인하였다.

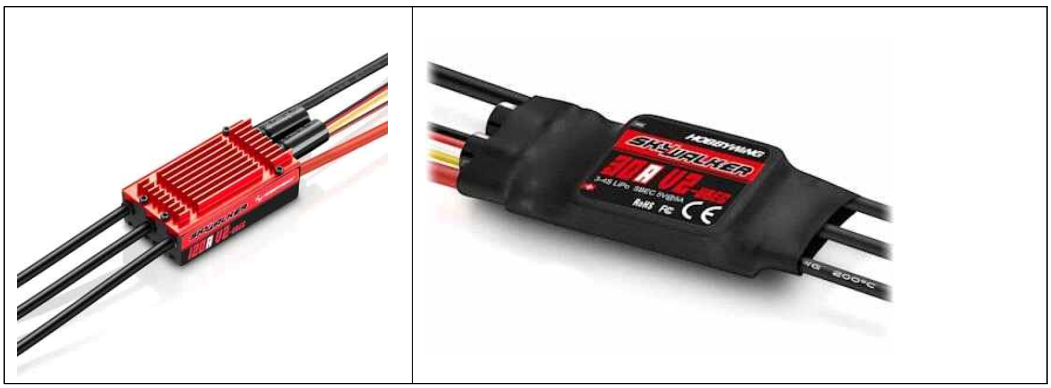


Fig. 17. Hobbywing Skywalker ESC V2 120A(좌) 30A(우)

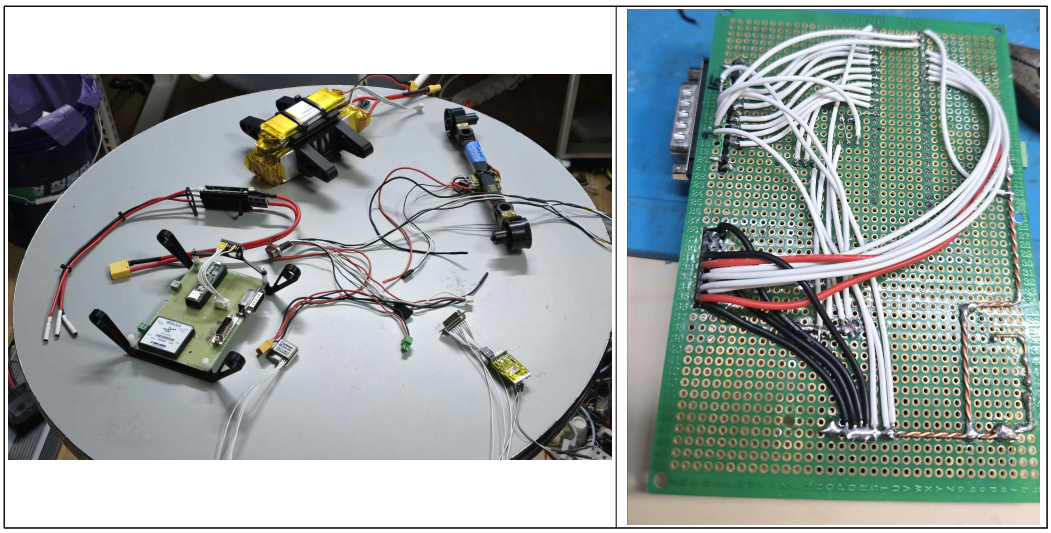


Fig. 18. VTVL의 전자 제어 시스템

3.4 배터리 설계 및 전원 공급 체계

본 연구에서는 전자 제어 계통과 추진 계통 간의 전기적 간섭을 최소화하고 안정적인 전원 공급을 확보하기 위해, 배터리 시스템을 분리하여 구성하였다. 비행 컴퓨터, IMU, GPS, 고도 센서 및 롤 제어기를 포함한 모든 전자 장치는 약 1600 mAh 리튬폴리머 배터리를 3셀로 하여 정격 11.1V 배터리를 구성하였으며, 메인 덕트팬은 고출력이 요구되므로 60C 2200 mAh 배터리를 8개 직렬, 2개 병렬로 연결하여 제작하였으나 너무 높은 무게로 인해 한 개씩 8개의 직렬 배터리를 구성하였다. 전류미터를 통해 30V에서 70A 수준을 소비하는 것을 확인하였기에 8S1P로 구성하는 것이 무리가 없을 것으로 생각되었다.

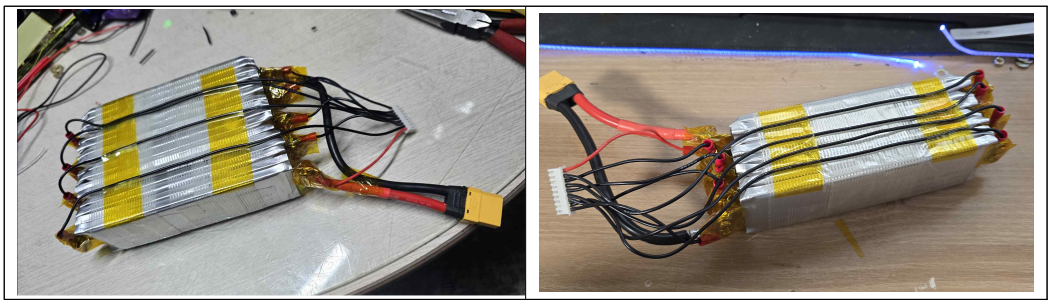


Fig. 19. 8S2P 배터리(좌) 이후 변경된 8S1P 배터리(우)

전장 구동용 3셀 배터리의 경우 롤 제어기에는 별도의 전환 없이 바로 연결되며, 전압 강하용 DC-DC 컨버터를 이용해 teensy와 센서류에 공급된다. MPU 9250의 3.3V의 경우 Teensy에 내장된 컨버터를 통해 전원을 공급하였다.

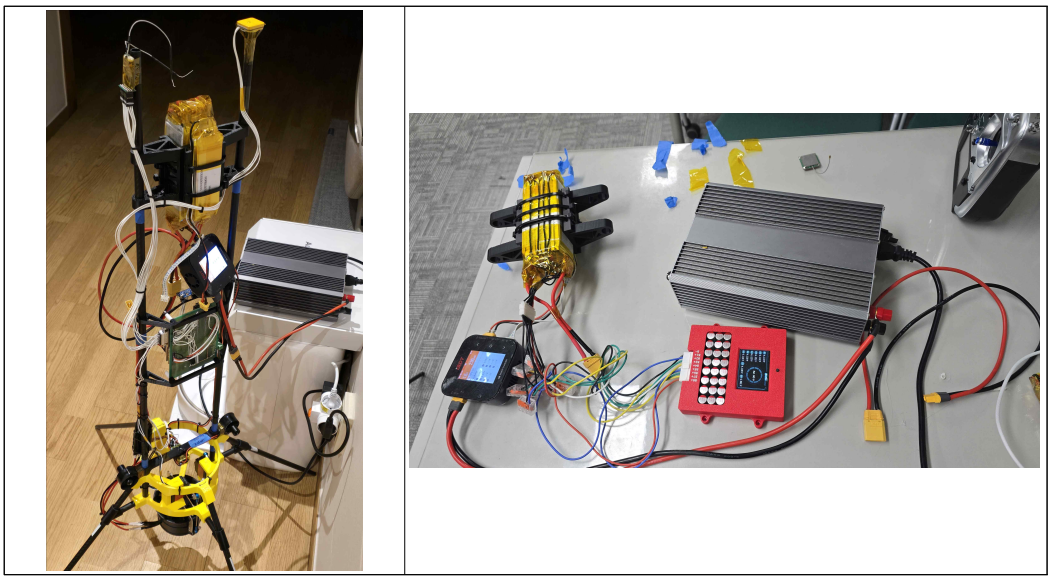


Fig. 20. 배터리팩의 충전

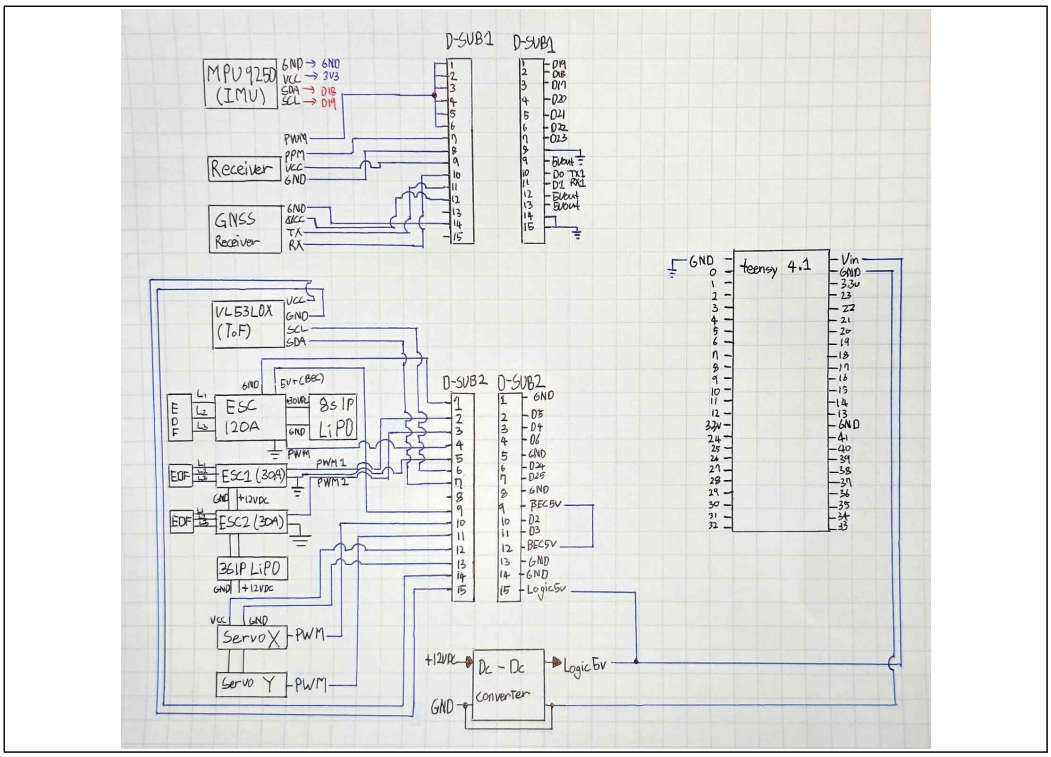


Fig. 21. 전자제어 시스템의 회로도

Ⅳ. GNC 알고리즘 모델링

멀티콥터와 달리 TVC를 사용하는 VTVL과 같은 로켓형 기체들은 제어가 독립적이지 않기 때문에 외란에 취약하다. [4] 따라서 기체의 비행 안정성과 비행제어 컴퓨터의 계산 안정성을 위해서 하나의 입력을 여러 요소에 대해 독립적으로 산출한 뒤, 다시 융합하여 하나의 출력으로 생성해야 한다.

입력으로 IMU의 3축 각속도, 3축 가속도, 3축 지자기와 GNSS의 시간, 위도, 경도, 고도 데이터, ToF의 고도 데이터가 있다. VTVL이 도심에서 시험 되며, 기체에 고전류 배선들이 자기장을 생성하기 때문에 지자계의 데이터를 신뢰하기 어려울 것으로 판단하여 지자계 센서를 사용하지 않기로 하였다. 또한 단

시간 저고도에서 시험을 진행할 예정이기에 GNSS 센서의 표준 시간은 Teensy 4.1의 RTC(Real Time Clock)으로, 고도 데이터는 Tof 센서값으로 대체하였다. 본 연구에서는 이러한 입력을 받아 항법 데이터를 추정하고, 조종기에서 보낸 수신기의 신호와 융합하여 에러를 산출한다. 이 에러는 자세 제어기(피치, 요), 롤 제어기, 위치 제어기, 고도 제어기를 통해 제어 신호를 생성하고 이를 출력 신호로 변환하여 메인 덕트팬 스로틀, 롤제어기 스로틀, 두 개의 TVC 액추에이터에 신호를 보냄으로써 VTVL을 제어한다. [5]

4.1 항법 알고리즘의 설계

항법은 센서의 융합을 통해 자세와 위치를 추정하는 부분이다. 자세추정(AHRS, Attitude and Heading Reference System)을 위해서는 Madwick 필터를 사용하며, 위치추정에는 IMU의 값과 절대 위치를 융합하는 단순 상보 필터를 이용할 것이다.

4.1.1 AHRS 알고리즘

Madgwick AHRS filter는 각속도를 누적하여 구한 자세정보를 vector observation을 통해 구한 자세정보로 보정하는 fusion 알고리즘이다. 각속도를 단순히 수치적분 하여 자세를 추정한다면, 센서의 노이즈, 잘못된 값을 읽는 경우, 수치적분의 오차등을 이유로 정확하지 않으며 Drift가 발생한다. 이는 단시간에는 빠르고 정확하게 각도 정보를 얻을 수 있다는 장점이 있으나 장시간에 걸쳐 누적되는 오차에 취약하다. 반면 특정 벡터 두 개의 상관관계를 통해 사이의 각도를 구할 수 있는데, 이는 장시간 Drift의 발생이 적다. 하지만

단시간에서는 노이즈 등의 이유로 신뢰성이 떨어지기에 이들을 융합하여 상호 보완적으로 보다 정확한 데이터를 구할 수 있다 [6].

제어에서는 먼저 센서를 지상에 정지시킨 후 캘리브레이션을 진행하여 지면과의 평행 상태를 유지할 수 있도록 한다. 또한 센서 값의 노이즈가 매우 크기 때문에 저역필터를 통해 비이상 데이터를 차단한 후 madwick을 적용한다 [2]. 쿼터니언으로 출력된 AHRS 값은 추후 자세제어에 이용된다. 또한 위치 제어에서 동체 좌표계를 ENU 좌표계로의 변환에 사용된다. 위치제어는 ENU를 기준으로 제어해야 하지만, 가속도와 GPS등의 센서는 동체에 장착되어 있기에 AHRS 데이터를 통해 좌표를 변환한 후 위치 추정에 사용한다.

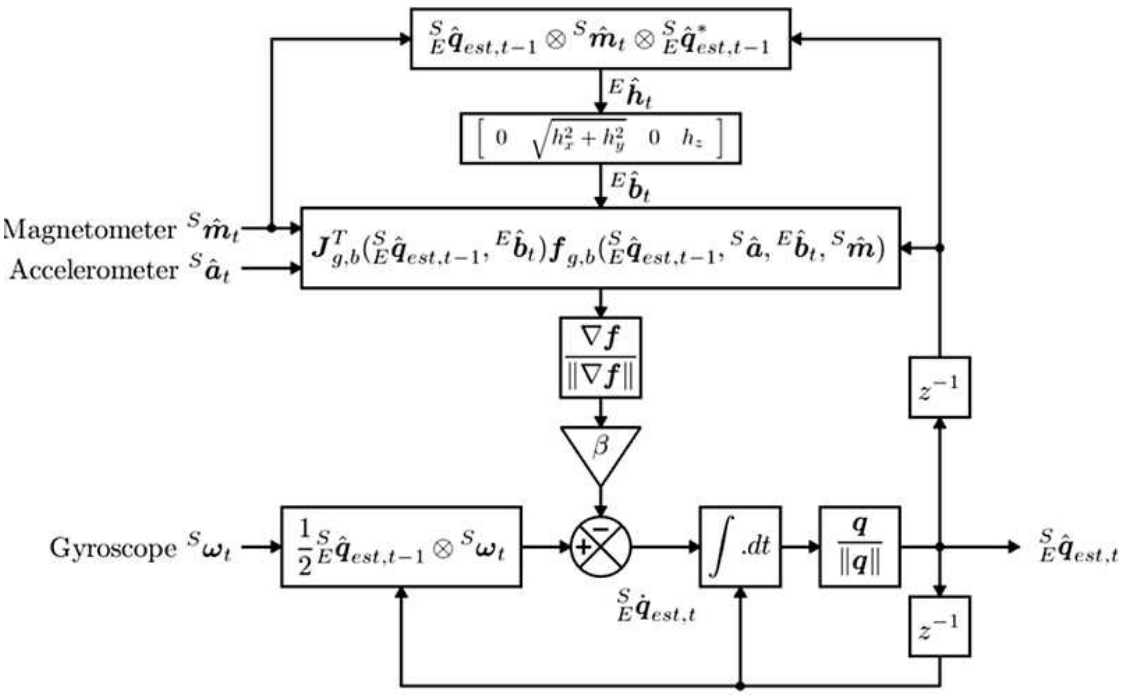


Fig. 22. Madwick filter 블록 다이어그램 [6]

4.1.2 위치추정 알고리즘

위치 추정에서도 AHRS 알고리즘과 같이 센서의 캘리브레이션, 저역 필터가 사용된다. 이후 Body frame에서 계측된 센서값들을 AHRS 데이터를 통해 ENU 좌표계로 변환하여 사용된다 [3]. 위치추정에는 상보필터를 사용한다. 상보필터란 두 개의 보완적인 값을 상보적으로 융합하여 더 정확한 값을 얻는 필터이다. 일반적으로 하나의 고역 데이터와 하나의 저역 데이터를 사용하며, IMU의 자이로 데이터와 GNSS의 위치 데이터와 같은 상황에 적합하다 [7]. Fig. 23. 와 같이 표현되며, 적정한 α의 값을 조정하여 사용한다.

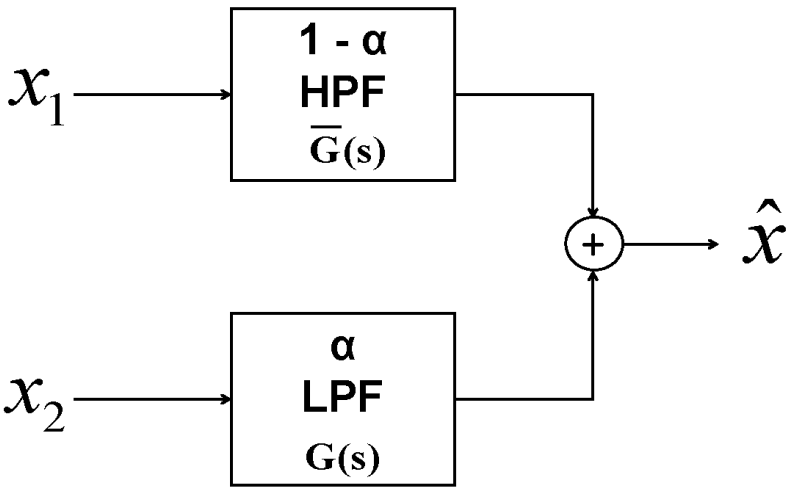


Fig. 23. 상보필터(Complimentary Filter) [7]

수평 위치는 두 번 적분한 IMU의 가속도 데이터와 GNSS의 위치를 융합한다. 추후 속도기반 제어기를 활용하기 위해 IMU 가속도를 수치적분한 값과, 위치 제어에 사용되는 값의 변화율을 상보필터로 융합하여 속도를 추정한다. 최종적으로 고도는 ToF 센서의 값과 IMU의 수직 위치를 융합한 값으로 추정한다.

최종 항법 알고리즘은 Fig. 24. 과 같다. 이를 비행제어 컴퓨터에 업로드하여 시험하여 구동 여부를 확인하였으며 육안의 Drift는 확인되지 않았다.

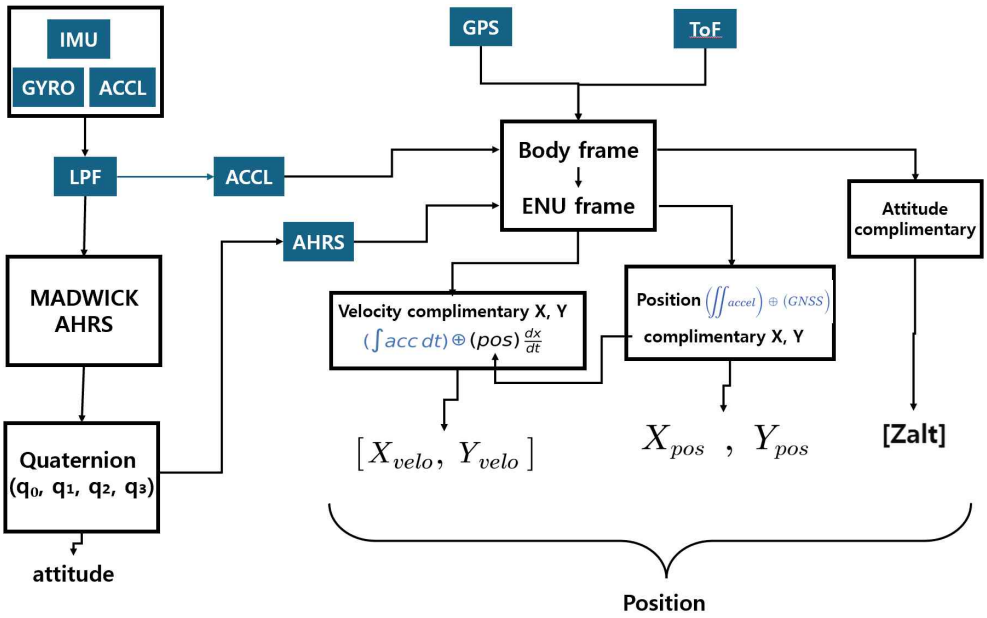


Fig. 24. 항법 알고리즘 블록다이어 그램

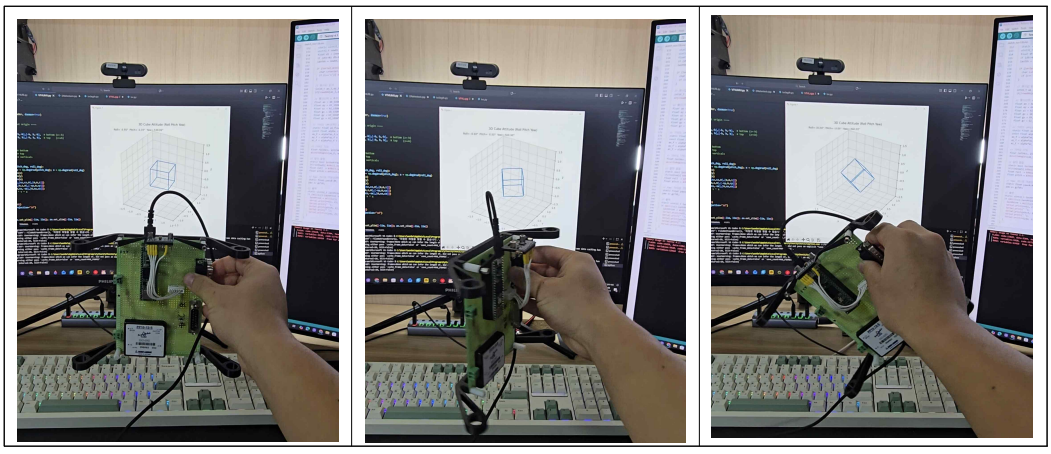


Fig. 25. AHRS 알고리즘 시험

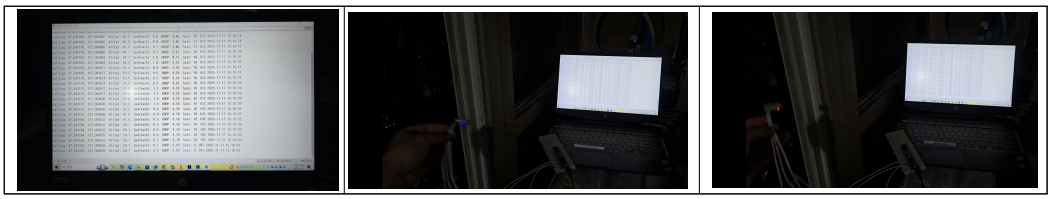


Fig. 26. GNSS 및 위치 추정 알고리즘 시험

4.2 제어 알고리즘의 설계

앞 절에서 센서 융합을 통해 추정된 기체의 자세 및 위치 정보는 본 절에서 설계한 제어 알고리즘의 입력으로 사용된다. 제어(Control)는 현재 상태(state)를 목표 상태(desired state)에 수렴시키기 위한 과정으로, 외란이 지속적으로 작용하는 실제 비행 환경에서 기체의 안정성을 유지시킨다.

본 연구에서는 구조적 단순성과 실시간 구현 가능성을 고려하여 PID(Proportional-Integral-Derivative) 제어기를 기반으로 제어 알고리즘을 설계하였다. PID 제어기는 제어 대상의 정확한 수학적 모델이 없더라도 안정적인 제어가 가능하며, 항공기 및 로켓을 포함한 다양한 공학 시스템에서 널리 사용되고 있다.

4.2.1 PID 제어기

PID 제어기는 목푯값(reference)과 현재 측정값 사이의 오차(error)를 기반으로 제어 출력을 계산한다. 제어 출력 $u(t)$는 다음과 같이 정의된다.

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}$$

PID는 비례항, 적분항, 미분항으로 이루어져 있다. 비례항은 현재 오차의 크

기에 비례한 제어 출력을 생성하여 빠른 응답을 생성하는 항으로, 목푯값과 선형적인 관계를 가진 항이다. 비례항의 K_p는 비례 이득(gain)으로, 이 상수를 조절하여 제어기를 튜닝할 수 있다. 제어성이 실제 원하는 값 대비 부족할 경우 K_p gain을 키우면 된다. 적분항은 시간에 따른 오차의 누적을 보상하여 정상 상태 오차(steady-state error)를 제거하는 항이다. 예를 들어, 착륙 후 정지된 기체의 TVC에서 다양한 외란에 의해 짐벌의 각도가 수직이 아닐 수 있다. 이때 적분항은 짐벌 각도를 계속 적분하여 최종 목푯값과 현재 값의 차이를 추적하고 0에 수렴하도록 상쇄시켜준다. 그 크기는 K_i 이득을 따른다. 미분항은 오차의 변화율을 통해 과도 응답에서의 진동과 오버슈트를 억제한다. 한 시스템에서 비례 이득의 값이 클 경우 목푯값 대비 더 많이 제어되는 경우가 발생한다. PI 제어기에서 적분항은 이를 억제하기 위해 그 반대 값만큼의 제어 입력을 생성하고, 이는 계속 반복되어 진동이나 오버슈트를 발생시킨다. 오버슈트는 TVC와 같은 환경에서 기체의 구조적 불안정성과 추력 손실을 야기하기 때문에 이를 상쇄하기 위해 K_d(미분 이득) 만큼의 변화율 내로 제어가 이루어질 수 있도록 하고, 그 역할이 미분항이다.

4.2.2 각도 및 위치 기반 제어 (Angle / Position Control)

각도와 위치는 절대적인 상태라는 공통점을 가지고 있다. 또한 이들은 일반적으로 목표값으로써 사용된다. 예를 들어, 기체를 특정 각도로 유지한다거나, 특정 고도를 유지하라는 것이 목표 사항이라면, 이때의 제어 오차는 다음과 같다.

$$e(t) = x_{desired} - x_{current\,state}$$

여기서 x는 각도나 위치의 상태이다. 각도/위치 제어 PID 제어기는 이 오차값을 목표로 수렴하도록 제어 출력을 생성한다. 각도 및 위치 제어는 직관적이

며 사용자가 이해하기 쉽다는 장점이 있으나, 외란이 큰 환경이나 응답 속도가 중요한 상황에서는 오버슈트 및 진동이 발생하기 쉽다. 특히 VTVL과 같이 추력 기반으로 자세를 유지하는 시스템에서는 느린 응답 특성이 문제가 될 수 있다.

4.2.3 변화율 기반 제어 (Rate Control)

각도나 속도의 변화율을 제어함으로써 각도나 속도 제어의 단점을 극복할 수 있다. 예를 들어 기체의 한 축을 0도로 유지하는 것이 아니라 각속도를 0으로 유지하면 기체의 운동을 멈추게 할 수 있다. 이때의 제어 오차는 다음과 같다.

$$e(t) = \dot{x}_{desired} - \dot{x}_{currentstate}$$

변화율 기반의 제어는 외란에 대한 반응이 빠르고, 감쇠능력이 안정적인 장점이 있다. 또한 IMU등 다양한 센서에서 절댓값보다는 상대 속도(가속도, 각속도)를 제공하는 경우가 많아 실제로 적용하기 유용하다는 장점도 있다. 따라서 본 연구에서는 변화율을 기반으로 기체를 제어한다.

4.2.4 이중 루프 제어 구조 (Dual-Loop Control)

변화율을 통해 제어할 경우 일반적으로 요구되는 목표와 차원이 다르기 때문에 수치적으로 현재값이 목푯값까지 수렴하도록 하기 위해서 각도 및 위치 기반 제어가 수렴의 정도를 파악하고, 변화율을 제어해야 한다. 이를 위해서 절댓값 기반 PID 루프의 미분항에 변화율 PID 루프를 넣어 미분항을 다른 PID 루프로 제어하는 이중루프 PID를 구성할 수 있다.

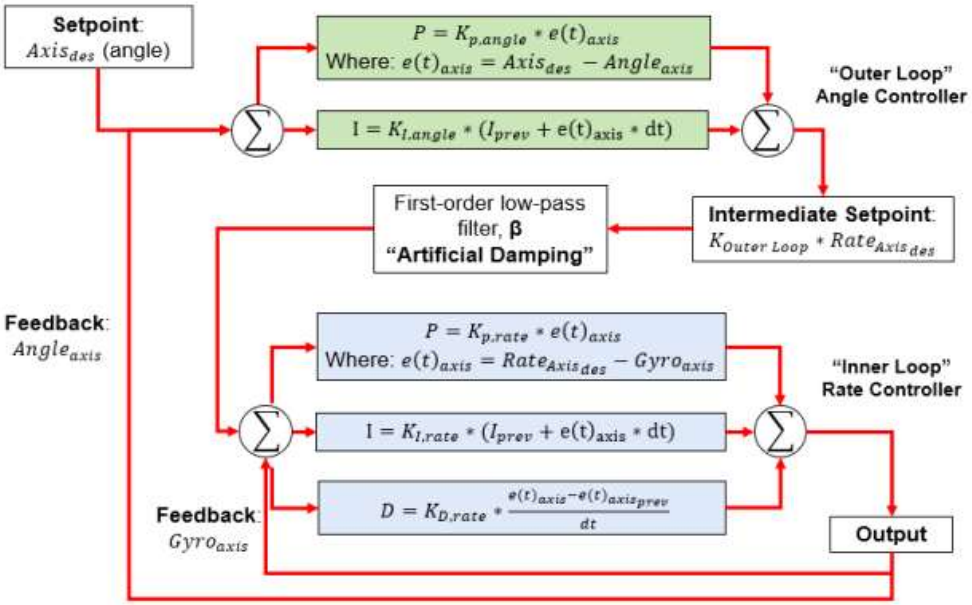


Fig. 27. 각도, 각속도 이중루프 PID 제어기의 구조 [4]

이중 루프 PID 제어기에서 외부 루프(각도)는 비교적 느린 응답 특성으로 각도의 상태를 제어하며, 내부 루프(각속도)는 빠른 응답 속도로 외란에 직접적으로 대응하는 특성을 가진다. 본 연구에서는 이중 루프 구조를 통해 각도와 위치 제어기를 구성하였으며, 이러한 구조를 통해 전반적인 제어의 안정성을 향상시키고, 비교적 간단하게 튜닝을 할 수 있었다.

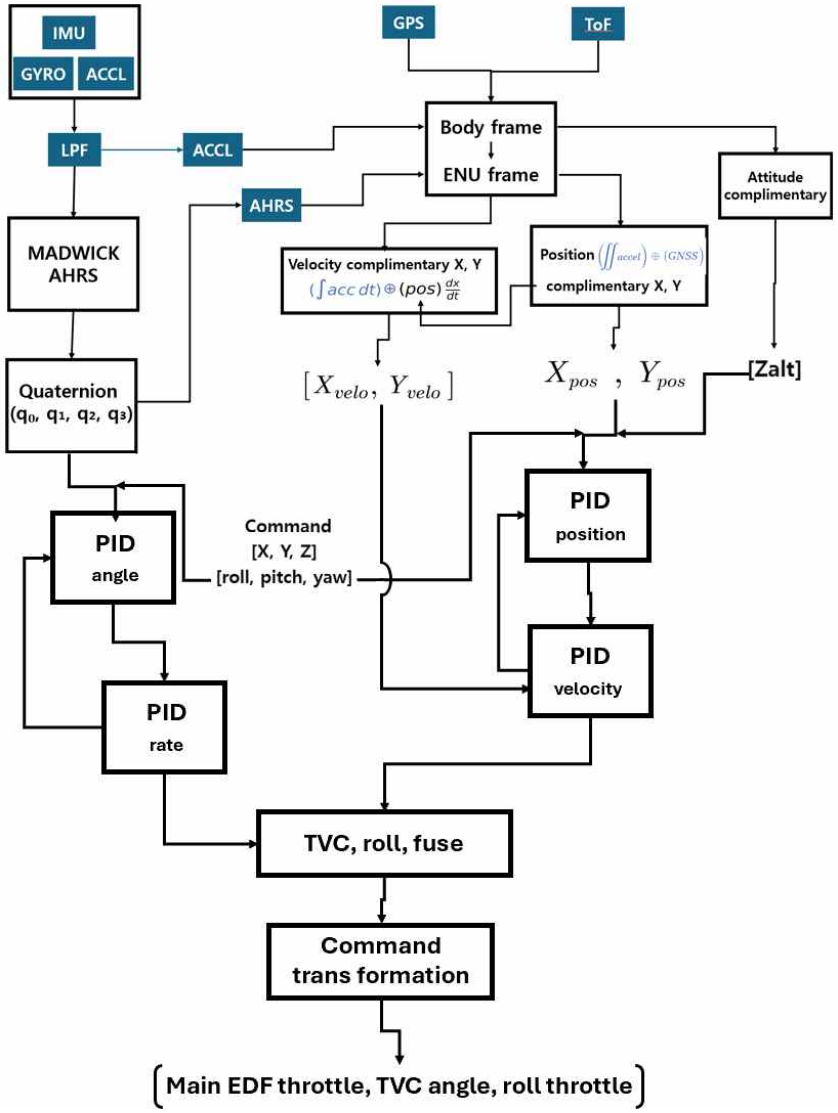


Fig. 28. 제어 루프를 포함한 전체 GNC 알고리즘의 블록 다이어그램

이를 바탕으로 최종 비행제어 알고리즘을 C++에서 작성하였으며, Arduino IDE를 통해 Teensy로 업로드 하였다. 최종 코드는 부록 2에 게시되어 있으며, 같은 계열로 GNC 시스템을 구성할 경우 일부 변수 조건만 변경한 뒤 사용할 수 있다.

Ⅴ. 비행시험

본 연구의 초기 비행시험은 완전한 자유 비행을 목표로 하지 않고, 지상 약 1 m 이내의 근거리 호버링 구간에서 자세 추정(AHRS)과 이중 루프 PID 기반 자세 제어가 실제 추력, 진동, 기구 오차 등 외란 환경에서도 발산하지 않고 동작하는지에 초점을 두었다. 이를 위해 지상 정적 시험으로 PWM/구동기 및 센서 초기화, 수렴을 확인한 뒤 계류시험에서 내부(각속도) 루프의 감쇠 성능과 외부(각도) 루프의 수렴성을 단계적으로 튜닝하고, 최종적으로 제한된 호버링 조건에서 TVC 및 보조 구동기 출력이 자세 오차를 감소시키는 방향으로 작용함을 검증하였다. 다만 시험 환경이 제한되어 장시간 안정성 및 위치, 고도 유지와 같은 완전한 GNC 성능은 확인하지 못했으며, 향후에는 자유 비행 단계로 확장하여 체공 시간 증가, ToF/GNSS 기반 외부 루프(고도・위치) 제어 검증, TVC 기구학 보정 및 포화/안티윈드업 강화, 로그 기반 정량 평가와 PID 튜닝 자동화를 통해 시스템을 고도화할 계획이다.

Fig. 29. VTVL 호버링 시험

Ⅵ. 결론

본 연구에서는 전기 덕트팬을 기반으로 한 소형 VTVL 실증기를 제작하고, 해당 기체에 적용 가능한 GNC 시스템을 이론-시뮬레이션-하드웨어-비행시험의 단계적 절차를 통해 설계 및 검증하였다. 기존 멀티콥터 기반 실험 플랫폼과 달리, 원통형 로켓 구조와 TVC를 적용함으로써 실제 재사용 발사체의 재착륙 환경과 유사한 동역학적 특성을 모사하고자 하였다.

항법 알고리즘 측면에서는 쿼터니언 기반 자세 표현을 중심으로 IMU 고속 추정과 GPS, 고도계 저속 보정을 결합한 계층적 상태 추정 구조를 구현하였다. 이를 통해 수직 이착륙 및 저속 착륙 구간에서 요구되는 수치적 안정성과 계산 효율성을 동시에 확보할 수 있었다. 제어 측면에서는 TVC를 이용한 피치, 요 축 제어와 보조 EDF를 이용한 롤 축 제어를 분리 설계함으로써, 제한된 제어 입력을 가진 VTVL 시스템에서도 3축 자세 제어가 가능함을 보였다.

하드웨어적으로는 회전형 서보모터와 4절 링크 기구를 활용한 TVC 짐벌을 설계하고, 구조 해석을 통해 실험 환경에서 충분한 강성과 안정성을 확보하였다. 비행시험은 반복 실험을 통해 제어기 파라미터를 조정함으로써 안정적인 자세 유지가 가능함을 확인하였다. 본 연구에서는 완전 자율 착륙을 위한 유도(Guidance) 알고리즘과 고도 제어의 정밀도 측면에서 한계가 존재하지만, 학생 수준의 장비와 공개 가능한 알고리즘만으로도 VTVL 재착륙 제어의 핵심 개념을 실험적으로 검증할 수 있음을 보여준다.

향후 연구에서는 공력 항력 모델을 포함한 보다 정교한 동역학 모델링, 칼만 필터 기반의 센서 퓨전 고도화, 그리고 완전 자율 유도 알고리즘의 도입을 통해 실제 재사용 발사체 재착륙에 더욱 근접한 실험이 가능할 것으로 기대된다. 본 연구에서 제안한 플랫폼과 설계 절차는 실험실 규모 재사용 발사체 GNC 연구의 출발점으로써 활용 가치가 있을 것이다.

참 고 문 헌

[1] Design of Thrust Vectoring attitude control system for Lunar Lander flying testbed(David Bernacchia, 2019)

[2] 회전익 무인비행체를 위한 가속도계, 지자기센서, GPS 통합 항법시스템 구현 및 정지비행 제어기 설계에 관한 연구(2003, 박종원)

[3] INS/GPS/ 속도계 결합 항법시스템의 구성 및 성능분석(2001, 박영범)

[4] dRehmFlight VTOL(Nicholas Rehm)

[5] Time-delay flight control of small-scale rocket demonstrator using a mini gas turbine engine based on PX4 Autopilot(Seungwoo Park, 2023)

[6] Sebastian O.H. Madgwick, An efficient orientation filter for inertial and inertial/magnetic sensor arrays, 2010

[7] https://velog.io/@soup1997/Complimentary-Filter, 2023

<부록 1> 4절 링크 옵티마이저 코드

```
import numpy as np
import math
from scipy.optimize import minimize
import matplotlib.pyplot as plt

# ============================
# Fixed geometry
# ============================
A = np.array([-40.0,  50.0])   # crank pivot
D = np.array([  0.0, -20.0])   # rocker pivot

R1 = 24.0                      # crank length [mm]
L4 = 100.0                     # rocker length [mm]

CRANK_SPAN_DEG = 60.0          # crank: center ±60 deg ⇒ total 120 deg
TARGET_CENTER_DEG = -127.9     # desired rocker center angle (given)
TARGET_SWING_DEG = 20.0        # desired ±20 deg swing

DEG2RAD = math.pi / 180.0
RAD2DEG = 180.0 / math.pi


def wrap_pi(a):
    """Wrap angle [rad] into [-pi, pi]."""
    return (a + math.pi) % (2 * math.pi) - math.pi


# ============================================
# 4-bar kinematics: crank→rocker solutions
# ============================================
def rocker_solutions(L2, phi):
    """
    For given coupler length L2 and crank angle phi [rad],
    return the two possible rocker angles [rad] (assembly branches).
    """
    # crank tip B
    B = A + R1 * np.array([math.cos(phi), math.sin(phi)])

    # D→B vector
    Ex, Ey = (B - D)
    R = math.hypot(Ex, Ey)

    num = Ex**2 + Ey**2 + L4**2 - L2**2
    den = 2.0 * L4 * R
    if den == 0.0:
        return None

    arg = num / den
    if abs(arg) > 1.0 + 1e-6:
        return None
    arg = max(-1.0, min(1.0, arg))

    alpha = math.atan2(Ey, Ex)
    delta = math.acos(arg)
```

```
    th1 = wrap_pi(alpha + delta)
    th2 = wrap_pi(alpha - delta)
    return [th1, th2]


def rocker_extremes(L2, phi0):
    """
    For given L2, crank center ϕ0, compute rocker extreme angles
    at ϕ0±60°, then derive rocker center and swing.

    Returns:
      success: bool
      theta_center_deg: rocker center angle [deg]
      up_mag_deg:       |upper deviation| [deg]
      down_mag_deg:     |lower deviation| [deg]
    """
    phi1 = phi0 + CRANK_SPAN_DEG * DEG2RAD
    phi2 = phi0 - CRANK_SPAN_DEG * DEG2RAD

    sols1 = rocker_solutions(L2, phi1)
    sols2 = rocker_solutions(L2, phi2)
    if sols1 is None or sols2 is None:
        return False, None, None, None

    best = None

    # choose consistent assembly branch: (+,+) or (-,-)
    for sign in (+1, -1):
        th1 = sols1[0] if sign == +1 else sols1[1]
        th2 = sols2[0] if sign == +1 else sols2[1]

        # center angle via vector average
        v1 = np.array([math.cos(th1), math.sin(th1)])
        v2 = np.array([math.cos(th2), math.sin(th2)])
        v_sum = v1 + v2
        if np.allclose(v_sum, 0.0):
            continue

        th_center = wrap_pi(math.atan2(v_sum[1], v_sum[0]))

        # deviations from center
        d1 = wrap_pi(th1 - th_center)
        d2 = wrap_pi(th2 - th_center)

        # magnitude of "up" and "down" deviations
        up_mag = max(abs(d1), abs(d2))
        down_mag = min(abs(d1), abs(d2))

        th_center_deg = th_center * RAD2DEG

        # choose branch whose center is closer to target
        center_err = (th_center_deg - TARGET_CENTER_DEG) ** 2

        if best is None or center_err < best[0]:
            best = (center_err, th_center_deg, up_mag * RAD2DEG, down_mag * RAD2DEG)

    if best is None:
```

```
        return False, None, None, None

    _, thc_deg, up_deg, down_deg = best
    return True, thc_deg, up_deg, down_deg


# ============================================
# Objective for optimizer: vars = [L2, phi0]
# ============================================
def objective(vars):
    L2, phi0 = vars

    # basic constraints
    if L2 <= 0.0 or L2 > 1000.0:
        return 1e9

    ok, center_deg, up_deg, down_deg = rocker_extremes(L2, phi0)
    if not ok:
        return 1e9

    # 1) rocker center angle close to target
    err_center = (center_deg - TARGET_CENTER_DEG) ** 2

    # 2) swing magnitudes close to TARGET_SWING_DEG
    err_swing = (abs(up_deg) - TARGET_SWING_DEG) ** 2 ₩
              + (abs(down_deg) - TARGET_SWING_DEG) ** 2

    return err_center +  err_swing


# ============================
# Optimization
# ============================
if __name__ == "__main__":
    # Initial guess: L2 ~ 140 mm, phi0 ~ -120 deg
    x0 = np.array([140.0, -120.0 * DEG2RAD])

    res = minimize(
        objective,
        x0,
        method="Nelder-Mead",
        options={"xatol": 1e-8, "fatol": 1e-8, "maxiter": 20000},
    )

    L2_opt, phi0_opt = res.x
    ok, center_deg, up_deg, down_deg = rocker_extremes(L2_opt, phi0_opt)

    print("=== Optimization result (L4 = 100 mm, theta_center = -127.9 deg) ===")
    print(f"success             : {res.success}")
    print(f"L2 (coupler)        : {L2_opt:.6f} mm")
    print(f"phi0 (deg)          : {phi0_opt * RAD2DEG:.6f}")
    if ok:
        print(f"rocker center     : {center_deg:.6f} deg (target = {TARGET_CENTER_DEG})")
        print(f"rocker up dev     : +{up_deg:.6f} deg (target ≈ +{TARGET_SWING_DEG})")
        print(f"rocker down dev   : -{down_deg:.6f} deg (target ≈ -{TARGET_SWING_DEG})")
    else:
        print("Kinematics invalid at optimum (unexpected).")
```

```
# ============================
# Visualization
# ============================
# 1) rocker vs crank angle
phi_list = np.linspace(
    phi0_opt - CRANK_SPAN_DEG * DEG2RAD,
    phi0_opt + CRANK_SPAN_DEG * DEG2RAD,
    181,
)

crank_deg = []
rocker_deg = []
theta_prev = None

for phi in phi_list:
    sols = rocker_solutions(L2_opt, phi)
    if sols is None:
        crank_deg.append(phi * RAD2DEG)
        rocker_deg.append(np.nan)
        continue

    # pick branch that keeps motion continuous
    if theta_prev is None:
        theta = min(sols, key=lambda t: abs(t * RAD2DEG - TARGET_CENTER_DEG))
    else:
        theta = min(sols, key=lambda t: abs(wrap_pi(t - theta_prev)))
    theta_prev = theta

    crank_deg.append(phi * RAD2DEG)
    rocker_deg.append(theta * RAD2DEG)

# 2) mechanism snapshot at three crank angles (phi0-60, phi0, phi0+60)
def point_B(phi):
    return A + R1 * np.array([math.cos(phi), math.sin(phi)])

def point_C(phi, theta_guess=None):
    sols = rocker_solutions(L2_opt, phi)
    if sols is None:
        return np.array([np.nan, np.nan]), None
    if theta_guess is None:
        theta = min(sols, key=lambda t: abs(t * RAD2DEG - TARGET_CENTER_DEG))
    else:
        theta = min(sols, key=lambda t: abs(wrap_pi(t - theta_guess)))
    C = D + L4 * np.array([math.cos(theta), math.sin(theta)])
    return C, theta

phi_minus = phi0_opt - CRANK_SPAN_DEG * DEG2RAD
phi_zero  = phi0_opt
phi_plus  = phi0_opt + CRANK_SPAN_DEG * DEG2RAD

Bm = point_B(phi_minus)
B0 = point_B(phi_zero)
Bp = point_B(phi_plus)

C0, th0 = point_C(phi_zero)
Cm, _   = point_C(phi_minus, th0)
```

```
Cp, _    = point_C(phi_plus, th0)

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# --- Mechanism view ---
ax = axes[0]
# pose at phi0-60
ax.plot([A[0], Bm[0]], [A[1], Bm[1]], "C0-")
ax.plot([Bm[0], Cm[0]], [Bm[1], Cm[1]], "C0-")
ax.plot([D[0], Cm[0]], [D[1], Cm[1]], "C0-", label="phi = phi0 - 60 deg")

# pose at phi0
ax.plot([A[0], B0[0]], [A[1], B0[1]], "C1-")
ax.plot([B0[0], C0[0]], [B0[1], C0[1]], "C1-")
ax.plot([D[0], C0[0]], [D[1], C0[1]], "C1-", label="phi = phi0")

# pose at phi0+60
ax.plot([A[0], Bp[0]], [A[1], Bp[1]], "C2-")
ax.plot([Bp[0], Cp[0]], [Bp[1], Cp[1]], "C2-")
ax.plot([D[0], Cp[0]], [D[1], Cp[1]], "C2-", label="phi = phi0 + 60 deg")

ax.scatter(
    [A[0], D[0], Bm[0], B0[0], Bp[0], Cm[0], C0[0], Cp[0]],
    [A[1], D[1], Bm[1], B0[1], Bp[1], Cm[1], C0[1], Cp[1]],
    s=20
)

ax.set_aspect("equal", "box")
ax.set_title(f"Four-bar mechanism (L2 ≈ {L2_opt:.2f} mm, L4 = 100 mm)")
ax.set_xlabel("X [mm]")
ax.set_ylabel("Y [mm]")
ax.grid(True)
ax.legend()

# --- Rocker vs crank ---
ax2 = axes[1]
ax2.plot(crank_deg, rocker_deg, "-")
ax2.axhline(TARGET_CENTER_DEG, color="r", linestyle="--", label="target center")
ax2.axhline(TARGET_CENTER_DEG + TARGET_SWING_DEG, color="g", linestyle="--", label="target ±20 deg")
ax2.axhline(TARGET_CENTER_DEG - TARGET_SWING_DEG, color="g", linestyle="--")
ax2.set_xlabel("Crank angle [deg]")
ax2.set_ylabel("Rocker angle [deg]")
ax2.set_title("Rocker angle vs crank angle")
ax2.grid(True)
ax2.legend()

plt.tight_layout()
plt.show()
```

<부록 2> VTVL GNC 코드

```
GNC_Teensy41_VTVL.ino
- Teensy 4.1 based VTVL (EDF + TVC + roll EDF) flight computer
- Navigation:
    * High-rate AHRS (Madgwick quaternion) from IMU
    * Low-rate position/altitude from GNSS (BK-252Q) + ToF (VL53L0X)
- Dual-loop attitude control:
    * Outer loop: angle -> desired rate
    * Inner loop: rate -> TVC deflection (pitch/yaw) and roll EDF command
- Designed to match the paper's stated architecture:
    state x = [p_E, v_E, q_E^B, omega_B] (ENU + Body) and hierarchical loops.

IINTEGRATION NOTES
1) Axis convention:
   - Paper uses ENU, body x_B points "up" when vehicle vertical.
   - IMU axes on your MPU9250 breakout may differ -> you MUST map gyro/accel axes correctly.
2) Sign of TVC:
   - Positive pitch command should tilt thrust to create restoring torque in pitch axis.
   - You must verify servo direction on bench (swap sign if needed).
3) Pin mapping:
   - Replace pins with your actual wiring / PCB.
4) Output protocols:
   - ESC: standard 50 Hz PWM (1ms~2ms) as described in paper.
   - Servos: standard 50 Hz PWM.

Safety:
- This is for small experimental platforms only. Always use a physical kill switch and tether tests first.

Main Code
================================================================================

#include <Arduino.h>
#include <Wire.h>
#include <SPI.h>
#include <Servo.h>

// --------------------------------
// USER CONFIG (PINS / LIMITS)
// --------------------------------

// ESC / Servo output pins (EDIT to your wiring)
static const int PIN_ESC_MAIN   = 2;   // main EDF ESC PWM
static const int PIN_ESC_ROLL_L = 3;   // left roll EDF ESC PWM
static const int PIN_ESC_ROLL_R = 4;   // right roll EDF ESC PWM (optional: differential)

static const int PIN_SERVO_PITCH = 5;  // TVC pitch servo
static const int PIN_SERVO_YAW   = 6;  // TVC yaw servo

// IMU (MPU9250) SPI pins (Teensy uses hardware SPI; CS is chosen)
static const int PIN_IMU_CS = 10;

// I2C (VL53L0X) uses Wire (SDA/SCL fixed by board)
static const uint8_t I2C_ADDR_VL53L0X = 0x29;
```

```
// GNSS UART (BK-252Q) - choose a HardwareSerial port you used
// Teensy 4.1 has Serial1..Serial8. Here we use Serial1.
#define GNSS_SERIAL Serial1
static const uint32_t GNSS_BAUD = 9600;

// Control loop rates
static const uint32_t LOOP_HZ_FAST = 500;  // AHRS + rate control (e.g., 500 Hz)
static const uint32_t LOOP_HZ_SLOW = 50;   // GNSS/ToF + outer control (e.g., 50 Hz)

// PWM ranges (microseconds)
static const int PWM_MIN = 1000;
static const int PWM_MAX = 2000;
static const int PWM_ARM = 1000;

// TVC mechanical limit (deg) - set to your gimbal's safe range
static const float TVC_MAX_DEG = 20.0f;

// Throttle limit (0..1)
static const float THR_MIN = 0.0f;
static const float THR_MAX = 1.0f;

// Failsafe timeouts
static const uint32_t SENSOR_TIMEOUT_MS = 200; // if IMU stops updating -> failsafe

// ------------------------------
// SIMPLE MATH HELPERS
// ------------------------------
static inline float clampf(float x, float lo, float hi) {
  return (x < lo) ? lo : (x > hi) ? hi : x;
}
static inline float deg2rad(float d) { return d * 0.017453292519943295f; }
static inline float rad2deg(float r) { return r * 57.29577951308232f; }

// Map [-1..1] to [1000..2000] (ESC/servo style)
static inline int pwmFromUnit(float u) {
  u = clampf(u, -1.0f, 1.0f);
  float mid = 1500.0f;
  float amp = 500.0f;
  return (int)(mid + amp * u);
}

// Map throttle [0..1] to [1000..2000]
static inline int pwmFromThrottle(float t) {
  t = clampf(t, 0.0f, 1.0f);
  return (int)(PWM_MIN + (PWM_MAX - PWM_MIN) * t);
}

// ------------------------------
// QUATERNION + MADGWICK (minimal)
// ------------------------------
struct Quaternion {
  float w, x, y, z;
};

static Quaternion q_hat {1,0,0,0};      // estimated attitude
static float gyro_b[3] = {0,0,0};       // p,q,r (rad/s)
static float acc_b[3]  = {0,0,0};       // ax,ay,az (m/s^2 or g)
```

```
static float mag_b[3]  = {0,0,0};        // optional

// Madgwick gain (tune)
static float beta = 0.08f;

// Normalize quaternion
static void quatNormalize(Quaternion &q) {
  float n = sqrtf(q.w*q.w + q.x*q.x + q.y*q.y + q.z*q.z);
  if (n > 1e-9f) { q.w/=n; q.x/=n; q.y/=n; q.z/=n; }
}

// Convert quaternion to Euler (roll, pitch, yaw) in radians
static void quatToEuler(const Quaternion &q, float &roll, float &pitch, float &yaw) {
  // roll (x-axis rotation)
  float sinr_cosp = 2.0f * (q.w*q.x + q.y*q.z);
  float cosr_cosp = 1.0f - 2.0f * (q.x*q.x + q.y*q.y);
  roll = atan2f(sinr_cosp, cosr_cosp);

  // pitch (y-axis rotation)
  float sinp = 2.0f * (q.w*q.y - q.z*q.x);
  if (fabsf(sinp) >= 1.0f) pitch = copysignf(PI/2.0f, sinp);
  else pitch = asinf(sinp);

  // yaw (z-axis rotation)
  float siny_cosp = 2.0f * (q.w*q.z + q.x*q.y);
  float cosy_cosp = 1.0f - 2.0f * (q.y*q.y + q.z*q.z);
  yaw = atan2f(siny_cosp, cosy_cosp);
}

// Minimal Madgwick IMU update (gyro+acc only).
// (Mag is optional; you can extend later if yaw drift matters.)
static void madgwickUpdateIMU(float gx, float gy, float gz, float ax, float ay, float az, float dt) {
  // Normalize accelerometer
  float norm = sqrtf(ax*ax + ay*ay + az*az);
  if (norm < 1e-9f) return;
  ax /= norm; ay /= norm; az /= norm;

  float q1=q_hat.w, q2=q_hat.x, q3=q_hat.y, q4=q_hat.z;

  // Gradient descent corrective step (IMU version)
  float f1 = 2.0f*(q2*q4 - q1*q3) - ax;
  float f2 = 2.0f*(q1*q2 + q3*q4) - ay;
  float f3 = 2.0f*(0.5f - q2*q2 - q3*q3) - az;

  float J_11or24 = 2.0f*q3;
  float J_12or23 = 2.0f*q4;
  float J_13or22 = 2.0f*q1;
  float J_14or21 = 2.0f*q2;
  float J_32 = 2.0f*J_14or21;
  float J_33 = 2.0f*J_11or24;

  float s1 = -J_11or24*f1 + J_14or21*f2;
  float s2 =  J_12or23*f1 + J_13or22*f2 - J_32*f3;
  float s3 = -J_13or22*f1 + J_12or23*f2 - J_33*f3;
  float s4 =  J_14or21*f1 + J_11or24*f2;

  norm = sqrtf(s1*s1 + s2*s2 + s3*s3 + s4*s4);
```

```
  if (norm < 1e-9f) return;
  s1/=norm; s2/=norm; s3/=norm; s4/=norm;

  // Quaternion derivative from gyro
  float qDot1 = 0.5f * (-q2*gx - q3*gy - q4*gz) - beta*s1;
  float qDot2 = 0.5f * ( q1*gx + q3*gz - q4*gy) - beta*s2;
  float qDot3 = 0.5f * ( q1*gy - q2*gz + q4*gx) - beta*s3;
  float qDot4 = 0.5f * ( q1*gz + q2*gy - q3*gx) - beta*s4;

  q_hat.w += qDot1 * dt;
  q_hat.x += qDot2 * dt;
  q_hat.y += qDot3 * dt;
  q_hat.z += qDot4 * dt;
  quatNormalize(q_hat);
}

// ------------------------------
// PID CONTROLLERS (dual-loop)
// ------------------------------
struct PID {
  float kp{0}, ki{0}, kd{0};
  float integ{0};
  float prevErr{0};
  float outMin{-1}, outMax{1};
  float iMin{-0.5f}, iMax{0.5f};

  float update(float err, float dt, float derr_meas=0.0f, bool use_meas_derivative=false) {
    // Integrator
    integ += err * dt;
    integ = clampf(integ, iMin, iMax);

    float deriv;
    if (use_meas_derivative) {
      // Use measured derivative (e.g., -rate) to avoid derivative kick
      deriv = derr_meas;
    } else {
      deriv = (dt > 1e-6f) ? (err - prevErr)/dt : 0.0f;
      prevErr = err;
    }

    float u = kp*err + ki*integ + kd*deriv;
    return clampf(u, outMin, outMax);
  }

  void reset() { integ = 0; prevErr = 0; }
};

// Outer loop: angle -> desired rate (rad/s)
static PID pidAngPitch, pidAngYaw, pidAngRoll;
// Inner loop: rate -> deflection command (unit)
static PID pidRatePitch, pidRateYaw, pidRateRoll;

// Desired setpoints (from guidance / RC)
static float roll_d = 0.0f;
static float pitch_d = 0.0f;
static float yaw_d = 0.0f;
static float throttle_d = 0.0f; // 0..1
```

```
// Estimated euler
static float roll=0, pitch=0, yaw=0;

// Rate commands (rad/s)
static float p_d=0, q_d=0, r_d=0;

// Outputs (unit -1..1)
static float u_pitch=0, u_yaw=0, u_roll=0;
static float out_thr=0;

// ------------------------------
// ACTUATORS
// ------------------------------
static Servo escMain, escRollL, escRollR;
static Servo servoPitch, servoYaw;

// Convert TVC command (unit) to servo PWM using mechanical limit
static int tvcServoPWM(float u_unit) {
  // u_unit is -1..1, map to -TVC_MAX..+TVC_MAX deg
  float angle_deg = clampf(u_unit, -1.0f, 1.0f) * TVC_MAX_DEG;
  // Map to servo PWM around 1500us; you may adjust scaling after calibration
  // Assume full deflection occurs at +/-500us (typical). Change if your linkage differs.
  float pwm = 1500.0f + (angle_deg / TVC_MAX_DEG) * 500.0f;
  return (int)clampf(pwm, PWM_MIN, PWM_MAX);
}

// ------------------------------
// IMU READ (PLACEHOLDER)
// ------------------------------
// You must replace with your MPU9250 SPI driver code.
// For paper consistency: MPU9250 uses SPI; it must be powered/logic at 3.3V. :contentReference[oaicite:13]{index=13}
static bool imuReadOK = false;
static uint32_t lastImuMs = 0;

static void imuInit() {
  SPI.begin();
  pinMode(PIN_IMU_CS, OUTPUT);
  digitalWrite(PIN_IMU_CS, HIGH);

  // TODO: write registers to configure gyro/acc ranges, sample rate, etc.
  // Keep this minimal in the paper; you can reference an MPU9250 SPI driver module.
}

static void imuRead() {
  // TODO: replace with actual SPI transactions.
  // For now, set imuReadOK=false to indicate you must plug your driver.
  imuReadOK = false;

  // Example expected outputs:
  // gyro_b[0]=p, gyro_b[1]=q, gyro_b[2]=r (rad/s)
  // acc_b[0]=ax, acc_b[1]=ay, acc_b[2]=az (normalized ok)
  // lastImuMs = millis();
}

// ------------------------------
// ToF / GNSS (MINIMAL PLACEHOLDER)
```

```
// ------------------------------
static float z_hat = 0.0f;    // altitude (m)
static float vz_hat = 0.0f;   // vertical speed (m/s)
static uint32_t lastSlowMs = 0;

static void tofInit() {
  Wire.begin();
  // TODO: VL53L0X init (address 0x29)
  // In paper, VL53L0X uses I2C. :contentReference[oaicite:14]{index=14}
}

static void gnssInit() {
  GNSS_SERIAL.begin(GNSS_BAUD);
  // TODO: parse NMEA/UBX depending on BK-252Q configuration
}

static void slowNavUpdate(float dt) {
  // TODO: read VL53L0X range -> z_hat
  // TODO: read GNSS -> lat/lon -> ENU -> p_hat, v_hat
  // For appendix/paper: explain this is low-rate correction layer.

  // Placeholder (no-op)
  (void)dt;
}

// ------------------------------
// GUIDANCE (SIMPLE)
// ------------------------------
// Paper states guidance is simplified (desired state from RC / computer). :contentReference[oaicite:15]{index=15}
static void guidanceUpdate() {
  // TODO: Replace with RC receiver parsing (PPM/SBUS/etc).
  // For now, hold a neutral attitude and manual throttle.
  roll_d = 0.0f;
  pitch_d = 0.0f;
  yaw_d = 0.0f;
  throttle_d = 0.25f;  // example hover-ish throttle; you must tune.
}

// ------------------------------
// CONTROL (DUAL-LOOP)
// ------------------------------
static void controlInitGains() {
  // Outer loop (angle -> rate command)
  pidAngRoll.kp  = 3.0f; pidAngRoll.ki  = 0.0f; pidAngRoll.kd  = 0.0f;
  pidAngPitch.kp = 3.0f; pidAngPitch.ki = 0.0f; pidAngPitch.kd = 0.0f;
  pidAngYaw.kp   = 2.0f; pidAngYaw.ki   = 0.0f; pidAngYaw.kd   = 0.0f;

  // Inner loop (rate -> actuator)
  pidRateRoll.kp  = 0.12f; pidRateRoll.ki  = 0.02f; pidRateRoll.kd  = 0.002f;
  pidRatePitch.kp = 0.12f; pidRatePitch.ki = 0.02f; pidRatePitch.kd = 0.002f;
  pidRateYaw.kp   = 0.10f; pidRateYaw.ki   = 0.01f; pidRateYaw.kd   = 0.001f;

  // Output clamps
  pidRateRoll.outMin  = -1; pidRateRoll.outMax  = 1;
  pidRatePitch.outMin = -1; pidRatePitch.outMax = 1;
  pidRateYaw.outMin   = -1; pidRateYaw.outMax   = 1;
}
```

```
static void controlUpdateFast(float dt) {
  // 1) Convert attitude to Euler for intuitive control (you can also do quaternion error)
  quatToEuler(q_hat, roll, pitch, yaw);

  // 2) Outer loop: angle error -> desired rate (rad/s)
  float eRoll  = roll_d  - roll;
  float ePitch = pitch_d - pitch;

  // yaw: for VTOL you may keep yaw hold weak or rate-based
  float eYaw   = yaw_d   - yaw;

  // Outer PIDs output "desired rate" (rad/s). Clamp to safe values.
  p_d = clampf(pidAngRoll.update(eRoll, dt),  -2.0f, 2.0f);
  q_d = clampf(pidAngPitch.update(ePitch, dt),-2.0f, 2.0f);
  r_d = clampf(pidAngYaw.update(eYaw, dt),    -2.0f, 2.0f);

  // 3) Inner loop: rate error -> actuator command
  // Use derivative-on-measurement approach to reduce derivative kick:
  // For a rate controller, measured derivative is not needed; gyro itself is the state.
  float eP = p_d - gyro_b[0];
  float eQ = q_d - gyro_b[1];
  float eR = r_d - gyro_b[2];

  u_roll  = pidRateRoll.update(eP, dt);
  u_pitch = pidRatePitch.update(eQ, dt);
  u_yaw   = pidRateYaw.update(eR, dt);

  // 4) Throttle (manual or altitude hold later)
  out_thr = clampf(throttle_d, THR_MIN, THR_MAX);
}

// Mix outputs to actual actuators
static void writeActuators() {
  // TVC servos (pitch/yaw)
  int pwmPitch = tvcServoPWM(u_pitch);
  int pwmYaw   = tvcServoPWM(u_yaw);

  // Roll EDF: simplest is differential thrust:
  // left = base + roll, right = base - roll
  // If you have only one roll EDF (as paper suggests "상쇄 방향으로만"), map accordingly.
  float rollBase = out_thr;      // you may choose separate base
  float rollCmd  = 0.15f * u_roll; // scale; tune experimentally

  int pwmMain  = pwmFromThrottle(out_thr);

  int pwmRollL = pwmFromThrottle(clampf(rollBase + rollCmd, 0.0f, 1.0f));
  int pwmRollR = pwmFromThrottle(clampf(rollBase - rollCmd, 0.0f, 1.0f));

  // Write
  servoPitch.writeMicroseconds(pwmPitch);
  servoYaw.writeMicroseconds(pwmYaw);

  escMain.writeMicroseconds(pwmMain);
  escRollL.writeMicroseconds(pwmRollL);
  escRollR.writeMicroseconds(pwmRollR);
}
```

```
// Failsafe: shut down thrust, neutral TVC
static void failsafeKill() {
  escMain.writeMicroseconds(PWM_ARM);
  escRollL.writeMicroseconds(PWM_ARM);
  escRollR.writeMicroseconds(PWM_ARM);
  servoPitch.writeMicroseconds(1500);
  servoYaw.writeMicroseconds(1500);

  pidAngRoll.reset(); pidAngPitch.reset(); pidAngYaw.reset();
  pidRateRoll.reset(); pidRatePitch.reset(); pidRateYaw.reset();
}

// ------------------------------
// SETUP / LOOP
// ------------------------------
static uint32_t tFastPrev=0, tSlowPrev=0;

void setup() {
  Serial.begin(115200);
  delay(500);

  // Actuators
  escMain.attach(PIN_ESC_MAIN, PWM_MIN, PWM_MAX);
  escRollL.attach(PIN_ESC_ROLL_L, PWM_MIN, PWM_MAX);
  escRollR.attach(PIN_ESC_ROLL_R, PWM_MIN, PWM_MAX);

  servoPitch.attach(PIN_SERVO_PITCH, PWM_MIN, PWM_MAX);
  servoYaw.attach(PIN_SERVO_YAW, PWM_MIN, PWM_MAX);

  // Arm outputs
  failsafeKill();
  delay(1000);

  // Sensors
  imuInit();
  tofInit();
  gnssInit();

  // Control
  controlInitGains();

  tFastPrev = micros();
  tSlowPrev = micros();

  Serial.println("GNC init done.");
}

void loop() {
  // FAST LOOP (AHRS + rate control)
  uint32_t tNow = micros();
  float dtFast = (tNow - tFastPrev) * 1e-6f;

  if (dtFast >= (1.0f / LOOP_HZ_FAST)) {
    tFastPrev = tNow;

    imuRead();
```

```
    if (imuReadOK) {
      lastImuMs = millis();

      // AHRS update
      madgwickUpdateIMU(
        gyro_b[0], gyro_b[1], gyro_b[2],
        acc_b[0],  acc_b[1],  acc_b[2],
        dtFast
      );

      // Guidance update (simple)
      guidanceUpdate();

      // Control update
      controlUpdateFast(dtFast);

      // Safety: if IMU stale -> kill
      if (millis() - lastImuMs > SENSOR_TIMEOUT_MS) {
        failsafeKill();
      } else {
        writeActuators();
      }

    } else {
      // IMU not working -> kill
      failsafeKill();
    }
  }

  // SLOW LOOP (GNSS/ToF corrections, future altitude hold)
  uint32_t tNow2 = micros();
  float dtSlow = (tNow2 - tSlowPrev) * 1e-6f;
  if (dtSlow >= (1.0f / LOOP_HZ_SLOW)) {
    tSlowPrev = tNow2;
    slowNavUpdate(dtSlow);

    // Optional: implement altitude hold:
    // throttle_d = throttle_from_altitude_pid(z_d - z_hat, vz_hat, ...);
  }

  // Debug print at low rate
  static uint32_t lastPrint = 0;
  if (millis() - lastPrint > 200) {
    lastPrint = millis();
    Serial.print("rpy(deg)= ");
    Serial.print(rad2deg(roll)); Serial.print(", ");
    Serial.print(rad2deg(pitch)); Serial.print(", ");
    Serial.print(rad2deg(yaw));
    Serial.print(" | u(p,y,roll)= ");
    Serial.print(u_pitch,3); Serial.print(", ");
    Serial.print(u_yaw,3); Serial.print(", ");
    Serial.print(u_roll,3);
    Serial.print(" | thr= ");
    Serial.println(out_thr,3);
  }
}
```