Quadcopter Dynamics

Innova Lee(이상훈)
gcccompil3r@gmail.com

Overview

1. 쿼드콥터 제어 블록 다이어그램 해석
2. 사용되는 센서들의 조사
3. 응용 분야 및 실제 구현 사례

Quadcopter Control Mechanics

우선 쿼드콥터의 블록 다이어그램을 해석하기 위해 특정 자료를 사용하도록 하겠습니다.

첫 번째 과제에서 보여드렸던 Quadcopter 를 구현하기 위해 참조했던 자료는 아래와 같습니다.
이 내용에 의거하여 쿼드콥터의 물리적 모델링에 기반한 블록 다이어그램 해석을 진행해보겠습니다.

https://www.intechopen.com/books/motion-control/intelligent-flight-control-of-an-autonomous-quadrotor

여기에 Quadrotor's Kinematics and Dynamics 라고 나오는 부분에 대한 설명을 적고
사이버랩 및 강의 노트에 있던 블록 다이어그램에 대한 설명을 함께 곁들이도록 하겠습니다.

Quadrotor's Kinematics and Dynamics

수학적 모델링은 시스템의 동작에 대한 설명을 제공합니다.
Quadrotor 의 비행 동작은 4 개의 모터 각각의 속도에 의해 결정되며
이것들은 서로 협조하기도 하며 서로 대립하기도 합니다.
(협조라는 것은 회전을 하며 만들어내는 양력을 합력을 의미하며 대립은 서로의 합력을 상쇄시킴을 의미합니다)
그러므로 입력에 따라 System 의 수학적 표현을 사용하여 Quadrotor 의 위치와 방향을 예측할 수 있습니다.
개별 모터의 속도를 조작하면 원하는 행동을 취할 수 있는 제어 전략을 개발할 수 있습니다.

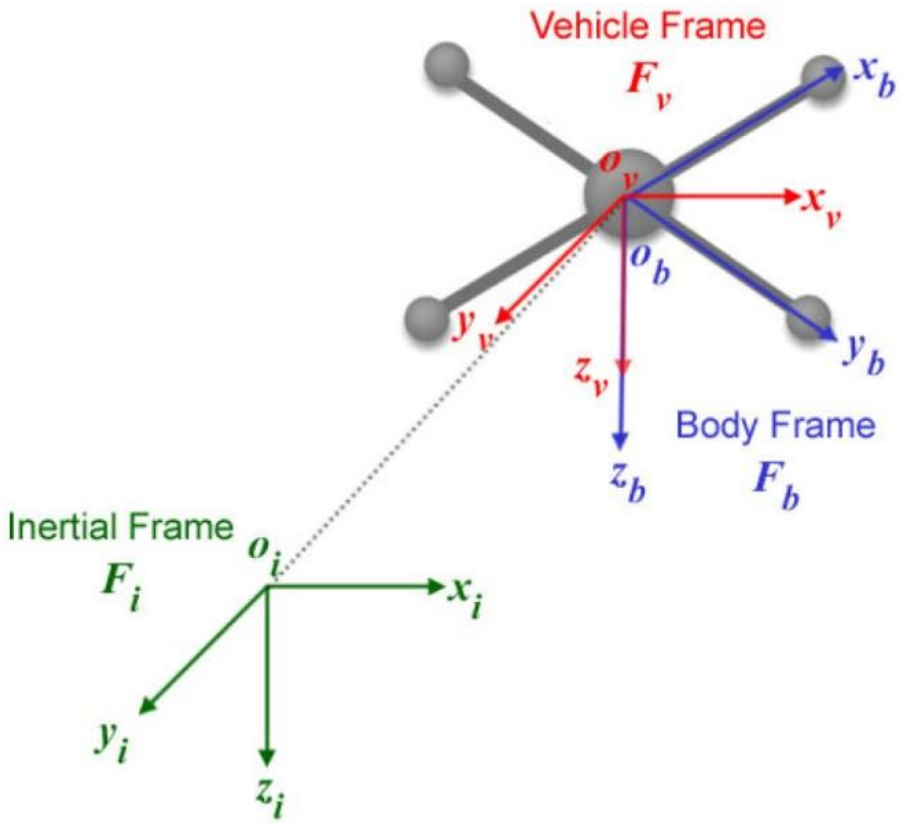
Quadrotor 의 전체 수학적 모델을 유도하려면 먼저 운동학 및 동역학을 정의해야 합니다.
운동학적 방정식은 차량의 위치와 속도 사이의 관계를 제공하는 반면
동역학적 모델은 적용된 힘과 그 결과로서의 가속도를 관리하는 관계를 정의합니다.

Reference Frames

운동학 방정식과 Quadrotor 의 동역학에 들어가기 전에 채택된 좌표계와 Reference Frame 을 지정하고
다른 좌표계 간의 변환이 수행되는 방법을 지정하도록 합니다.

다른 좌표 프레임의 사용은 6 DOF(6 개의 자유도)에서 Quadrotor 의 위치와 고도를 식별하는데 필수적입니다.
예로 운동 방정식을 평가하기 위해 quadrotor 에 부착된 좌표계가 필요합니다.
그러나 Quadrotor 에 작용하는 힘과 모멘트는 IMU 센서 값과 함께 Body Frame 을 기준으로 평가됩니다.
마지막으로 Quadrotor 의 위치와 속도는 조종기의 관성 프레임에 대해 GPS 측정을 사용하여 평가됩니다.

그러므로 아래와 같이 3 가지 주요 Reference Frame 이 채택된다:



관성 프레임 $F_i = (\overrightarrow{x_i}, \overrightarrow{y_i}, \overrightarrow{z_i})$ 는
조종기를 원점으로 잡고 있는 지구의 고종 좌표계입니다.
규칙에 따라 x 축은 북쪽을 향하고 y 축은 동쪽을 향하며
z 축은 지구의 중심을 향합니다.

Body Frame $F_b = (\overrightarrow{x_b}, \overrightarrow{y_b}, \overrightarrow{z_b})$ 은
원점이 Quadrotor 의 무게중심(COG) 에 존재합니다.
그리고 축은 Quadrotor 구조와 일직선으로 맞춰져서
x 축의 $\overrightarrow{x_b}$ 는 앞 부분의 motor 방향으로 뻗어나가며
y 축의 $\overrightarrow{y_b}$ 는 $\overrightarrow{x_b}$ 의 우측 motor 방향으로 뻗어나가고
z 축은 $\overrightarrow{z_b} = \overrightarrow{x_b} \times \overrightarrow{y_b}$ 과 같이 외적으로 나타납니다.

Vehicle Frame $F_v = (\overrightarrow{x_v}, \overrightarrow{y_v}, \overrightarrow{z_v})$ 은 관성 프레임이며
원점이 Quadrotor 의 무게중심(COG) 에 존재합니다.
Vehicle Frame 은 F_θ 및 F_ϕ 라는 2 개의 변형을 가집니다.
F_ϕ 은 x_v 와 y_v 가 각각 x_b 와 y_b 에 일직선으로 맞춰지도록
z 축의 z_v 를 중심으로 각도 v 만큼 회전된
Vehicle Frame F_v 를 의미합니다.
F_θ 은 Frame F_ϕ 가 y 축을 중심으로
y_ϕ 만큼 피칭 각도 θ 만큼 회전하여
x_ϕ 와 z_ϕ 가 각각 x_b 와 z_b 에 일직선이 되도록 합니다.

변환 및 회전 행렬은 하나의 좌표 Reference Frame 을 다른 원하는 Reference Frame 으로 변환하는데 사용됩니다.
예로 F_i 에서 F_v 로의 변환은 관성 프레임의 원점에서 Quadrotor 의 무게중심(COG)까지의 변위 벡터를 제공합니다.
또한 F_v 에서 F_b 로의 변환은 본질적으로 회전이므로 roll, pitch, yaw 각도를 산출합니다.

Quadrotor's Kinematics

$P_F^T = [p_x, p_y, -p_z]$ 와 $\Omega_F^T = [\phi, \theta, \psi]$ 는 주어진 Frame F 내에서의 Quadrotor 의 위치와 방향을 나타낸다.
3 개의 사전 정의된 Frame 에서 Quadrotor 의 속도 사이의 관계는 아래와 같이 표현됩니다.

$$\left[R_{F_v}^{F_b}\right]^T \in \mathbb{R}^{3\times 3}$$

위의 $\left[R_{F_v}^{F_b}\right]^T$ 는 Frame F_v 를 Frame F_b 에 맵핑하는 회전 행렬이며 아래와 같이 정의됩니다.

$$\left[R_{F_v}^{F_b}\right]^T = \begin{bmatrix} cos(\theta)cos(\psi) & sin(\phi)sin(\theta)cos(\psi) - cos(\phi)sin(\psi) & cos(\phi)sin(\theta)cos(\psi) + sin(\phi)sin(\psi) \\ cos(\theta)sin(\psi) & sin(\phi)sin(\theta)sin(\psi) + cos(\phi)sin(\psi) & cos(\phi)sin(\theta)sin(\psi) + sin(\phi)cos(\psi) \\ -sin(\theta) & sin(\phi)cos(\theta) & cos(\phi)cos(\theta) \end{bmatrix}$$

따라서 회전 운동 관계는 Vehicle Frame 각도(ϕ, θ, ψ) 및 Vehicle Frame 의 각속도$(\dot{\phi}, \dot{\theta}, \dot{\psi})$ 와 같은
적절한 상태 변수를 사용하여 도출할 수 있고 여기서 ψ 는 v 와 동일한 의미를 가집니다.
혼동을 방지하기 위해 앞으로 v 를 지속적으로 ψ 로 표기하도록 합니다.

위와 같이 하기 위해서는 위와 같은 변수를 하나의 공통된 Reference Frame 으로 가져와야 합니다.
Vehicle Frames F_ϕ, F_θ 및 $F_\psi == F_v$ 를 기준 F_b 의 Body Frame 으로 변환하기 위해
회전 행렬을 사용하여 아래의 식들을 얻습니다.

$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix}_{F_b} = R_{F_\phi}^{F_b}(\phi)\begin{bmatrix}\dot{\phi}\\ 0\\ 0\end{bmatrix} + R_{F_\phi}^{F_b}(\phi)R_{F_\theta}^{F_\phi}(\theta)\begin{bmatrix}0\\ \dot{\theta}\\ 0\end{bmatrix} + R_{F_\phi}^{F_b}(\phi)R_{F_\theta}^{F_\phi}(\theta)R_{F_\psi}^{F_\theta}(\psi)\begin{bmatrix}0\\ 0\\ \dot{\psi}\end{bmatrix}$$

여기서 각각의 의미는 아래와 같습니다.

$$R_{F_\theta}^{F_\phi} = \begin{bmatrix}1 & 0 & 0\\ 0 & cos(\phi) & sin(\phi)\\ 0 & -sin(\phi) & cos(\phi)\end{bmatrix}, \qquad R_{F_\psi}^{F_\theta} = \begin{bmatrix}cos(\theta) & 0 & -sin(\theta)\\ 0 & 1 & 0\\ sin(\theta) & 0 & cos(\theta)\end{bmatrix}$$

$$R_{F_\phi}^{F_b}(\phi) = R_{F_\theta}^{F_b}(\theta) = R_{F_\psi}^{F_b}(\psi) = I$$

그러므로 아래와 같이 정리됩니다.

$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix}_{F_b} = \begin{bmatrix}1 & 0 & -sin(\theta)\\ 0 & cos(\phi) & sin(\phi)cos(\theta)\\ 0 & -sin(\phi) & cos(\phi)cos(\theta)\end{bmatrix}\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix}_{F_\psi}$$

다음으로 아래와 같이 정리됩니다.

$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix}_{F_\psi} = \begin{bmatrix}1 & sin(\phi)tan(\theta) & cos(\phi)tan(\theta)\\ 0 & cos(\phi) & -sin(\phi)\\ 0 & sin(\phi)sec(\theta) & cos(\phi)sec(\theta)\end{bmatrix}\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix}_{F_b}$$

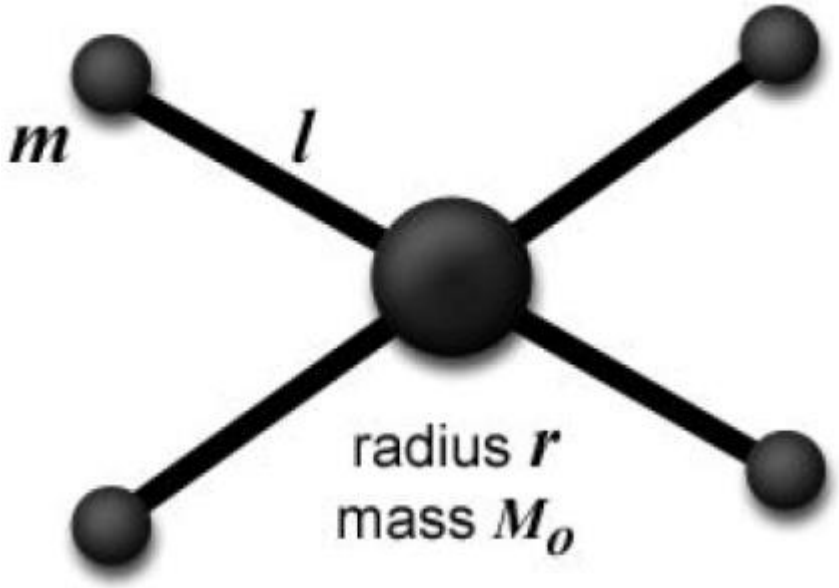
이것은 Quadrotor 의 운동 방정식을 표현합니다.

Quadrotor's Dynamics

Quadrotor 의 동적 모델을 만들려면 아래 가정을 취하면서 Netwon-Euler 형식론을 사용하도록 합니다.

1. **Quadrotor 의 구조는 강체에 해당합니다.**
2. **Quadrotor Frame 은 대칭입니다.**
3. **Quadrotor 의 COG(무게중심)은 강체의 중심과 일치합니다.**

관성 모멘트는 모터는 4 점 질량으로 가정한 이후
Quadrotor 를 반경 r 의 중심 구로 가정하고 모터에 둘러싸인 질량을 M_0 로 가정하여 계산하도록 합니다.
각 모터는 질량 m 이며 아래 그림과 같이 길이가 l 인 막대 끝부분의 중앙 구에 부착되어 있다고 가정합니다.



Quadrotor 는 모든 3 개의 축에 대해 대칭성을 가지므로 관성 행렬은 대칭이 되고 아래와 같이 정의됩니다.
원래 관성 모멘트는 I 로 표기하지만 전류 i 에 대한 문제가 있으므로 J 로 표기하도록 합니다.

$$J = \begin{bmatrix} j_x & 0 & 0 \\ 0 & j_y & 0 \\ 0 & 0 & j_z \end{bmatrix}$$

여기서 각 항목들은 아래와 같습니다.

$$j_x = j_y = j_z = \frac{2M_0 r^2}{5} + 2l^2 m$$

COG 에 적용되고 Body Frame 에서 표현된 외력에 의한 Quadrotor 의 동역학식은
아래와 같이 Newton-Euler 공식 Beard(2008) 를 적용하여 유도됩니다.

$$\begin{bmatrix} MI_{3\times3} & 0 \\ 0 & J_{3\times3} \end{bmatrix} \begin{bmatrix} \ddot{P}_{F_b} \\ \dot{\Omega}_{F_b} \end{bmatrix} + \begin{bmatrix} \dot{\Omega}_{F_b} \times M\dot{P}_{F_b} \\ \dot{\Omega}_{F_b} \times J\dot{\Omega}_{F_b} \end{bmatrix} = \begin{bmatrix} F_{F_b} \\ \tau_{F_b} \end{bmatrix}$$

위 식의 유도 부분은 아래 링크에 잘 설명되어 있습니다.
http://rwbclasses.groups.et.byu.net/lib/exe/fetch.php?media=quadrotor:beardsquadrotornotes.pdf

고정된 좌표계가 이동 좌표계를 바라볼 때 이동 좌표계의 어떤 변위 p 가 고정된 상태로 회전하는 경우를 고려해봅니다.
고정 좌표계 관점에서 이동 좌표계의 p 점을 바라보게되면 p 는 움직이지 않았지만
회전했으므로 각속도를 가지고 있는것 처럼 느껴지게 됩니다.
그렇기에 변위 p 의 변화량은 아래와 같이 표기되며 증명은 위 링크의 pdf 에 적혀있습니다.

$$\frac{d}{dt_i} p = \frac{d}{dt_b} p + \omega_{b/i} \times p$$

$$\begin{bmatrix} MI_{3\times3} & 0 \\ 0 & J_{3\times3} \end{bmatrix} \begin{bmatrix} \ddot{P}_{F_b} \\ \ddot{\Omega}_{F_b} \end{bmatrix} + \begin{bmatrix} \dot{\Omega}_{F_b} \times M\dot{P}_{F_b} \\ \dot{\Omega}_{F_b} \times J\dot{\Omega}_{F_b} \end{bmatrix} = \begin{bmatrix} F_{F_b} \\ \tau_{F_b} \end{bmatrix}$$

여기서 M 은 Quadrotor 의 총 질량이며 $F = [f_x, f_y, f_z]$ 및 $\tau = [\tau_\phi, \tau_\theta, \tau_\psi]$ 는
Quadrotor 의 COG 에 적용되는 외부 힘 및 토크 벡터입니다.
용어 $\tau_\phi, \tau_\theta, \tau_\psi$ 는 각각 Roll, Pitch, Yaw 에 해당하는 토크입니다.
그러므로 변환한 동역학 모델은 아래와 같이 쓰여질 수 있다.
위 식에서 P 는 앞서 살펴보았듯이 변위에 해당하며 Ω 는 각도로 받아들이면 되겠습니다.

여기서 하나 더 살펴봐야할 것이 각도에 대한 2 계 미분인데 이것은 각가속도인 α 가 됩니다.
회전력인 토크에 대한 정보가 나왔으므로 각운동량 보존의 법칙등을 사용하여 아래와 같은 결론을 도출할 수 있습니다.
각운동량은 보편적으로 L 로 표기하므로 문자를 그대로 사용하겠습니다.

$$\tau = \frac{dL}{dt} = \frac{d(r \times p)}{dt} = \frac{dr}{dt} \times p + r \times \frac{dp}{dt} = J\alpha$$

일반적인 상황에서 r 에 변화율이 없기 때문에 결국 $\tau = r \times F$ 가 됩니다.
그러나 이번에는 고정 좌표계가 이동 좌표계를 바라보는 관점이기에 상황이 변합니다.

$$\tau = \frac{dL}{dt_i} = \frac{d(r \times p)}{dt_b} + \omega_{b/i} \times (r \times p)$$

결국 위의 식이 문제가 없음을 개략적으로 확인할 수 있습니다.

위의 정보들을 기반으로 병진운동의 동역학 모델을 기술해보겠습니다.

$$\begin{bmatrix} \ddot{p}_x \\ \ddot{p}_y \\ \ddot{p}_z \end{bmatrix}_{F_b} = \begin{bmatrix} \dot{\psi}\dot{p}_y - \dot{\theta}\dot{p}_z \\ \dot{\phi}\dot{p}_z - \dot{\psi}\dot{p}_x \\ \dot{\theta}\dot{p}_x - \dot{\phi}\dot{p}_y \end{bmatrix}_{F_b} + \frac{1}{M}\begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix}_{F_b}$$

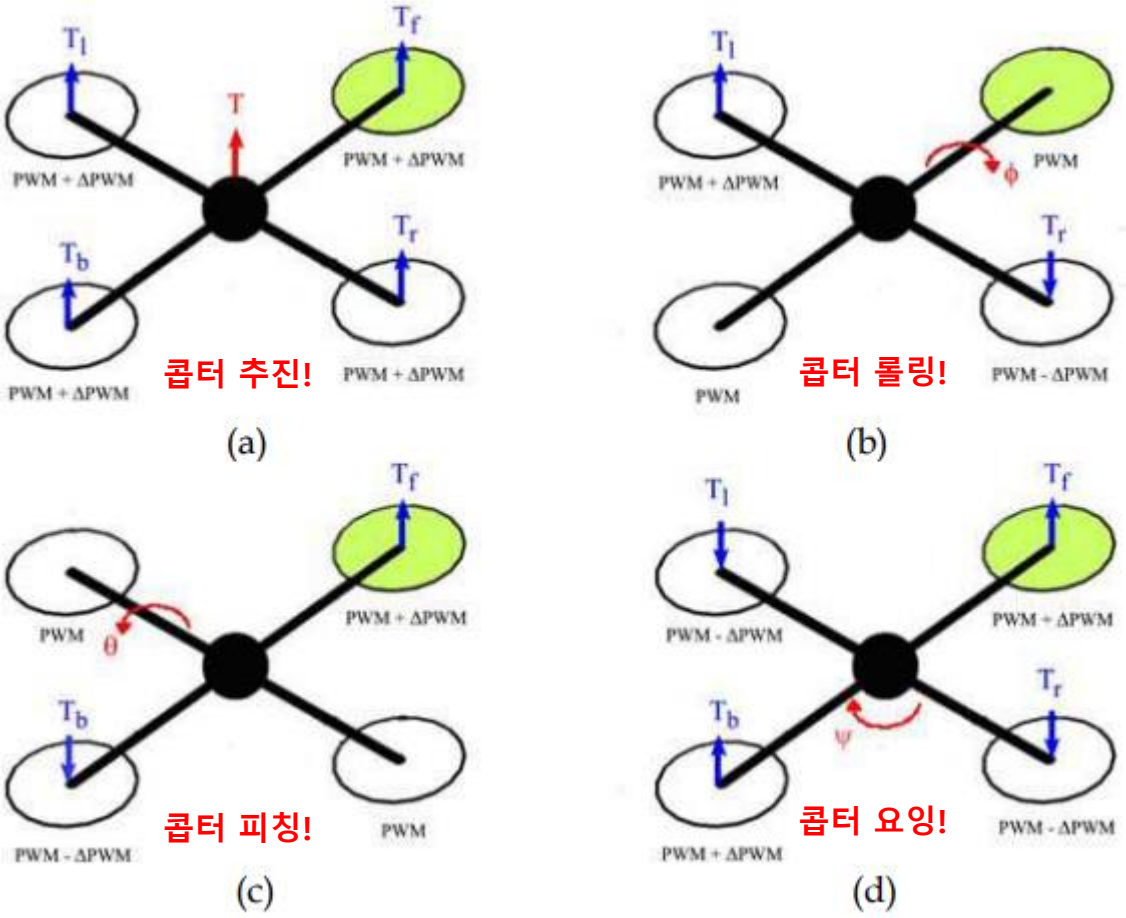
다음으로 회전 동역학 모델을 기술해보자면 아래와 같습니다.

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix}_{F_b} = J^{-1}\left\{\begin{bmatrix} 0 & \dot{\psi} & -\dot{\theta} \\ -\dot{\psi} & 0 & \dot{\phi} \\ \dot{\theta} & -\dot{\phi} & 0 \end{bmatrix} J \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix}\right\} = \begin{bmatrix} \frac{j_y - j_z}{j_x}\dot{\theta}\dot{\psi} \\ \frac{j_z - j_x}{j_y}\dot{\phi}\dot{\psi} \\ \frac{j_x - j_y}{j_z}\dot{\phi}\dot{\theta} \end{bmatrix}_{F_b} + \begin{bmatrix} \frac{1}{j_x}\tau_\phi \\ \frac{1}{j_y}\tau_\theta \\ \frac{1}{j_z}\tau_\psi \end{bmatrix}_{F_b}$$

여기서 관성 모멘트를 역수로 적지 않고 inverse 표기한 이유는 행렬이기 때문입니다.
이제 공기역학적인 부분을 해석해보도록 하겠습니다.

Aerodynamic Forces and Torques

유도된 운동학 및 동역학 모델을 사용하여 Quadroter 에 작용하는 힘과 토크를 정의합니다.
힘은 각 회전자에 의해 생성된 공기역학적 양력과 생성된 총 양력에 역행하는 중력을 포함합니다.
모멘트는 Roll, Pitch, Yaw 운동을 수행하기 위해 생성된 토크입니다.
그러므로 아래와 같은 힘과 토크가 생성됩니다.



Upward Force(Thrust)

Quadroter 의 총 추력은 각 프로펠러에 의해 생성된 추력의 합에 해당합니다.

$$T = T_f + T_r + T_b + T_l$$

Rolling Torque

왼쪽 로터의 추력을 증가시키면서 오른쪽 로터의 추력을 감소시킵니다.
혹은 그 반대로 수행해도 무방하며 이 경우 Rolling Torque 가 발생합니다.

$$\tau_\phi = l(T_l - T_r)$$

Pitching Torque

피칭 토크는 전방 로터의 추력을 증가시키면서 후방 로터의 출력을 증가시키거나 그 반대로 증가시킴으로써 생성됩니다.

$$\tau_\theta = l\left(T_f - T_b\right)$$

Yawing Torque

요잉 토크는 회전하는 로터로 인해 생성된 4 가지 개별 토크의 결과입니다.
전면 및 후면 로터는 시계 방향으로 회전하며 왼쪽 및 오른쪽 로터는 반시계 방향으로 회전합니다.
이들 두 쌍 사이의 불균형은 요잉 토크를 유발하여 Quadroter 가 z 축에 대해 회전하게 만듭니다.

$$\tau_\psi = \tau_f + \tau_b - \tau_r - \tau_l$$

Gravitational Force(weight)

다른 힘과 함께 중력은 Quadroter 의 COG 에 작용합니다.
Vehicle Frame 에서 이 힘은 아래와 같이 표기되며 g 는 중력 상수입니다.

$$W_{F_\psi} = \begin{bmatrix} 0 \\ 0 \\ Mg \end{bmatrix}$$

그러므로 Body Frame 에서 무게는 아래와 같이 표기됩니다.

$$W_{F_\psi} = R_{F_\psi}^{F_b} \begin{bmatrix} 0 \\ 0 \\ Mg \end{bmatrix} = \begin{bmatrix} -Mgsin(\theta) \\ Mgcos(\theta)sin(\phi) \\ Mgcos(\theta)cos(\phi) \end{bmatrix}$$

System 에 작용하는 힘 및 토크를 포함하여 운동 방정식을 재설계 하면 아래와 같이 기술할 수 있습니다.

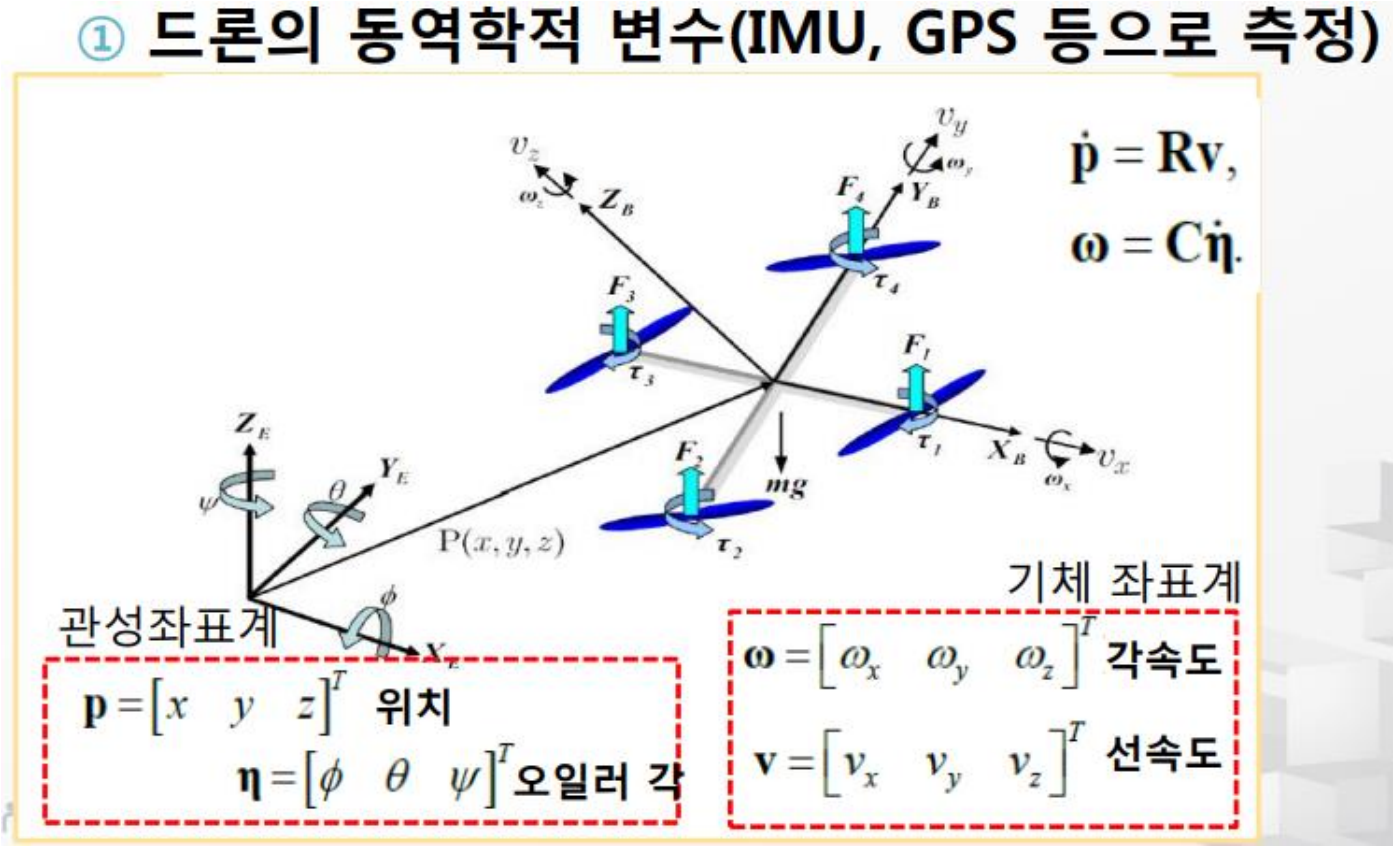
$$\begin{bmatrix} \ddot{p}_x \\ \ddot{p}_y \\ \ddot{p}_z \end{bmatrix}_{F_b} = \begin{bmatrix} \dot{\psi}\dot{p}_y - \dot{\theta}\dot{p}_z \\ \dot{\phi}\dot{p}_z - \dot{\psi}\dot{p}_x \\ \dot{\theta}\dot{p}_x - \dot{\phi}\dot{p}_y \end{bmatrix}_{F_b} + \begin{bmatrix} -gsin(\theta) \\ gcos(\theta)sin(\phi) \\ gcos(\theta)cos(\phi) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -\frac{f_z}{M} \end{bmatrix}$$

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix}_{F_b} = \begin{bmatrix} \frac{j_y - j_z}{j_x}\dot{\theta}\dot{\psi} \\ \frac{j_z - j_x}{j_y}\dot{\phi}\dot{\psi} \\ \frac{j_x - j_y}{j_z}\dot{\phi}\dot{\theta} \end{bmatrix}_{F_b} + \begin{bmatrix} \frac{1}{j_x}\tau_\phi \\ \frac{1}{j_y}\tau_\theta \\ \frac{1}{j_z}\tau_\psi \end{bmatrix}_{F_b}$$

Control Block Diagram Analysis

앞서 물리적인 절차를 살펴보면서 해당 내용을 파악할 수 있었습니다.

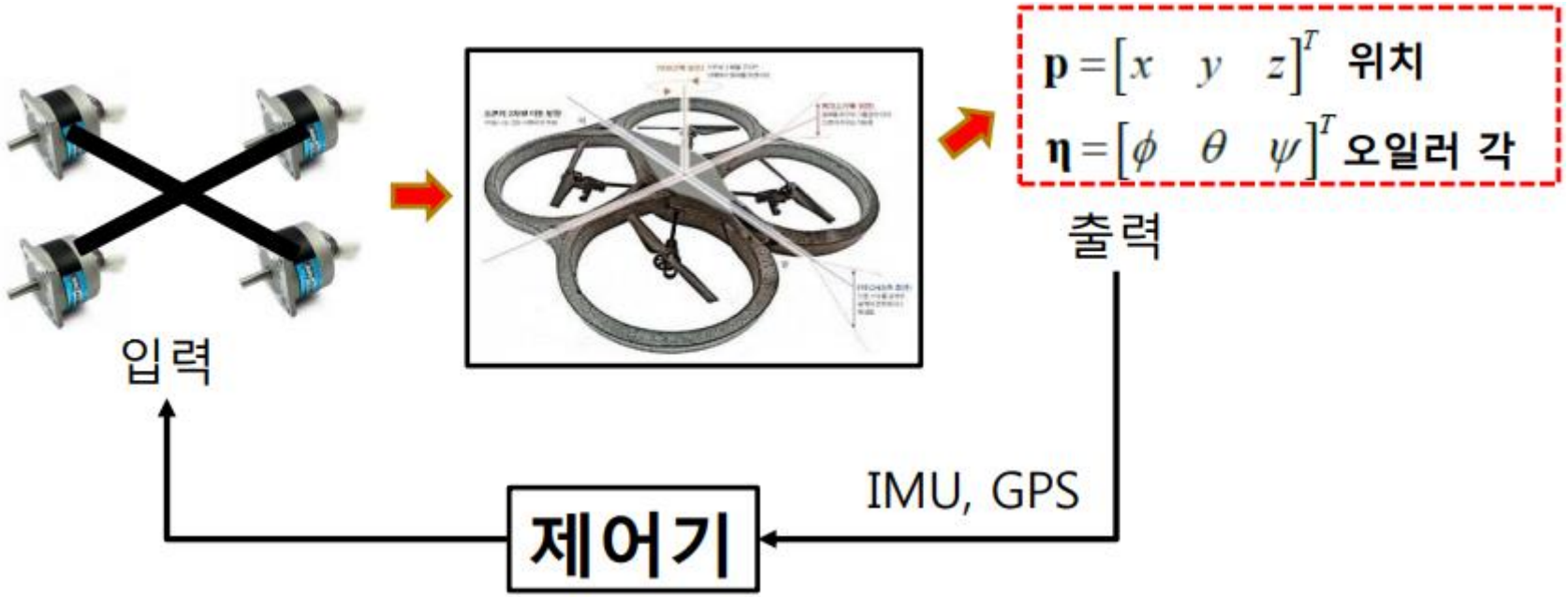
내용에는 빠져 있지만 관성 좌표계, 차체 좌표계, 기체 좌표계로 구성되며 각 벡터의 표현들이 보이고 있습니다.



Quadcopter Dynamics Model

쿼드콥터의 동역학적 모델을 살펴보겠습니다.

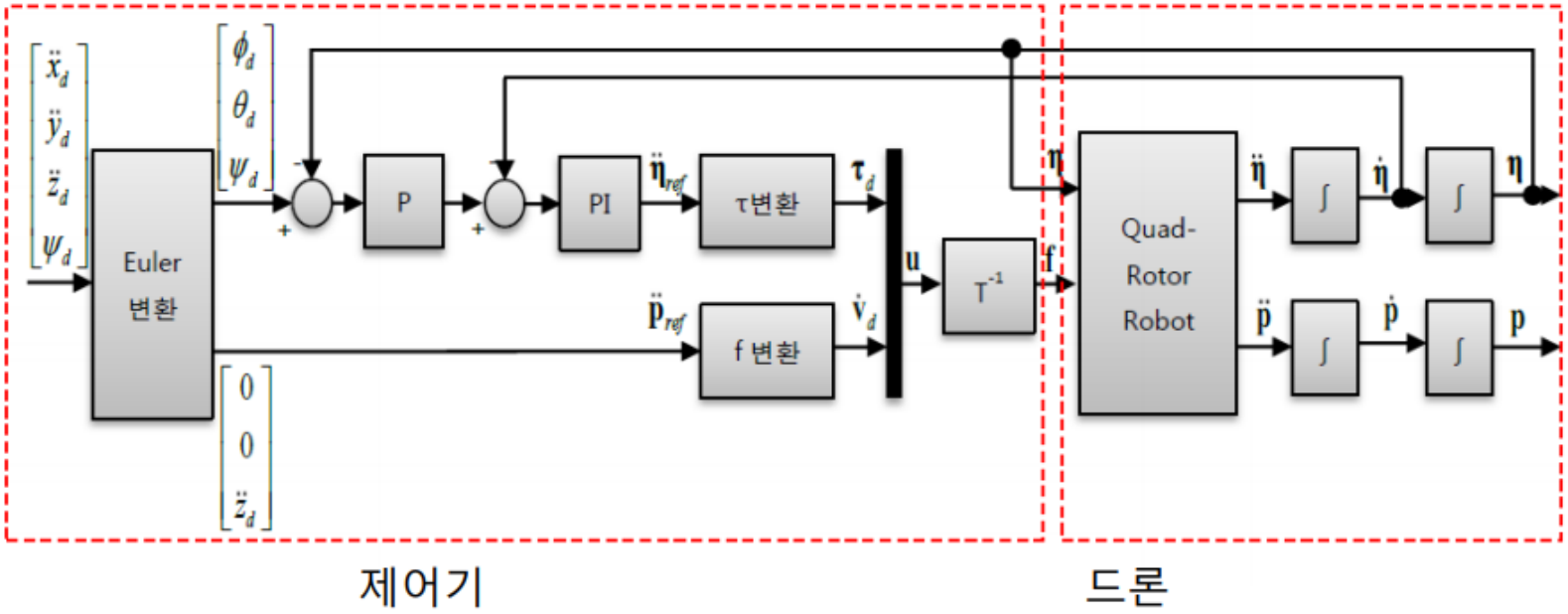
앞선 과제에서도 IMU 센서인 MPU-6050 을 통해서 가속도, 각속도를 얻을 수 있고
이를 기반으로 미분 및 적분을 활용하여 정보를 도출해낼 수 있음을 알고 있습니다.
여기서 제어기는 실제로 센서로 부터 출력된 정보를 입력 받아 PI, PD, PID 등의 제어기로 입력하고
내부에서 상보 필터나 칼만 필터등이 함께 활용될 수 있습니다.



Understanding Quadcopter Controller

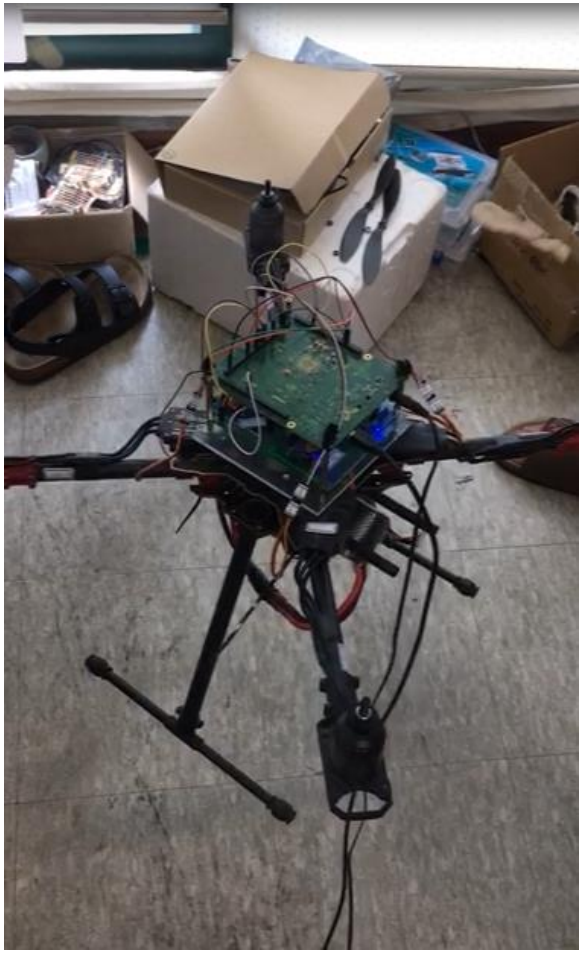
쿼드콥터의 동역학적 모델에 대한 제어기 부분을 살펴보겠습니다.

처음 블록 다이어그램을 보면 2 계 미분된 형태의 변위값으로 가속도 성분이 들어오고
이에 대한 Euler 변환을 수행하여 가속도와 각도와의 관계식을 유도합니다.
이 상태에서 관성 좌표계, 차체 좌표계(Body Frame), 기체 좌표계(Vehicle Frame)에서의 각도를
P 제어기에 전달하며 초기에 센서로부터 피드백은 없습니다.
또한 이 정보가 PI 제어기로 입력되며 마찬가지로 처음에 피드백은 없습니다.
이후 토크에 대한 식을 도출하기 위해 τ 변환을 통해 각가속도에 대한 정보를 도출합니다.
똬 중력 가속도에 대한 정보와 관성 좌표계의 가속도 정보를 기반으로
Vehicle Frame 에서의 가속도를 f 변환을 통해 얻습니다.
그리고 inverse T 행렬을 통해 힘과 토크에 대한 정보가 Quadcopter 에 전달됩니다.
토크와 힘은 모두 가속도와 각가속도를 가지고 있으므로 적분을 하면 속도, 각속도 한 번 더 하면 변위와 각도를 얻습니다.
이 정보들을 Feedback 으로 전달하여 오차를 수정하는 작업으로 쿼드콥터가 자세를 제어하게 됩니다.



Actual Application

실제 구현 및 적용 사례를 살펴보겠습니다.



쿼드콥터 프로젝트의 경우에는 PD 제어기를 활용하였으며
센서의 노이즈 및 기능이 많아 RFI(무선 신호 간섭) 로 인해
필터 처리를 수행하고 급격한 변화에도 잘 대응할 수 있도록
칼만 필터를 추가적으로 구현하였습니다.
또한 기능(Task)이 많다보니 RTOS 의 존재도 필요했습니다.
(참고로 모터는 ePWM 제어를 수행하였습니다)

전투함 프로젝트의 경우에는 PI 제어기를 사용하였기 때문에
저희 수업에서 설명해주신 블록 다이어그램과 유사한 형태를 가집니다.
물론 이번에는 자세 제어를 위해 내부에 자이로스코프 제어기가 추가됩니다.

선박은 쿼드콥터처럼 모터 기반 제어가 아니기 때문이며
자이로스코프는 모터로 제어되기에 ePWM 제어는 동일하게 사용되었습니다.

뿐만 아니라 이와 같은 PI, PD, PID 제어기는 아래와 같이 RC 자동차 및 전기 자동차를 제어하는데 활용할 수 있습니다.

추가적으로 근래 시도하고 있는 RC 제트기의 경우에는
아래와 같은 엔진 제어를 수행하는데
RC 계열의 쿼드콥터, 자동차, 선박은
전기와 역학 제어만 고려하면 될 사항에서
아래 엔진의 경우엔 항공유를 사용하며
적절한 공기와 기름의 적절한 혼합비, 압력, 유량, 온도 등에 따른
제어가 보다 복잡하고 밀접하게 관계가 되어 있어 훨씬 복잡합니다만
Feedback 을 기반으로 동작한다는 점에서
Feedback 제어를 사용하는 응용 사례라고 볼 수 있다 판단하였습니다.

```c
// FreeRTOS 스케쥴러
#include "FreeRTOS.h"
#include "os_task.h"

#define UART                    sciREG1
#define DATA_COUNT              14
#define MPU6050_ADDRESS         0x68 // MPU6050
#define RA_COUNT                1
#define MAX                     38U
#define PI                      3.14159265358979323846264338327950288419716939937510582097494459230781640628620899 86
#define n 2
#define m 1
#define r 1

volatile char RX_Data_Master[DATA_COUNT] = { 0 };
volatile int delay;
uint32 receiveData = 0;

unsigned int ePWM4A = 0, ePWM6B = 0, ePWM3B = 0, ePWM1B = 0;
unsigned char RA = 0x3B;
unsigned char Device_Init_Address = 0x6B;
unsigned char Device_Init_Option = 0x00;
unsigned char Gyro_Config_Address = 0x1B;
unsigned char Gyro_Init_Option = 0x18;
unsigned char Accel_Config_Address = 0x1C;
unsigned char Accel_Init_Option = 0x18;

float u;
float y;

float s_00;
float inn_00;
```

```
float buf[100];
float real_value[100];
float roll_output[200];
float pitch_output[200];
signed short temp = 0;
double accelX = 0.0, accelY = 0.0, accelZ = 0.0;
double gyroX = 0.0, gyroY = 0.0, gyroZ = 0.0;
double clk;
float dt = 0.00098147;
float roll_gyro_input;
float pitch_gyro_input;
float roll_accel_input;
float pitch_accel_input;
float roll_kalman_output = 0.0;
float pitch_kalman_output = 0.0;
kalman roll_kalman_state;
kalman pitch_kalman_state;

xTaskHandle xTask1Handle, xTask2Handle, xTask3Handle;

void kalman_init(kalman *self, float dt, float Sz_00, float Sw_00, float Sw_11);
float kalman_update(kalman *self,  float gyroscope_rate,  float accelerometer_angle);
```

```
void vTask3(void *pvParameters) {
        while(1) {
                for (delay = 0; delay < 250000; delay++);
                dt += 0.00098147;

                roll_gyro_input = gyroX;
                pitch_gyro_input = gyroY;
                roll_accel_input = atan(accelY/sqrt(accelX*accelX + accelZ*accelZ))*180/PI;
                pitch_accel_input = atan(accelX/sqrt(accelY*accelY + accelZ*accelZ))*180/PI;

                roll_kalman_output = kalman_update(&roll_kalman_state, roll_gyro_input, roll_accel_input);
                sprintf(roll_output, "roll_gyro(x)=%7.3f deg/sec\troll_accel(x)=%7.3f deg\troll_kalman_output(x)=%10.4f deg",
                                roll_gyro_input, roll_accel_input, roll_kalman_output);
                sciDisplayText(UART, (uint8 *)roll_output, strlen(roll_output));
                sciSendByte(UART, 10);
                sciSendByte(UART, 13);

                pitch_kalman_output = kalman_update(&pitch_kalman_state, pitch_gyro_input, pitch_accel_input);
                sprintf(pitch_output, "pitch_gyro(x)=%7.3f deg/sec\tpitch_accel(x)=%7.3f deg\tpitch_kalman_output(x)=%10.4f deg",
                                pitch_gyro_input, pitch_accel_input, pitch_kalman_output);
                sciDisplayText(UART, (uint8 *)pitch_output, strlen(pitch_output));
                sciSendByte(UART, 10);
                sciSendByte(UART, 13);
                sciSendByte(UART, 10);
                sciSendByte(UART, 13);

        }
}
```

```
void kalman_init(kalman *self, float dt, float Sz_00, float Sw_00, float Sw_11)
{
        self->x_00 = 0.0;
        self->x_10 = 0.0;

        self->P_00 = 3.0;
        self->P_01 = 0.0;
        self->P_10 = 0.0;
        self->P_11 = 3.0;

        self->A_00 = 1.0;
        self->A_01 = -dt;
        self->A_10 = 0.0;
        self->A_11 = 1.0;

        self->B_00 = dt;

        self->Sz_00 = Sz_00;

        self->Sw_00 = Sw_00;
        self->Sw_11 = Sw_11;
}
```

```
void pid(float rolla, float pitcha)
{
        Input _input;
        Force _force;                   //쿼드콥터 힘
        Moment _moment;                 //쿼드콥터 모멘트
        Body _body;                             //기체좌표계 변수
        Inertial _inertial;             //관성좌표계 변수
        Omega _omega;                   //각속도 모터에 속도
        Sensor _sensor;                 //센서에서 받는 값

        double enterT = 0, enterTauPhi = 0, enterTauTheta = 0, enterTauPsi = 0;
        double height;
        double degree_rad;                      //원하는 변경 각도
        double dt = 0.0025;             //성능에 따라 적분을 다르게 해주어야함

        int cnt = 0;                    //몇 번만에 원하는 각으로 가는지 체크
        int select;                             //호버링,전후좌우 모드 설정
        int flag_sensor = 1;            //센서 On
        int flag_Theta, flag_Psi, flag_Phi;
        int flag_init = 1;              //Roll Pitch 다 잡은후 Psi 잡을 때 초기화 사용

        double Pwm1, Pwm2, Pwm3, Pwm4;
        double Volt1, Volt2, Volt3, Volt4;

        double pst;                             //소요시간 확인 용
        clock_t start, end;             //소요시간 확인 용
```

```
// omega^2 값 //

_omega.w1_sq = (_input.T / (4 * Cthrust)) - (_input.tau_phi / (2 * L * sin(45 * DegToRad))) + (
_omega.w2_sq = (_input.T / (4 * Cthrust)) - (_input.tau_phi / (2 * L * sin(45 * DegToRad))) - (
_omega.w3_sq = (_input.T / (4 * Cthrust)) + (_input.tau_phi / (2 * L * sin(45 * DegToRad))) - (
_omega.w4_sq = (_input.T / (4 * Cthrust)) + (_input.tau_phi / (2 * L * sin(45 * DegToRad))) + (

// omega 값 //
_omega.w1 = sqrt(_omega.w1_sq);          //w1
_omega.w2 = sqrt(_omega.w2_sq);          //w2
_omega.w3 = sqrt(_omega.w3_sq);          //w3
_omega.w4 = sqrt(_omega.w4_sq);          //w4

if (_omega.w1>1050)
        _omega.w1 = 1050;
if (_omega.w4>1050)
        _omega.w4 = 1050;
if (_omega.w2>1050)
        _omega.w2 = 1050;
if (_omega.w3>1050)
        _omega.w3 = 1050;

////////////////// PWM /////////////////
Volt1 = _omega.w1 * 60 / (2 * PI * 690) + R / (60 / (2 * PI * 690))*Cdrag*_omega.w1*_omega.w1;
Volt2 = _omega.w2 * 60 / (2 * PI * 690) + R / (60 / (2 * PI * 690))*Cdrag*_omega.w2*_omega.w2;
Volt3 = _omega.w3 * 60 / (2 * PI * 690) + R / (60 / (2 * PI * 690))*Cdrag*_omega.w3*_omega.w3;
Volt4 = _omega.w4 * 60 / (2 * PI * 690) + R / (60 / (2 * PI * 690))*Cdrag*_omega.w4*_omega.w4;
```

```
// 관성 좌표계 //
_inertial.phiDot = (
        cos(_inertial.psi) / cos(_inertial.theta) * _body.p +
        sin(_inertial.psi) / cos(_inertial.theta) * _body.q);

_inertial.thetaDot = (
        -sin(_inertial.psi) * _body.p +
        cos(_inertial.psi) * _body.q);

_inertial.psiDot = (
        cos(_inertial.psi)*tan(_inertial.theta) * _body.p +
        sin(_inertial.psi)*tan(_inertial.theta) * _body.q +
        _body.r);

// Inertial Angular Velocity Intergral _ make angle phi, theta, psi //

if (_inertial.phiDot > _inertial.Old_phiDot)
        _inertial.phi = _inertial.phi + _inertial.Old_phiDot*dt + (_inertial.phiDot - _inertial
else
        _inertial.phi = _inertial.phi + _inertial.Old_phiDot*dt - (_inertial.Old_phiDot - _iner

if (_inertial.thetaDot > _inertial.Old_thetaDot)
        _inertial.theta = _inertial.theta + _inertial.Old_thetaDot*dt + (_inertial.thetaDot - _
else
        _inertial.theta = _inertial.theta + _inertial.Old_thetaDot*dt - (_inertial.Old_thetaDot

if (_inertial.psiDot > _inertial.Old_psiDot)
        _inertial.psi = _inertial.psi + _inertial.Old_psiDot*dt + (_inertial.psiDot - _inertial
```

```
// 물체 좌표계 _ accerlation //
_body.uDot = g*sin(_inertial.theta) + (_body.r*_body.v - _body.q*_body.w);
_body.vDot = -g*sin(_inertial.phi)*cos(_inertial.theta) + (_body.p*_body.w - _body.r*_
_body.wDot = _force.Fz / M - g*cos(_inertial.theta)*cos(_inertial.theta) + (_body.q*_b

// 몸체 속도 적분 _ make u,v,w //
if (_body.uDot >_body.Old_uDot)
        _body.u = _body.u + _body.Old_uDot*dt + (_body.uDot - _body.Old_uDot)*dt / 2;
else
        _body.u = _body.u + _body.Old_uDot*dt - (_body.Old_uDot - _body.uDot)*dt / 2;

if (_body.vDot >_body.Old_vDot)
        _body.v = _body.v + _body.Old_vDot*dt + (_body.vDot - _body.Old_vDot)*dt / 2;
else
        _body.v = _body.v + _body.Old_vDot*dt - (_body.Old_vDot - _body.vDot)*dt / 2;

if (_body.wDot >_body.Old_wDot)
        _body.w = _body.w + _body.Old_wDot*dt + (_body.wDot - _body.Old_wDot)*dt / 2;
else
        _body.w = _body.w + _body.Old_wDot*dt - (_body.Old_wDot - _body.wDot)*dt / 2;

_body.Old_uDot = _body.uDot;
_body.Old_vDot = _body.vDot;
_body.Old_wDot = _body.wDot;

// 관성 좌표계 //
_inertial.xDot = (
        cos(_inertial.psi)*cos(_inertial.theta)*_body.u +
        (cos(_inertial.psi)*sin(_inertial.theta)*sin(_inertial.phi) - sin(_inertial.ps
        (cos(_inertial.psi)*sin(_inertial.theta)*cos(_inertial.phi) + sin(_inertial.ps
        ).
```

References

1. Newton-Euler Equations Wiki
   https://en.wikipedia.org/wiki/Newton%E2%80%93Euler_equations

2. Intelligent Flight Control of an Autonomous Quadrotor
   http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.668.8759&rep=rep1&type=pdf

3. Quadrotor Dynamics and Control
   http://rwbclasses.groups.et.byu.net/lib/exe/fetch.php?media=quadrotor:beardsquadrotornotes.pdf

4. Coriolis Force Wiki
   https://en.wikipedia.org/wiki/Coriolis_force

5. Fundamentals of Aircraft Turbine Engine Control
   https://www.grc.nasa.gov/www/cdtb/aboutus/Fundamentals_of_Engine_Control.pdf