# “가상환경 기반의 드론 운동 시뮬레이터 개발”

서울과학기술대학교

한국기계연구원

# 제 출 문

한국기계연구원장 귀하

본 보고서를 “가상환경 기반의 드론 운동 시뮬레이터 개발”과제의 보고서로 제출합니다.

2020. 02. 26.

위탁연구기관명 : 서울과학기술대학교 산학협력단

위탁연구책임자 : 김동환

연 구 원 : 박성관, 한영일

목 차

# 제 1 장 가상환경 기반의 드론 운동 시뮬레이터의 개요

## 1.1 기존 시뮬레이터 분석

기존 드론 시뮬레이터에는 DRL, FPV AIR 2 등이 있다. 먼저 그림 1 좌측의 DRL는 상용화된 드론 파츠를 부착하는 것으로 무게, 속도, 관성 모멘트 등의 세부 물리 수치를 시뮬레이터에서 적용하도록 제작되었다. 그러나 그림 1 우측에서 볼 수 있듯 무게나 관성모멘텀의 직접적인 설정은 불가능하므로 직접 제작하거나 연구용으로 특수한 부품을 착용한 드론의 경우 유사한 설정을 적용할 수는 있으나 시뮬레이션을 할 때 오차가 생기는 단점이 생긴다.

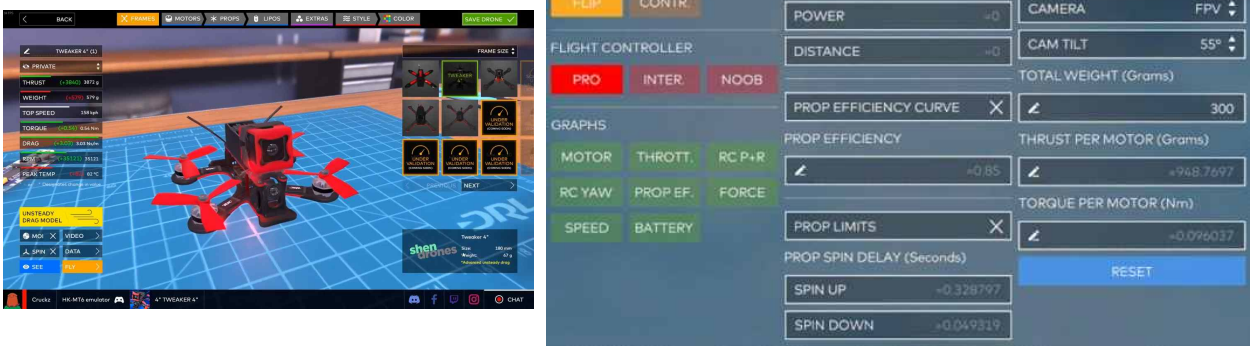


그림 1 DRL 시뮬레이터의 Drone Workbench와 Config창의 Physics 설정 페이지

다른 시뮬레이터 FPV AIR 2는 그림 2 우측 그림과 같이 크기, 추력, 공기 밀도 등의 데이터를 수정할 수 있다. 그러나 이 또한 무게, 관성모멘트 등의 중요 물리 데이터는 조종할 수 없어 상용 레이싱 드론이 아니면 시뮬레이션을 적용할 수 없게 된다.

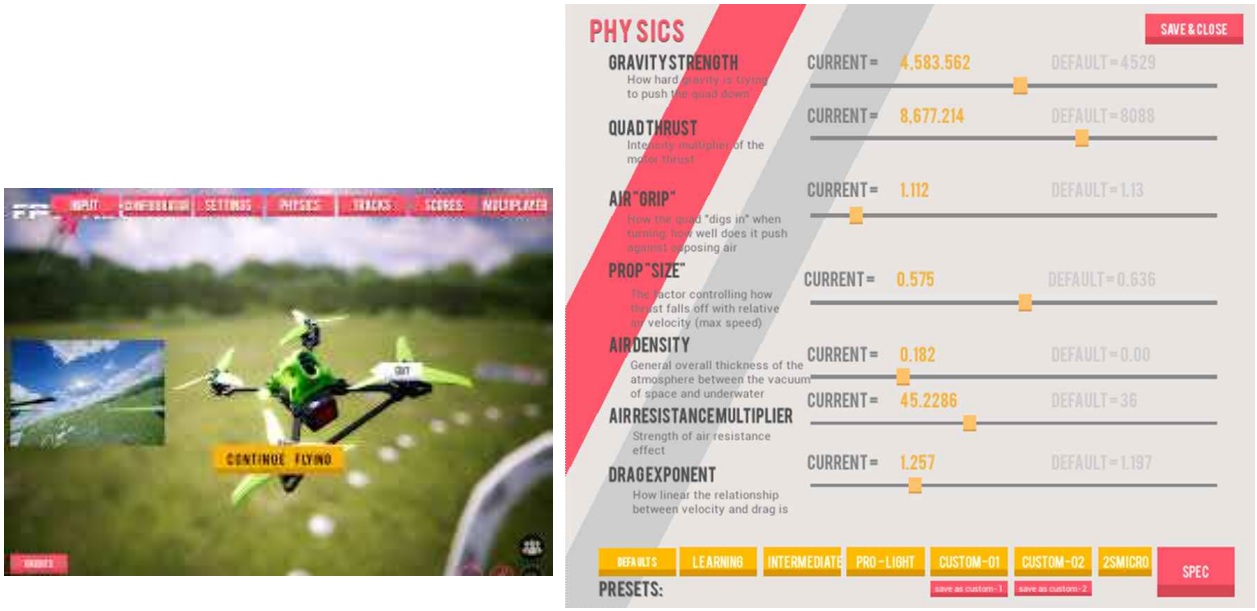


그림 2 FPV AIR 2 시뮬레이터의 시작 화면과 Physics 설정 페이지

## 1.2 기존 시뮬레이터에 비한 강점

기존 시뮬레이터의 단점을 정리하자면, 상용 드론의 부품 데이터를 가져오거나 일부 물리 수치를 조절할 수 없게 하였기 때문에 직접 만들거나 특수한 부품이 들어 있는 드론은 시뮬레이션을 할 수 없게 된다는 것이다. 그리하여 현재 제작한 Drone Simulator Program는 비행 연산에 있어 필요한 모든 입력 데이터를 수정할 수 있하였다. 그림 3의 코드는 Unity 시뮬레이터 내부 코드의 일부인데, 순서대로 드론의 무게, 지름, 항력상수, 관성모멘트, 양력상수, 감쇠상수를 외부에서 받아오는 코드이다. 아래 그림에서 볼 수 있듯, 시뮬레이터에서 원하는 대로 파라미터를 조절할 수 있고, 앞으로는 무게 중심 포인트 등 더 추가된 기능을 넣을 예정이다.

```
theIntegrator.m = UIButton.mass;
theIntegrator.l = UIButton.length;
theIntegrator.Ax = UIButton.ax;
theIntegrator.Ay = UIButton.ay;
theIntegrator.Az = UIButton.az;
theIntegrator.Ixx = UIButton.ixx;
theIntegrator.Iyy = UIButton.iyy;
theIntegrator.Izz = UIButton.izz;
theIntegrator.k = UIButton.k;
theIntegrator.b = UIButton.b;
```

그림 3 외부로부터 물리 파라미터를 받는 코드

# 제 2 장 Drone Simulator Program 제작 과정

## 2.1 논문을 통한 물리 모델 분석

쿼드콥터의 물리 모델링을 위해 Teppo Luukkonen, “Modelling and control of quadcopter”, Independent research project in applied mathematics, August 22, 2011 논문의 모델을 구현하였다. 본 논문에서는 상수인 동체 질량($m$), 지름($l$), 항력상수($A_x, A_y, A_z$), 양력상수($k$), 감쇠상수($b$), 관성모멘텀($I_{xx}, I_{yy}, I_{zz}$)을 입력하고 그림 4에서 보이는 변수인 목표 높이 $z$, Roll($\phi$), Pitch($\psi$), Yaw($\theta$)의 입력을 받아 동체의 스로틀($T$)와 토크($\tau$)로부터 각 모터의 각속도($w$)를 계산하고 미분방정식을 계산하여 속도, 각속도, 가속도, 각가속도를 출력하는 방법이 기술되어 있다. 또한 동체 좌표에서 볼 수 있듯, 동체 좌표계의 변수에는 $B$를 덧붙여 기술한다.

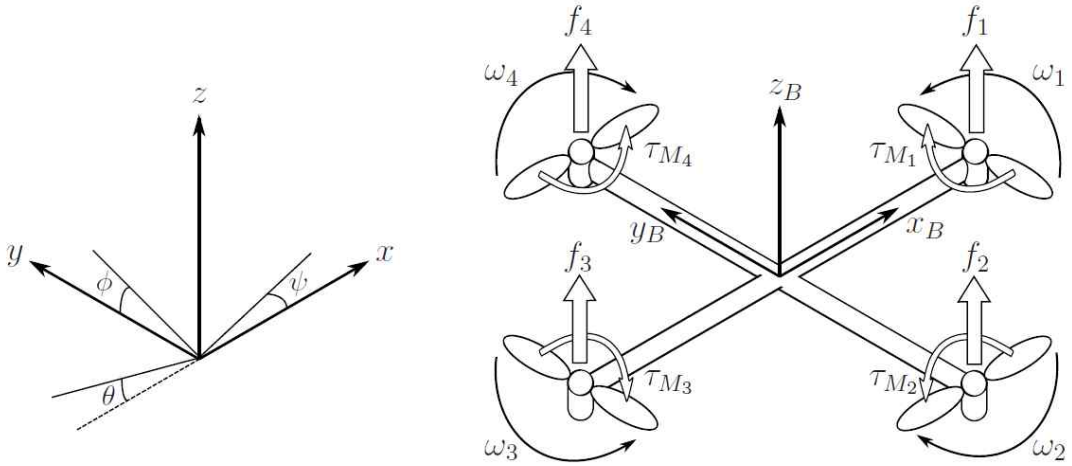


그림 4 각 드론 관성 좌표 및 동체 좌표

아래 수식은 타겟 변수 $z$, Roll($\phi$), Pitch($\psi$), Yaw($\theta$)에 도달하기 위한 $T$와 $\tau$의 PD 계산식으로, $K$는 PD 컨트롤 상수이며, $d$는 앞의 변수에 따라 타겟 높이, Roll. Pitch. Yaw를 의미한다. 또한 $S_x = \sin(x), C_x = \cos(x)$를 뜻하며 $T$는 각 프로펠러 힘의 합으로 다음 수식으로 정의된다. 또한 앞으로 각 수식의 번호는 이후 서술할 각 코드에 표시할 번호와 일치하게 된다.

$$T = (g + K_{z,D}(z'_d - z') + K_{z,P}(z_d - z))\frac{m}{C_\phi C_\theta} \qquad \text{(식 1)}$$

$$\tau_\phi = (K_{\phi,D}(\phi'_d - \phi') + K_{\phi,P}(\phi_d - \phi))I_{xx} \qquad \text{(식 2)}$$

$$\tau_\theta = (K_{\theta,D}(\theta'_d - \theta') + K_{\theta,P}(\theta_d - \theta))I_{yy} \qquad \text{(식 3)}$$

$$\tau_\psi = (K_{\psi,D}(\psi'_d - \psi') + K_{\psi,P}(\psi_d - \psi))I_{zz} \qquad \text{(식 4)}$$

아래 수식은 위 수식으로부터 모터의 각속도($w$)를 구하는 수식이며 $l$은 동체 길이를 뜻한다.

$$w_1^2 = \frac{T}{4k} - \frac{\tau_\theta}{2kl} - \frac{\tau_\psi}{4b} \quad \text{(식 5 )}$$

$$w_2^2 = \frac{T}{4k} - \frac{\tau_\phi}{2kl} + \frac{\tau_\psi}{4b} \quad \text{(식 6 )}$$

$$w_3^2 = \frac{T}{4k} + \frac{\tau_\theta}{2kl} - \frac{\tau_\psi}{4b} \quad \text{(식 7 )}$$

$$w_4^2 = \frac{T}{4k} + \frac{\tau_\phi}{2kl} + \frac{\tau_\psi}{4b} \quad \text{(식 8 )}$$

아래의 수식은 동체의 속도와 가속도를 구할 수 있는 미분방정식이다.

$$\begin{bmatrix} x'' \\ y'' \\ z'' \end{bmatrix} = -g \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \frac{T}{m} \begin{bmatrix} C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi S_\theta C_\phi - C_\psi S_\phi \\ C_\theta C_\phi \end{bmatrix} \quad \text{(식 9 )}$$

또한 아래 수식은 각속도, 각가속도를 구할 수 있는 미분방정식이며, $\eta = [\phi\, \theta\, \psi]^T$이다.

$$\eta'' = J^{-1}(\tau_B - C(\eta, \eta')\eta') \quad \text{(식 10 )}$$

위 식의 $J$는 동체 좌표의의 각속도로부터 관성 좌표의 각속도 $\eta'$으로 변환하는 자코비안 행렬이며, 다음과 같이 기술한다.

$$J = \begin{bmatrix} I_{xx} & 0 & -I_{xx} S_\theta \\ 0 & I_{yy} C_\phi^2 + I_{zz} S_\phi^2 & (I_{yy} - I_{zz}) C_\phi S_\phi C_\theta \\ -I_{xx} S_\theta & (I_{yy} - I_{zz}) C_\phi S_\phi C_\theta & I_{xx} S_\theta^2 + I_{yy} S_\phi^2 C_\theta^2 + I_{zz} C_\phi^2 C_\theta^2 \end{bmatrix} \quad \text{(식 11 )}$$

또한 $C(\eta, \eta')$는 코리올리 값을 뜻하는 행렬이며, 다음과 같이 기술한다.

$$C(\eta, \eta') = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \quad \text{(식 12 )}$$

$$C_{11} = 0$$

$$C_{12} = (I_{yy} - I_{zz})(\theta' C_\phi S_\phi + \psi' S_\phi^2 C_\theta) + (I_{zz} - I_{yy})\psi' C_\phi C_\theta - I_{xx}\psi' C_\theta$$

$$C_{13} = (I_{zz} - I_{yy})\psi' C_\phi S_\phi C_\theta^2$$

$$C_{21} = (I_{zz} - I_{yy})(\theta' C_\phi S_\phi + \psi' S_\phi C_\theta) + (I_{yy} - I_{zz})\psi' C_\phi^2 C_\theta - I_{xx}\psi' C_\theta$$

$$C_{22} = (I_{zz} - I_{yy})\phi' C_\phi S_\phi$$

$$C_{23} = -I_{xx}\psi' S_\theta C_\theta + I_{yy}\psi' S_\phi^2 S_\theta C_\theta + I_{zz}\psi' C_\phi^2 S_\theta C_\theta$$

$$C_{31} = (I_{yy} - I_{zz})\psi' C_\theta^2 S_\phi C_\phi - I_{xx}\psi'\theta' C_\theta$$

$$C_{32} = (I_{zz} - I_{yy})(\theta' C_\phi S_\phi S_\theta + \phi' S_\phi C_\theta) + (I_{yy} - I_{zz})\phi' C_\phi^2 C_\theta$$
$$+ I_{xx}\psi' S_\theta C_\theta - I_{yy}\psi' S_\phi^2 S_\theta C_\theta - I_{zz}\psi' C S_\theta C_\theta$$

$$C_{33} = (I_{yy} - I_{zz})\phi' C_\phi S_\phi C_\theta^2 - I_{yy}\theta' S_\phi^2 C_\theta S_\theta - I_{zz}\theta' C_\phi^2 C_\theta S_\theta + I_{xx}\theta' C_\theta S_\theta$$

물리 모델은 위 수식들을 사용하여 사용자가 목표 $z$, Roll($\phi$), Pitch($\psi$), Yaw($\theta$)를 넣을 시 PD 계산을 통해 얻을 수 있는 $T$와 $\tau$를 $x,y,z$와 $\phi,\theta,\psi$의 2차 미분방정식에 적용, $4^{th}$ Runge-Kutta method를 사용하여 위치, 속도, 가속도, 각도, 각속도, 각가속도를 얻을 수 있게 하였다.

## 2.2 분석한 물리 모델을 통한 Open Source에서의 구현

### 2.2.1 Matlab을 사용한 물리 모델 구현

특정 코드 뒤에 표시되는 번호는 위 논문에서 기술한 식과 동일한 코드를 나타낸 것이다.

```
% program for PID control of drone
%t0=initial time, tf=final time
clear;
t0=0;
tf=4;
%x0=initial value of state variables
x0=[0.1 0.1 0 0 0 10*pi/180 10*pi/180 10*pi/180 0 0 0]';
%tol=numerical error bound
tol=0.1;
tspan1=[t0 tf];
%Numerical analysis by Rouge-Kutta Method, return values=time t, and state
variable x
[t,X]=ode23(@(t,X) DronePIDControl_one(t,X),tspan1,x0);
subplot(2,2,1);
plot(t,X(:,1:3));
title('Subplot 1: x,y,z','FontSize',15)
legend('x','y','z')
subplot(2,2,2);
plot(t,X(:,4:6));
title('Subplot 2: vx,vy,vz','FontSize',15)
legend('vx','vy','vz')
subplot(2,2,3);
plot(t,X(:,7:9));
title('Subplot 3: roll,pitch,yaw','FontSize',15)
legend('roll','pitch','yaw')
subplot(2,2,4);
plot(t,X(:,10:12));
title('Subplot 4: vroll,vpitch,vyaw','FontSize',15)
legend('vroll','vpitch','vyaw')
```

```
%xls = [t X]
%writematrix(xls,'example.csv')
function  Xdot=DronePIDControl_one(t,X)


x=X(1,1); %x displacement
y=X(2,1); %y displacement
z=X(3,1); %z displacement
x_dot=X(4,1); %x velocity
y_dot=X(5,1); %y velocity
z_dot=X(6,1); %z velocity
phi=X(7,1);    % roll angle
theta=X(8,1); % pitch angle
psi=X(9,1);    % yawl angle
phi_dot=X(10,1);   % roll angular velocity
theta_dot=X(11,1); % pitch angular velocity
psi_dot=X(12,1);   % Yaw angular velocity

g=9.81;  % gravitational acceleration
k=2.98*10^(-6);  % thrust coefficient over angular velocity= Thrust force= k* angular velocity^2
l=0.225; % drone length
Ixx=4.856*10^-3; % x axis rotational inertia
Iyy=Ixx; % y axis rotational inertia
Izz=8.801*10^-3;  % z axis rotational inertia
m=0.468; % drone mass
b=1.14*10^-7; % damping coefficient
Ax=0.25; % x axis drag force coefficient
Ay=0.25; % y axis drag force coefficient
Az=0.25; % z axis drag force coefficient

phi_d=0*pi/180; %roll target angle (deg)
theta_d=0*pi/180; %pitch target angle
psi_d=0*pi/180; %yaw target angle
phi_d_dot=0*pi/180; %roll target angluar velocity (deg/sec)
theta_d_dot=0*pi/180; %pitch target angluar velocity
psi_d_dot=0*pi/180; %yaw target angluar velocity
z_d=0; %Attitude target (m)
z_d_dot=0; %Target Attitude velocity

K_phiD=1.75; % roll D gain
```

```
K_thetaD=1.75; % pitch D gain
K_psiD=1.75; % Yaw D gain
K_phiP=6; % roll P gain
K_thetaP=6; % pitch P gain
K_psiP=6; % Yaw P gain
K_zD=2.5; %Attitude D gain
K_zP=1.5; %Attitude P gain

T=(g+K_zD*(z_d_dot-z_dot)+K_zP*(z_d-z))*m/(cos(phi)*cos(theta));
%PID Attitude control input                                          (식 1)
tau_phi=(K_phiD*(phi_d_dot-phi_dot)+K_phiP*(phi_d-phi))*Ixx;
% Roll PID control input                                            (식 2)
tau_theta=(K_thetaD*(theta_d_dot-theta_dot)+K_thetaP*(theta_d-theta))*Iyy;
% Pitch PID control input                                           (식 3)
tau_psi=(K_psiD*(psi_d_dot-psi_dot)+K_psiP*(psi_d-psi))*Izz;
% Yaw PID control input                                             (식 4)

w1=sqrt(T/(4*k)-tau_theta/(2*k*l)-tau_psi/(4*b));
% 1st motor angular velocity (rad/sec)                              ( 5 )
w2=sqrt(T/(4*k)-tau_phi/(2*k*l)+tau_psi/(4*b));
% 2nd motor angular velocity (rad/sec)                              ( 6 )
w3=sqrt(T/(4*k)+tau_theta/(2*k*l)-tau_psi/(4*b));
% 3rd motor angular velocity (rad/sec)                              ( 7 )
w4=sqrt(T/(4*k)+tau_phi/(2*k*l)+tau_psi/(4*b));
% 4th motor angular velocity (rad/sec)                              ( 8 )

x_ddot=T/m*(cos(psi)*sin(theta)*cos(phi)+sin(psi)*sin(phi))-1/m*Ax*x_dot;
% dx^2/dt^2; x acceleration                                         ( 9 x'' )
y_ddot=T/m*(sin(psi)*sin(theta)*cos(phi)-cos(psi)*sin(phi))-1/m*Ay*y_dot;
% dy^2/dt^2; y acceleration                                         ( 9 y'' )
z_ddot=-g+T/m*(cos(theta)*cos(phi))-1/m*Az*z_dot;
% dz^2/dt^2; z acceleration                                         ( 9 z'' )

J=[Ixx 0  -Ixx*sin(theta);
   0    Iyy*cos(phi)^2+Izz*sin(phi)^2
(Iyy-Izz)*cos(phi)*sin(phi)*cos(theta);
   -Ixx*sin(theta)  (Iyy-Izz)*cos(phi)*sin(phi)*cos(theta)
Ixx*sin(theta)^2+Iyy*sin(phi)^2*cos(theta)^2+Izz*cos(phi)^2*cos(theta)^2];
                                                                    ( 11 )
tau_B=[ tau_phi; tau_theta; tau_psi]; % roll, pitch, yaw torque

eta_dot=[phi_dot; theta_dot; psi_dot];
%roll angular velocity, pitch angular velocity, Yaw angular velocity
```

```
c11=0;

c12=(Iyy-Izz)*(theta_dot*cos(phi)*sin(phi)+psi_dot*sin(phi)^2*cos(theta))+(Iz
z-Iyy)*psi_dot*cos(phi)^2*cos(theta)-Ixx*psi_dot*cos(theta);

c13=(Izz-Iyy)*psi_dot*cos(phi)*sin(phi)*cos(theta)^2;

c21=(Iyy-Izz)*(theta_dot*cos(phi)*sin(phi)+psi_dot*sin(phi)^2*cos(theta))+(Iy
y-Izz)*psi_dot*cos(phi)^2*cos(theta)+Ixx*psi_dot*cos(theta);

c22=(Izz-Iyy)*phi_dot*cos(phi)*sin(phi);

c23=-Ixx*psi_dot*sin(theta)*cos(theta)+Iyy*psi_dot*sin(phi)^2*sin(theta)*cos(
theta)+Izz*psi_dot*cos(phi)^2*sin(theta)*cos(theta);

c31=(Iyy-Izz)*psi_dot*cos(theta)^2*sin(phi)*cos(phi)-Ixx*theta_dot*cos(theta)
;

c32=(Izz-Iyy)*(theta_dot*cos(phi)*sin(phi)*sin(theta)+phi_dot*sin(phi)^2*cos(
theta))+(Iyy-Izz)*phi_dot*cos(phi)^2*cos(theta)+Ixx*psi_dot*sin(theta)*cos(th
eta)-Iyy*psi_dot*sin(phi)^2*sin(theta)*cos(theta)-Izz*psi_dot*cos(phi)^2*sin(
theta)*cos(theta);

c33=(Iyy-Izz)*phi_dot*cos(phi)*sin(phi)*cos(theta)^2-Iyy*theta_dot*sin(phi)^2
*cos(theta)*sin(theta)-Izz*theta_dot*cos(phi)^2*cos(theta)*sin(theta)+Ixx*the
ta_dot*cos(theta)*sin(theta);

C=[c11 c12 c13; c21 c22 c23; c31 c32 c33];      ( 12 )



eta_ddot=inv(J)*(tau_B-C*eta_dot);                    ( 10 [φ″,θ″,ψ″])
% roll, pitch, yaw angular acceleration
phi_ddot=eta_ddot(1); %roll angular acceleration      ( 10 φ″)
theta_ddot=eta_ddot(2); %pitch angular acceleration   ( 10 θ″)
psi_ddot=eta_ddot(3); %Yaw angular acceleration       ( 10 ψ″)

Xdot(1,1)=X(4,1);
```

```
Xdot(2,1)=X(5,1);
Xdot(3,1)=X(6,1);
Xdot(4,1)=x_ddot;
Xdot(5,1)=y_ddot;
Xdot(6,1)=z_ddot;
Xdot(7,1)=X(10,1);
Xdot(8,1)=X(11,1);
Xdot(9,1)=X(12,1);
Xdot(10,1)=phi_ddot;
Xdot(11,1)=theta_ddot;
Xdot(12,1)=psi_ddot;
w=[w1 w2 w3 w4]';
X
end
```

그림 5는 위에서 타게팅한 각도와 높이로 도달할 때 발생하는 드론의 좌표, 속도, 각도, 각속도를 표기한 전체 플롯이다.

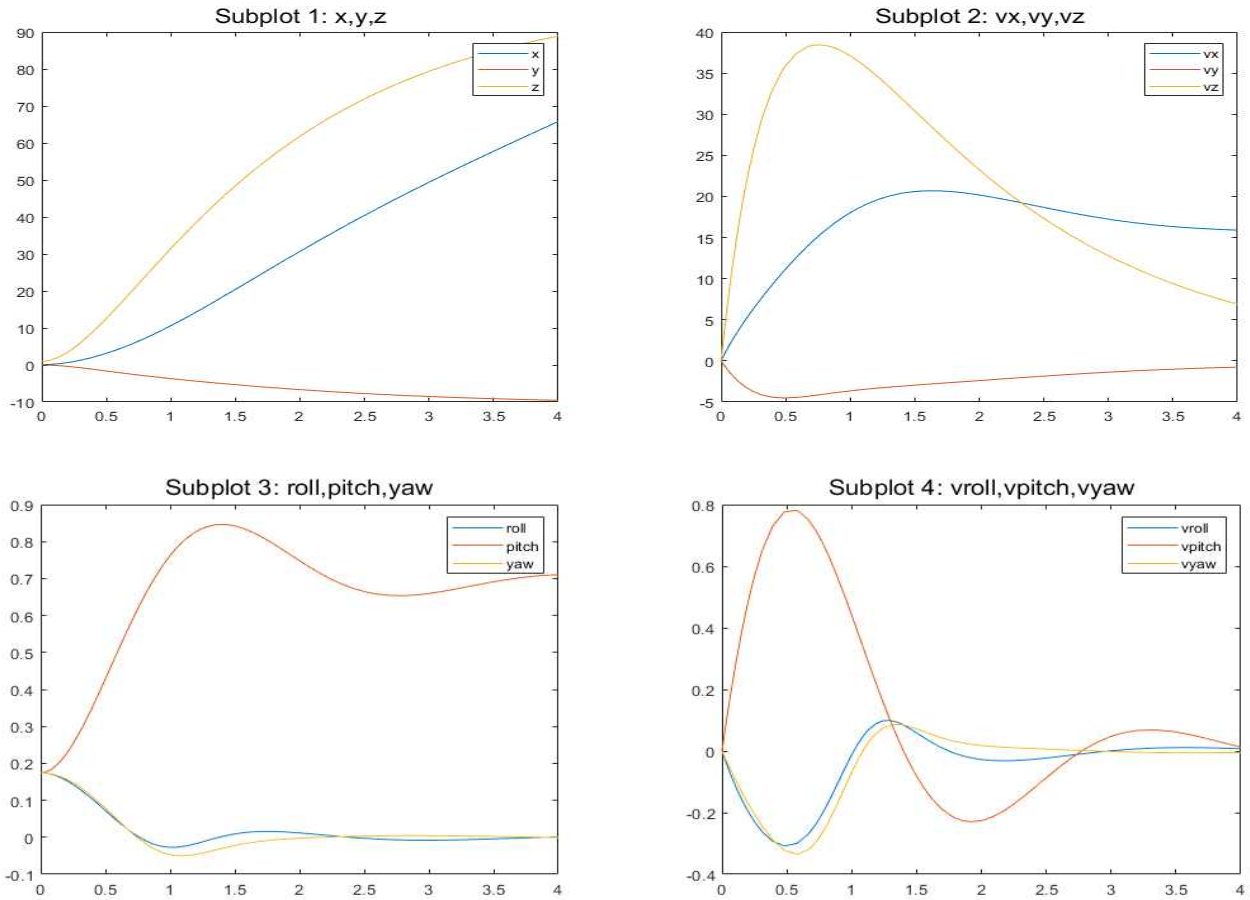


그림 5 Matlab 각도, 각속도 데이터 plot

### 2.2.2 Scilab을 사용한 물리 모델 구현

특정 코드 뒤에 표시되는 번호는 위 논문에서 기술한 식을 표현한 것이다.

```
function Xdot =DronePIDControl_one (t , X )
x =X (1 );  //x displacement
y =X (2 );  //y displacement
z =X (3 );//z displacement
x_dot =X (4 );// x velocity
y_dot =X (5 );// y velocity
z_dot =X (6 );// z velocity
phi =X (7 );  //roll angle
theta =X (8 );//pitch angle
psi =X (9 );//yaw angle
phi_dot =X (10 );  // roll angular velocity
theta_dot =X (11 );// pitch angular velocity
psi_dot =X (12 );// Yaw angular velocity

 g =9.81 ;// gravitational acceleration
k =2.98 *10 ^(-6 );// thrust coefficient over angular velocity= Thrust force= k* angular velocity^2
l =0.225 ;// drone length
Ixx =4.856 *10 ^-3 ;// x axis rotational inertia
Iyy =Ixx ;   // y axis rotational inertia
Izz =8.801 *10 ^-3 ;  // z axis rotational inertia
m =0.468 ;  // drone mass
b =1.14 *10 ^-7 ;// damping coefficient
Ax =0.25 ;  // x axis drag force coefficient
Ay =0.25 ;  // y axis drag force coefficient
Az =0.25 ;  // z axis drag force coefficient

phi_d =0 *%pi /180 ;//roll target angle (deg)
theta_d =0 *%pi /180 ;//pitch target angle
psi_d =0 *%pi /180 ;//yaw target angle
phi_d_dot =0 *%pi /180 ;  //roll target angluar velocity (deg/sec)
theta_d_dot =0 *%pi /180 ;//pitch target angluar velocity
psi_d_dot =0 *%pi /180 ;//yaw target angluar velocity
z_d =0 ;//Attitude target (m)
z_d_dot =0 ;// Target Attitude velocity

K_phiD =1.75 ;// roll D gain
K_thetaD =1.75 ;// pitch D gain
K_psiD =1.75 ;// Yaw D gain
K_phiP =6 ;// roll P gain
K_thetaP =6 ;// pitch P gain
K_psiP =6 ;// Yaw P gain
K_zD =2.5 ;//Attitude D gain
K_zP =1.5 ;//Attitude P gain

T =(g +K_zD *(z_d_dot -z_dot )+K_zP *(z_d -z ))*m /(cos (phi )*cos (theta ));      ( 1 )
//PID Attitude control input
tau_phi =(K_phiD *(phi_d_dot -phi_dot )+K_phiP *(phi_d -phi ))*Ixx ;                ( 2 )
// Roll PID control input
tau_theta =(K_thetaD *(theta_d_dot -theta_dot )+K_thetaP *(theta_d -theta ))*Iyy ;  ( 3 )
```

```
// Pitch PID control input
tau_psi =(K_psiD *(psi_d_dot -psi_dot )+K_psiP *(psi_d -psi ))*Izz ;                    ( 4 )
// Yaw PID control input


w1 =sqrt (T /(4 *k )-tau_theta /(2 *k *l )-tau_psi /(4 *b ));                          ( 5 )
// 1st motor angular velocity (rad/sec)
w2 =sqrt (T /(4 *k )-tau_phi /(2 *k *l )+tau_psi /(4 *b ));                            ( 6 )
// 2nd motor angular velocity (rad/sec)
w3 =sqrt (T /(4 *k )+tau_theta /(2 *k *l )-tau_psi /(4 *b ));                          ( 7 )
// 3rd motor angular velocity (rad/sec)
w4 =sqrt (T /(4 *k )+tau_phi /(2 *k *l )+tau_psi /(4 *b ));                            ( 8 )
// 4th motor angular velocity (rad/sec)

x_ddot =T /m *(cos (psi )*sin (theta )*cos (phi )+sin (psi )*sin (phi ))-1 /m *Ax *x_dot ;
// dx^2/dt^2; x acceleration                                                           ( 9  x'' )
y_ddot =T /m *(sin (psi )*sin (theta )*cos (phi )-cos (psi )*sin (phi ))-1 /m *Ay *y_dot ;
// dy^2/dt^2; y acceleration                                                           ( 9  y'' )
z_ddot =-g +T /m *(cos (theta )*cos (phi ))-1 /m *Az *z_dot ;
// dz^2/dt^2; z acceleration                                                           ( 9  z'' )

J =[Ixx 0  -Ixx *sin (theta );
    0     Iyy *cos (phi )^2 +Izz *sin (phi )^2    (Iyy -Izz )*cos (phi )*sin (phi )*cos (theta );
    -Ixx *sin (theta )  (Iyy -Izz )*cos (phi )*sin (phi )*cos (theta )    Ixx *sin (theta )^2 +Iyy
*sin (phi )^2 *cos (theta )^2 +Izz *cos (phi )^2 *cos (theta )^2 ];                                  (
11 )

tau_B =[tau_phi ;tau_theta ;tau_psi ];
// roll, pitch, yaw torque

eta_dot =[phi_dot ;theta_dot ;psi_dot ];
//roll angular velocity, pitch angular velocity, Yaw angular velocity
c11 =0 ;

c12 =(Iyy -Izz )*(theta_dot *cos (phi )*sin (phi )+psi_dot *sin (phi )^2 *cos (theta ))+(Izz -Iyy
)*psi_dot *cos (phi )^2 *cos (theta )-Ixx *psi_dot *cos (theta );

c13 =(Izz -Iyy )*psi_dot *cos (phi )*sin (phi )*cos (theta )^2 ;

c21 =(Iyy -Izz )*(theta_dot *cos (phi )*sin (phi )+psi_dot *sin (phi )^2 *cos (theta ))+(Iyy -Izz
)*psi_dot *cos (phi )^2 *cos (theta )+Ixx *psi_dot *cos (theta );

c22 =(Izz -Iyy )*phi_dot *cos (phi )*sin (phi );

c23 =-Ixx *psi_dot *sin (theta )*cos (theta )+Iyy *psi_dot *sin (phi )^2 *sin (theta )*cos (theta
)+Izz *psi_dot *cos (phi )^2 *sin (theta )*cos (theta );

c31 =(Iyy -Izz )*psi_dot *cos (theta )^2 *sin (phi )*cos (phi )-Ixx *theta_dot *cos (theta );

c32 =(Izz -Iyy )*(theta_dot *cos (phi )*sin (phi )*sin (theta )+phi_dot *sin (phi )^2 *cos (theta
))+(Iyy -Izz )*phi_dot *cos (phi )^2 *cos (theta )+Ixx *psi_dot *sin (theta )*cos (theta )-Iyy
*psi_dot *sin (phi )^2 *sin (theta )*cos (theta )-Izz *psi_dot *cos (phi )^2 *sin (theta )*cos
(theta );
```

```
c33 =(Iyy -Izz )*phi_dot *cos (phi )*sin (phi )*cos (theta )^2 -Iyy *theta_dot *sin (phi )^2
*cos (theta )*sin (theta )-Izz *theta_dot *cos (phi )^2 *cos (theta )*sin (theta )+Ixx *theta_dot
*cos (theta )*sin (theta );

C =[c11 c12 c13 ;c21 c22 c23 ;c31 c32 c33 ];                                    ( 12 )

eta_ddot =inv (J )*(tau_B -C *eta_dot );
// roll, pitch, yaw angular acceleration                                        ( 10 [φ'',θ'',ψ''])
phi_ddot =eta_ddot (1 );  //roll angular acceleration                           ( 10 φ'')
theta_ddot =eta_ddot (2 );//pitch angular acceleration                          ( 10 θ'')
psi_ddot =eta_ddot (3 );//Yaw angular acceleration                              ( 10 ψ'')

Xdot (1 )=X (4 );
Xdot (2 )=X (5 );
Xdot (3 )=X (6 );
Xdot (4 )=x_ddot ;
Xdot (5 )=y_ddot ;
Xdot (6 )=z_ddot ;
Xdot (7 )=X (10 );
Xdot (8 )=X (11 );
Xdot (9 )=X (12 );
Xdot (10 )=phi_ddot ;
Xdot (11 )=theta_ddot ;
Xdot (12 )=psi_ddot ;

w =[w1 w2 w3 w4 ]';
endfunction

t =0 :0.01 :5 *%pi ;
t0 =min (t );
X0 =[0.1 0.1 0 0 0 5 *%pi /180 10 *%pi /180 15 *%pi /180 0 0 0 ]';
y =ode (X0 ,t0 ,t ,DronePIDControl_one );
//Numerical analysis by Rouge-Kutta Method, return values=time t, and state variable x
subplot (2 ,2 ,1 )
xlabel ('$t ₩quad [s]$','FontSize ',3 )
ylabel ('$[m]$','FontSize ',3 )
plot (t ,y (1 ,:),'LineWidth ',2 )
plot (t ,y (2 ,:),'r ','LineWidth ',2 )
plot (t ,y (3 ,:),'g ','LineWidth ',2 )
legend (['x ';'y ';'z '],2 )

subplot (2 ,2 ,2 )
xlabel ('$t ₩quad [s]$','FontSize ',3 )
ylabel ('$[m/sec]$','FontSize ',3 )
plot (t ,y (4 ,:)*180 /%pi ,'LineWidth ',2 )
plot (t ,y (5 ,:)*180 /%pi ,'r ','LineWidth ',2 )
plot (t ,y (6 ,:)*180 /%pi ,'g ','LineWidth ',2 )
legend (['vx ';'vy ';'vz '],2 )

subplot (2 ,2 ,3 )
xlabel ('$t ₩quad [s]$','FontSize ',3 )
ylabel ('$[deg]$','FontSize ',3 )
plot (t ,y (7 ,:)*180 /%pi ,'LineWidth ',2 )
```

```
plot (t ,y (8 ,:)*180 /%pi ,'r ','LineWidth ',2 )
plot (t ,y (9 ,:)*180 /%pi ,'g ','LineWidth ',2 )
legend (['roll ';'pitch ';'yaw '],2 )

subplot (2 ,2 ,4 )
xlabel ('$t ₩quad [s]$','FontSize ',3 )
ylabel ('$[deg/sec]$','FontSize ',3 )
plot (t ,y (10 ,:)*180 /%pi ,'LineWidth ',2 )
plot (t ,y (11 ,:)*180 /%pi ,'r ','LineWidth ',2 )
plot (t ,y (12 ,:)*180 /%pi ,'g ','LineWidth ',2 )
legend (['vroll ';'vpitch ';'vyaw '],2 )
```

그림 6은 위에서 타게팅한 각도와 높이로 도달할 때 발생하는 드론의 좌표와 각도 플롯이다.

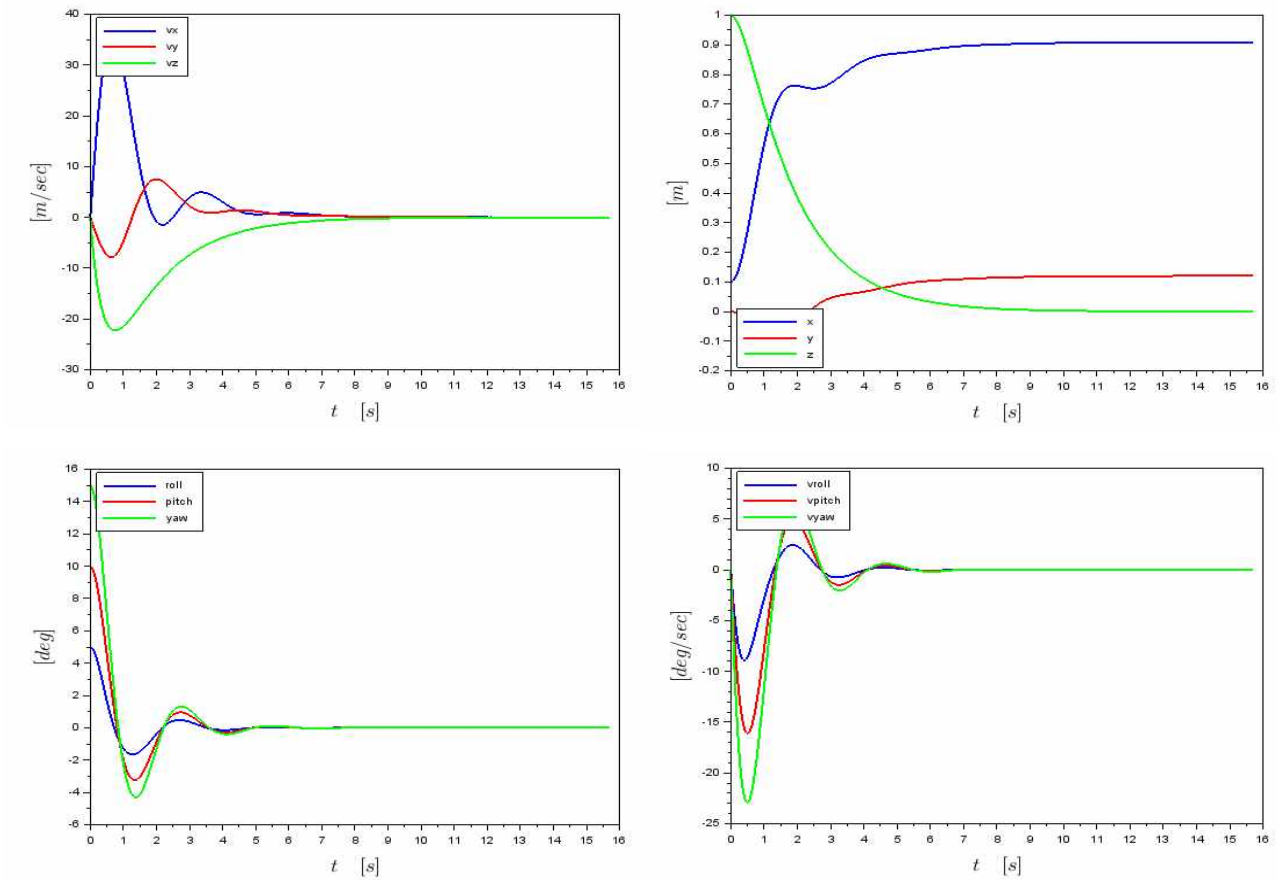


그림 6 Scilab 좌표 및 각도 plot 차트

## 2.3 드론 운동 시뮬레이터 구현

### 2.3.1 Unreal 엔진을 사용한 접근

#### 2.3.1.1 Airsim 개요

Airsim 은 Microsoft에서 개발한 Unreal Engine 4를 기반으로하는 드론, 자동차 및 기타 다양한 오브젝트를 위한 오픈 소스 크로스 플랫폼 시뮬레이터이다. 소스코드가 오픈되어있어 개발이나 수정이 용이하고 무엇보다 물리엔진이 내부에 들어있어 따로 물리엔진을 구현하지 않아도 사용이 편리하다. 이에 따라 초기 시뮬레이터 컨셉이나 환경 구축을 위하여 그림 7의 Airsim을 사용하여 여러 가지 시도를 하였다.

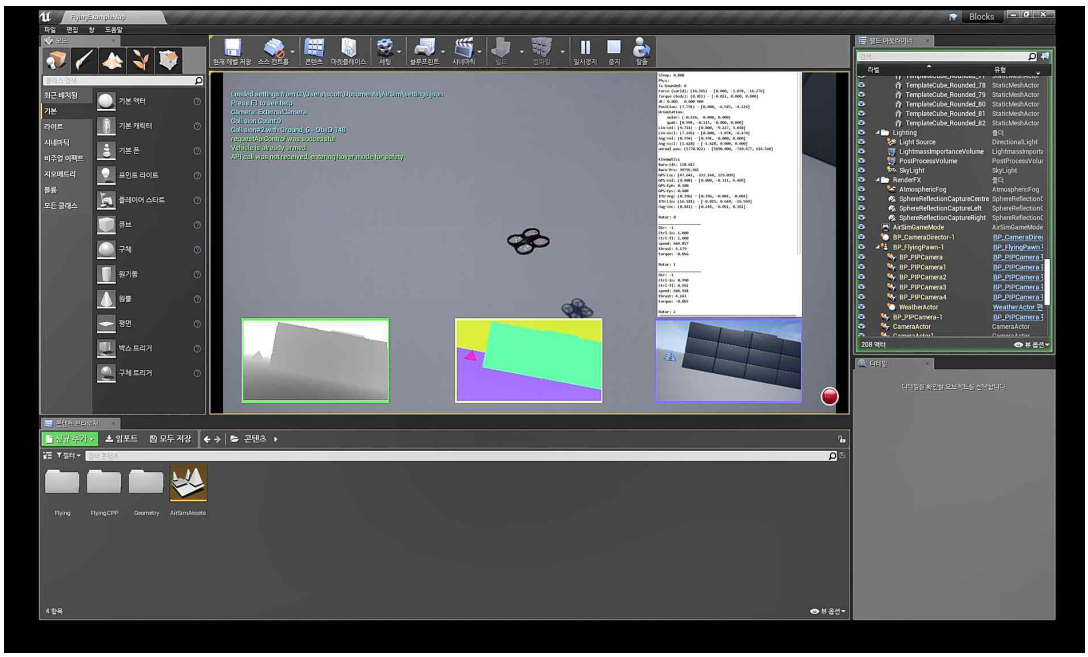

그림 7 드론 시뮬레이터 UI 화면

Airsim 의 인터페이스는 기본적으로 차량과 드론 파트로 나누어지는데 드론파트의 경우 키보드로 조종이 불가능하며, UDP통신 혹은 Pixhawk R/C를 통하여 제어가 가능하였다. 하지만 추후 Python API를 사용하여 키보드로 조종이 가능하도록 수정하였다. 또한 서브 카메라가 기본적으로 3개가 달려있으며 모든 카메라는 드론의 앞부분을 보고있으며, 메인 카메라는 드론을 3인칭 화면으로 보여준다.

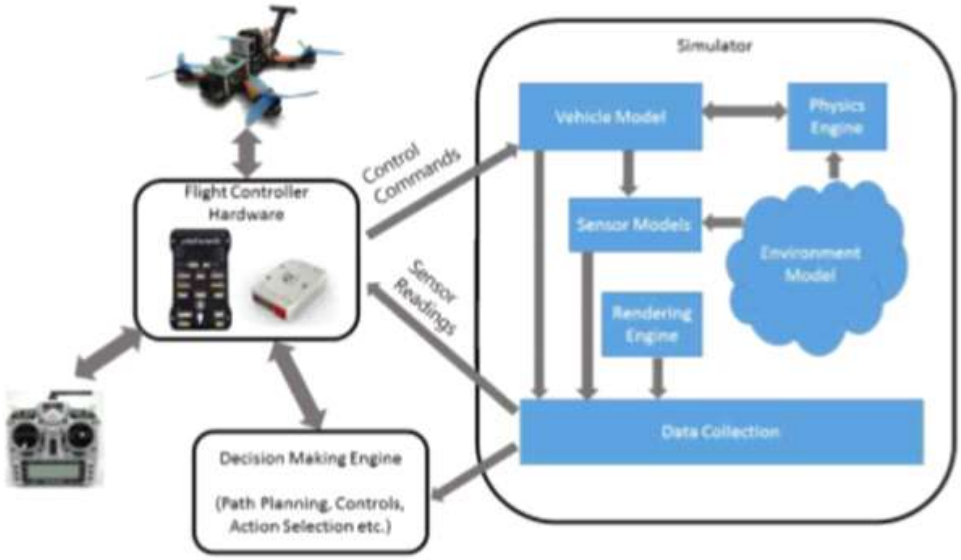


그림 8 "AirSim 구조 연구 및 예제 코드 흐름 분석" 이정현, 이웅희, 김황남 고려대학교 전기전자공학부

Pixhawk로 Airsim을 구동 시 Airsim내 HILS(Hardware-in-the-Loop simulation)를 위한 코드 구조는 그림 8과 같다.
조종기로부터 pixhawk로 컨트롤 데이터가 들어가고, 그 데이터를 차량 모델에 반영하며 물리엔진과 연동시킨다, 또한 차량모델의 위치와 각도가 변하며 이를 센서 모델에 반영하며, 카메라 등의 위치가 차량 모델에 따라 바뀌게 된다,
환경 모델의 경우 충돌이나 착륙시 물리엔진과 연동되어 충격시의 물리효과를 만들어주는 역할을 한다.

그림 9의 RC 조종기로부터 Pixhawk를 경유하여 control data를 받고 이를 Airsim 안으로 넘겨주면 이 데이터로부터 물리엔진을 통해 드론의 움직임을 구현하게 된다.
먼저 airsim HILS을 사용하기 위해 C:\Users\사용자\Documents\AirSim 폴더 내 settings.json 파일을 열어 아래와 같이 수정해 주어야 한다.

```
{
"SimMode": "",
"SettingsVersion": 1.2,
"SeeDocsAt": "https://github.com/Microsoft/AirSim/blob/master/docs/settings.md",

"Vehicles": {
 "PX4": {
  "VehicleType": "PX4Multirotor",
  "AllowAPIAlways": true,

  "LogViewerHostIp": "127.0.0.1",
  "LogViewerPort": 14388,
  "OffboardCompID": 1,
```

```
  "OffboardSysID": 134,
  "QgcHostIp": "127.0.0.1",
  "QgcPort": 14550,
  "SerialBaudRate": 115200,
  "SerialPort": "*",
  "SimCompID": 42,
  "SimSysID": 142,
  "SitlIp": "127.0.0.1",
  "SitlPort": 14556,
  "UdpIp": "127.0.0.1",
  "UdpPort": 14560,
  "UseSerial": true,
  "VehicleCompID": 1,
  "VehicleSysID": 135,
  "Model": "Generic",
  "LocalHostIp": "127.0.0.1"
  }
 }
}
```


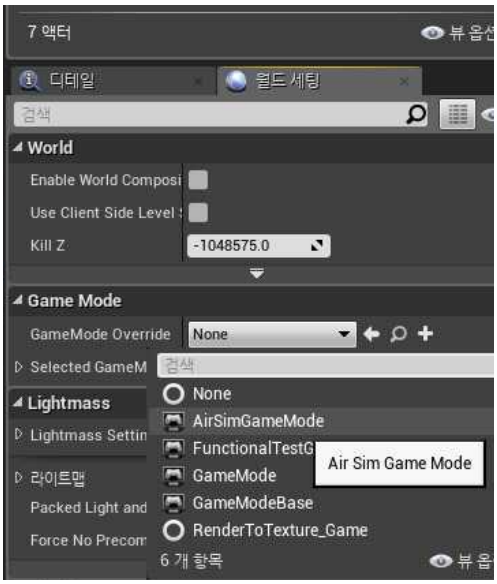


그림 9 airsim setting

airsim을 사용하기 전, airsim 플러그인을 설치하고 월드세팅의 gamemode를 airsim으로 바꿔준 후 상단의 플레이 버튼을 눌러주게 되면 그림 10과 같이 자동차나 드론으로 airsim을 실행 할 수 있게 된다.

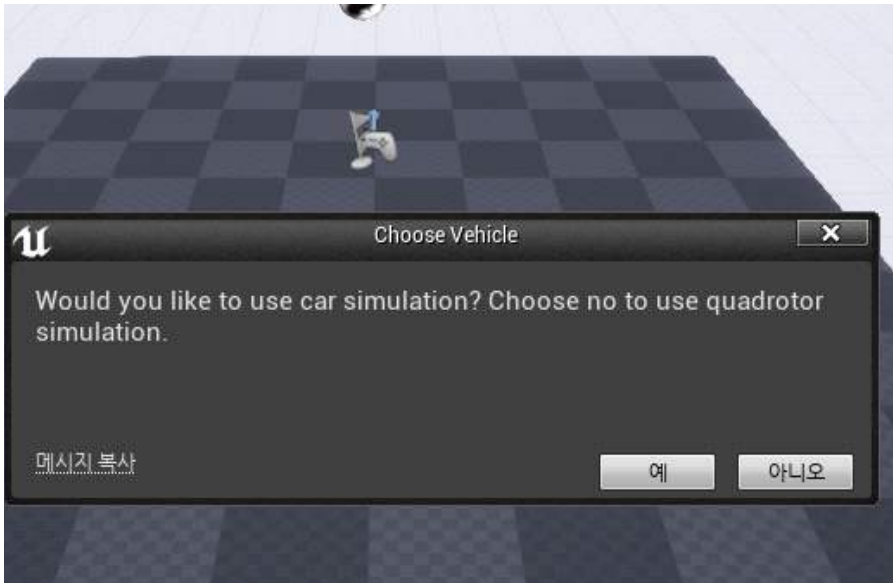


그림 10 airsim game mode

그 이후 pixhawk를 사용하여 그림 11과 같이 사용할 수 있다.

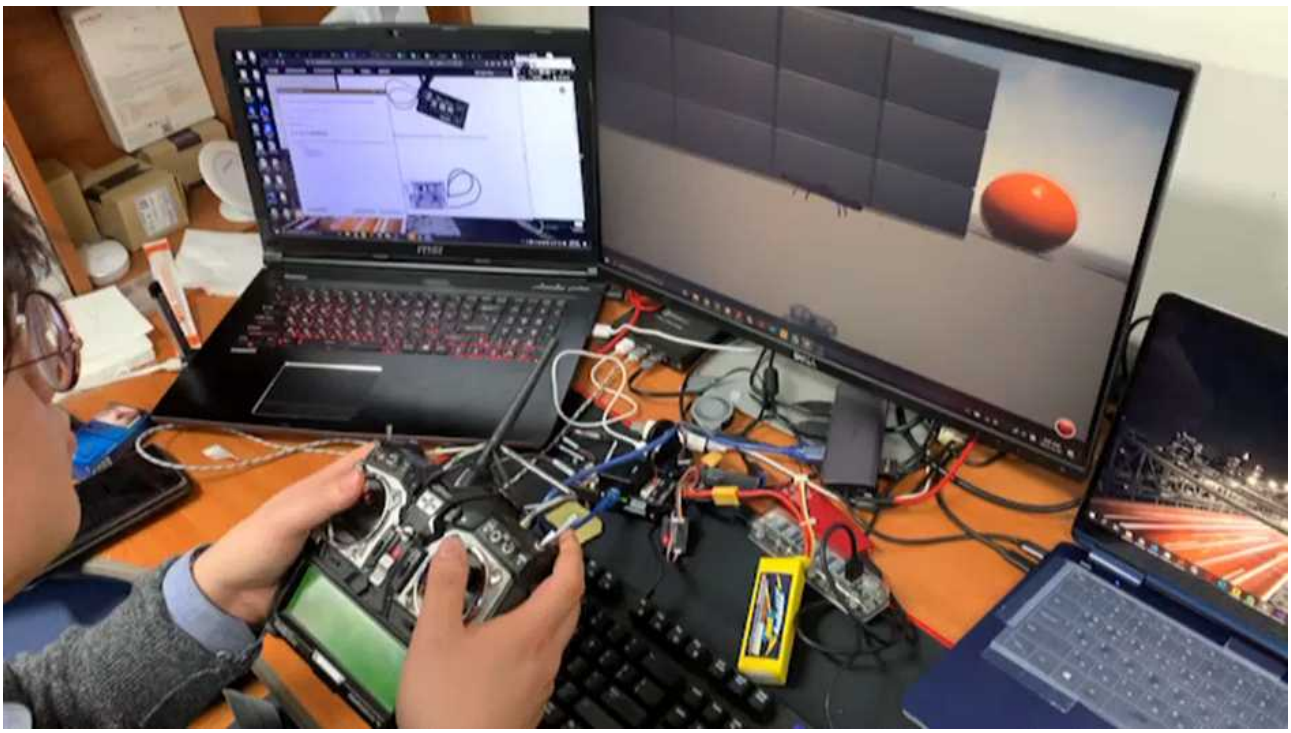

그림 11 Airsim을 Pixhawk로 구동할때의 모습

그림 8 의 개념도를 실제로 구현한 모습이다. 조종기로부터 control data를 생성하고, pixhawk로 수신하여 이를 airsim에 데이터를 전송하여 드론의 움직임을 구현한다.

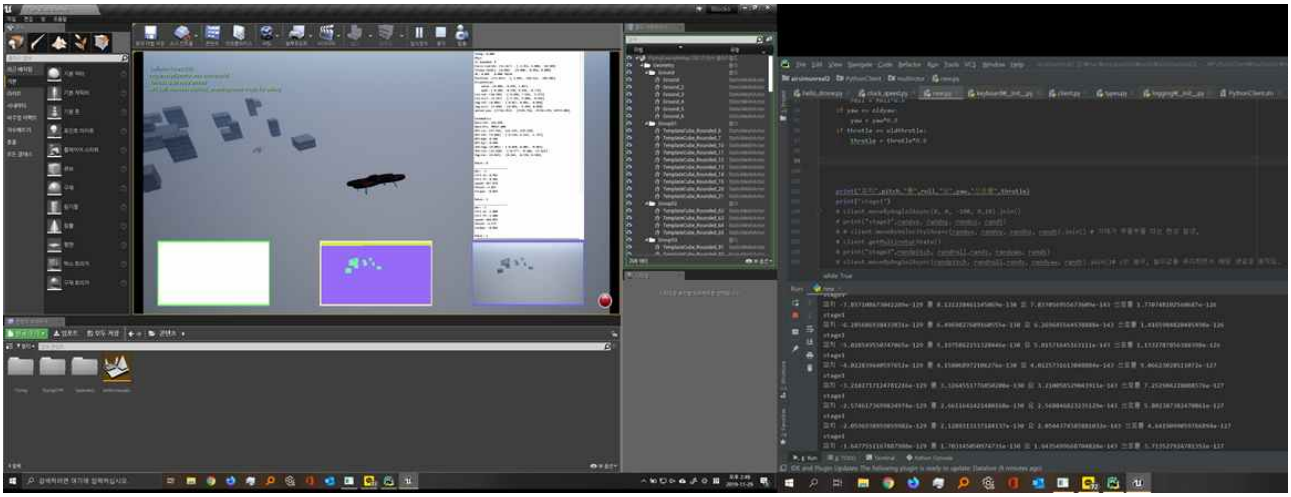

그림 12 Python API UI 화면

Airsim 에서는 그림 10과 같이 Python으로 컨트롤이 가능하도록 API를 제공하고 있으며, 이것을 통해 키보드로 드론을 조종할 수 있도록 Python을 통하여 키보드로 조종하도록 만들었다.

Python API를 사용하기 전 C:\Users\사용자\Documents\AirSim 폴더 내 settings.json 파일을 열어 아래와 같이 수정해 주어야 한다.

```
{
"SeeDocsAt": "https://github.com/Microsoft/AirSim/blob/master/docs/settings.md",
"SettingsVersion": 1.2

}
```

HILS의 경우와 다르게 따로 UDP통신이나 serial 포트를 구성해줄 필요가 없이 바로 api내에서 통신하기 때문에 위와같이 세팅 파일을 공란으로 비워주어야 한다.

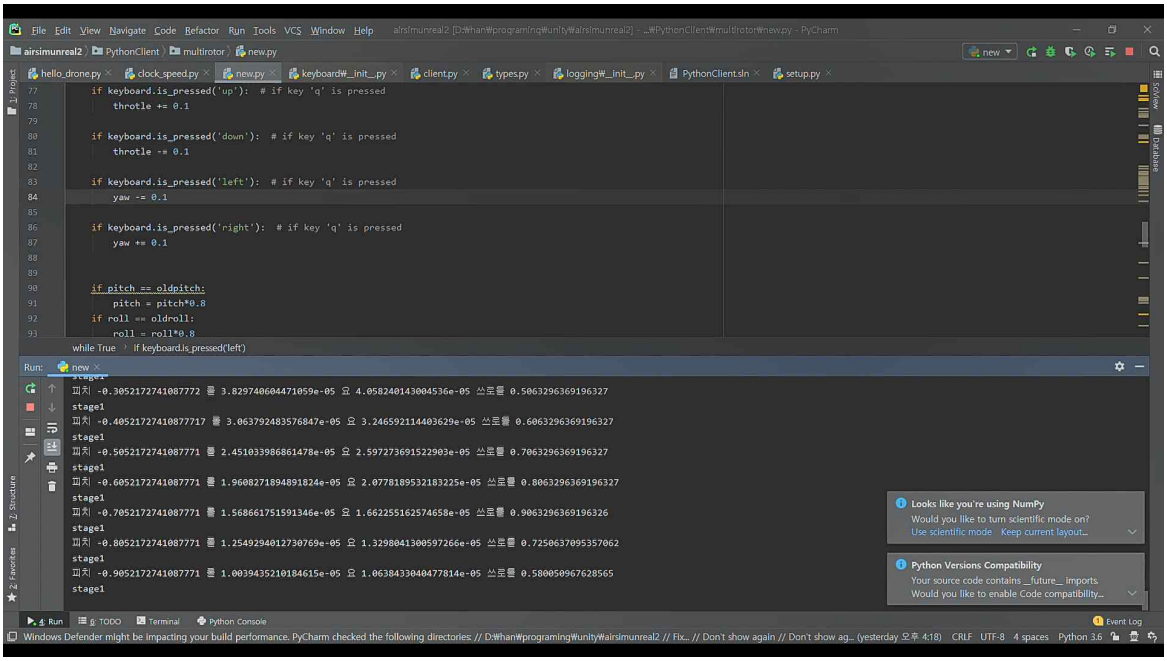


그림 13 Airsim API 구동화면

```
client = airsim.MultirotorClient()
client.confirmConnection()
client.enableApiControl(True)
client.armDisarm(True)

client.moveByVelocityZAsync(0, 0, -2, 3).join()
client.simEnableWeather(True)
client.simSetWeatherParameter(airsim.WeatherParameter.Rain, 1)

while True:
  if keyboard.is_pressed('w'):  # if key 'q' is pressed
   pitch -= 0.1
  if keyboard.is_pressed('s'):  # if key 'q' is pressed
   pitch += 0.1
  if keyboard.is_pressed('d'):  # if key 'q' is pressed
   roll += 0.1
  if keyboard.is_pressed('a'):  # if key 'q' is pressed
   roll -= 0.1
  if keyboard.is_pressed('up'):  # if key 'q' is pressed
   throtle += 0.1
  if keyboard.is_pressed('down'):  # if key 'q' is pressed
   throtle -= 0.1
  if keyboard.is_pressed('left'):  # if key 'q' is pressed
   yaw -= 0.1
  if keyboard.is_pressed('right'):  # if key 'q' is pressed
   yaw += 0.1

  if pitch == oldpitch:
   pitch = pitch*0.8
  if roll == oldroll:
   roll = roll*0.8
  if yaw == oldyaw:
   yaw = yaw*0.8
```

```
    if throtle == oldthrotle:
     throtle = throtle*0.8



    print("피치",pitch,"롤",roll,"요",yaw,"쓰로틀",throtle)

    client.moveByAngleThrottleAsync(pitch, roll, throtle, yaw, 0.1).join()
     # 앵글,쓰로틀을 통한 roll pitch yaw 제어,

    oldpitch=pitch
    oldroll=roll
    oldthrotle=throtle
    oldyaw=yaw

client.armDisarm(False)
client.reset()

client.enableApiControl(False)
```

위 코드를 통해 Python API를 사용하여 드론을 컨트롤 하였다.
while loop가 진행될때마다 키보드 눌림을 감지하여 roll pitch yaw throtle을 증가시키고 눌리지 않았을 때 안정화를 위해 각 변수들을 감소시키도록 작성되었다.
python api는 드론이나 차량 자율주행 등을 만들기 위해 작성된 것으로,
SILS(software in the loop) 방식으로 작동한다.
이를 이용하여 센서 데이터를 받아오거나 센서값에 의한 프로그램화된 컨트롤이 가능하며
본 보고서에서는 간단하게 키보드를 사용하여 드론을 움직이는 코드를 작성하였다.
사용방법은 pixhawk를 사용하지 않고 airsim을 실행 시킨 상태에서 상기 python 코드를 실행하게 될 때 udp통신을 이용하여 control data를 airsim에 전송하게 된다.

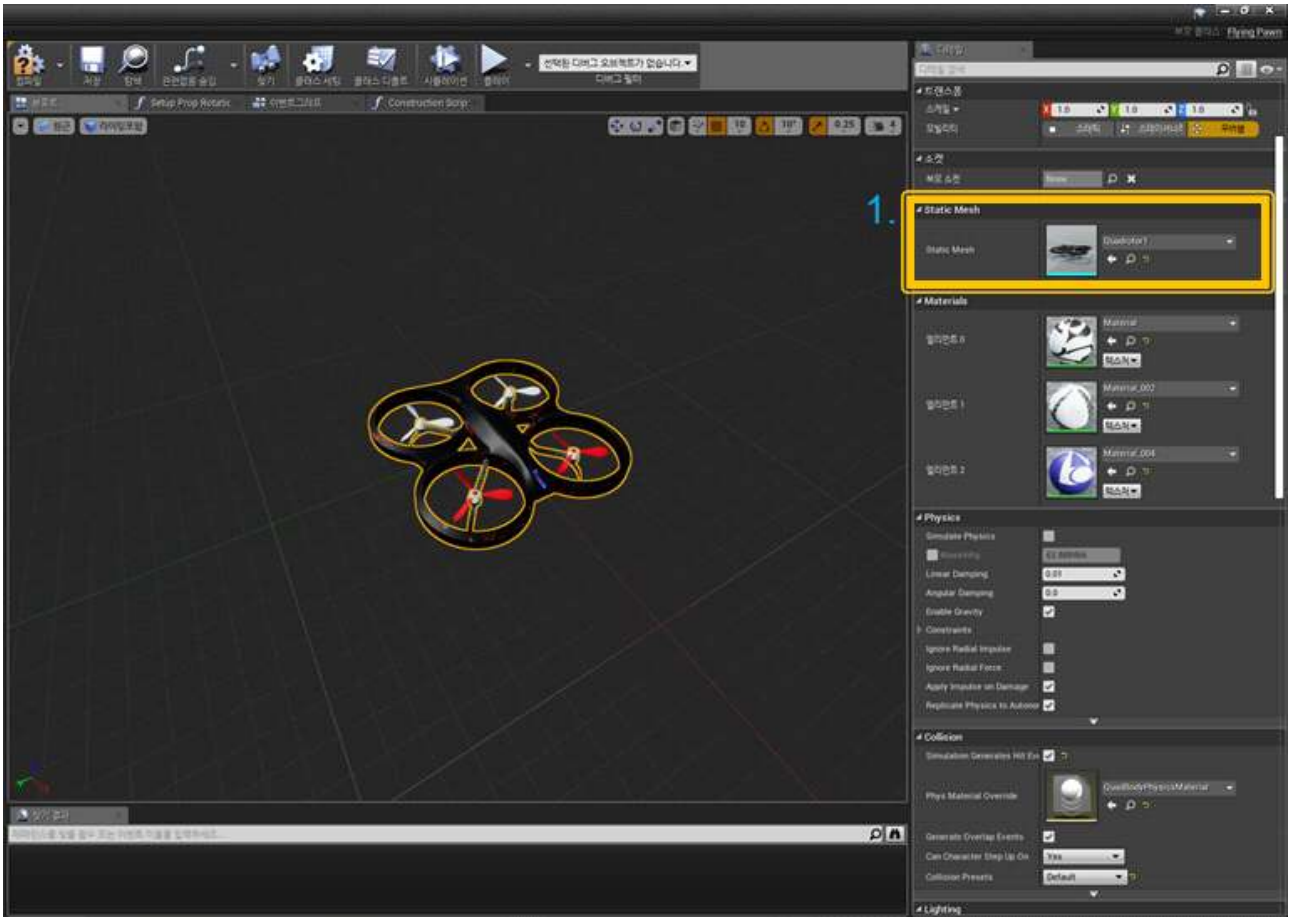


그림 14 드론 시뮬레이터 모델링 화면

또한 Unreal 에서는 그림 12와 같은 화면에서 1번 항목에 메시를 수정 혹은 교체하여 드론의 모델을 바꿀 수 있으며, 이를 통해 다양한 드론 모델로 적용 가능한 것을 확인하였다.

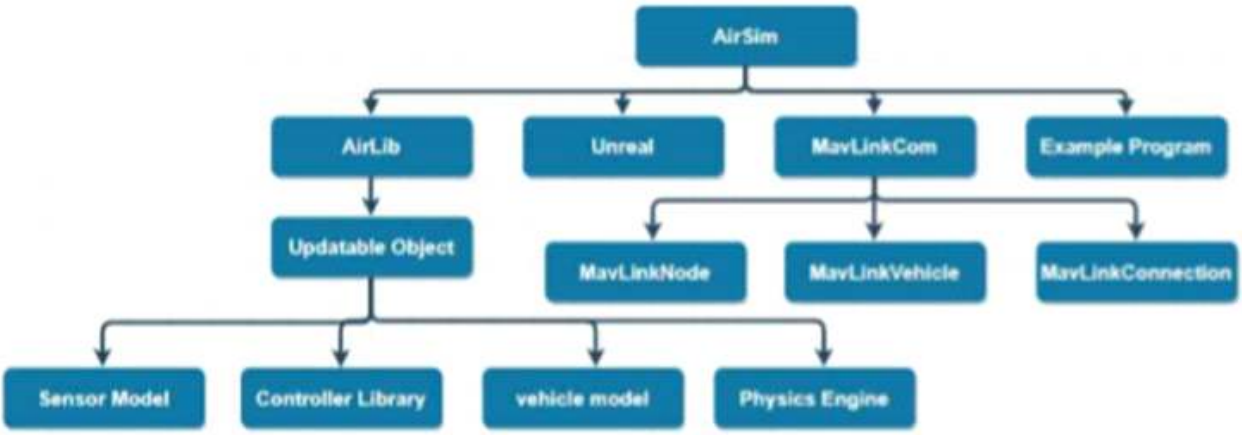


그림 15 "AirSim 구조 연구 및 예제 코드 흐름 분석" 이정현, 이웅희, 김황남 고려대학교

Airsim 코드는 그림 13과 같은 구조로 되어있으며, airlib 의 경우 airsim의 물리엔진, 비행모델, 기체 파라메터등이 포함되어 있으나, 접근하기 어려운 구조로 인하여 수정이 용이하지 않았다. 또한 mavlink의 경우 pixhawk와 통신을 위한 모듈들로 HILS(Hardware-in-the-Loop simulation) 구현을 위하여 있는 모델들이다.

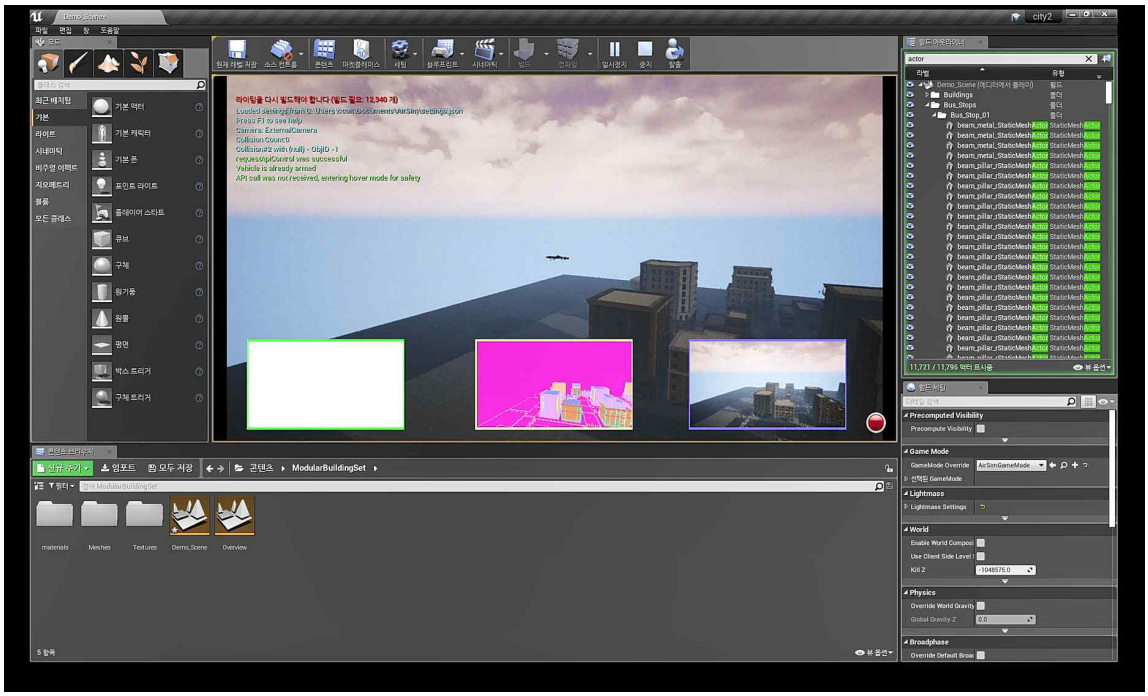

그림 16 airsim 도시환경

Airsim은 Unreal의 plugin 형태로 존재하며 환경만 조성해두고 Airsim 플러그인으로 추가하면 어느 환경에서든 시뮬레이터를 만들 수 있다. 이를 위해 그림 14와 같은 도시환경을 베이스로 한 Airsim을 구동했다.

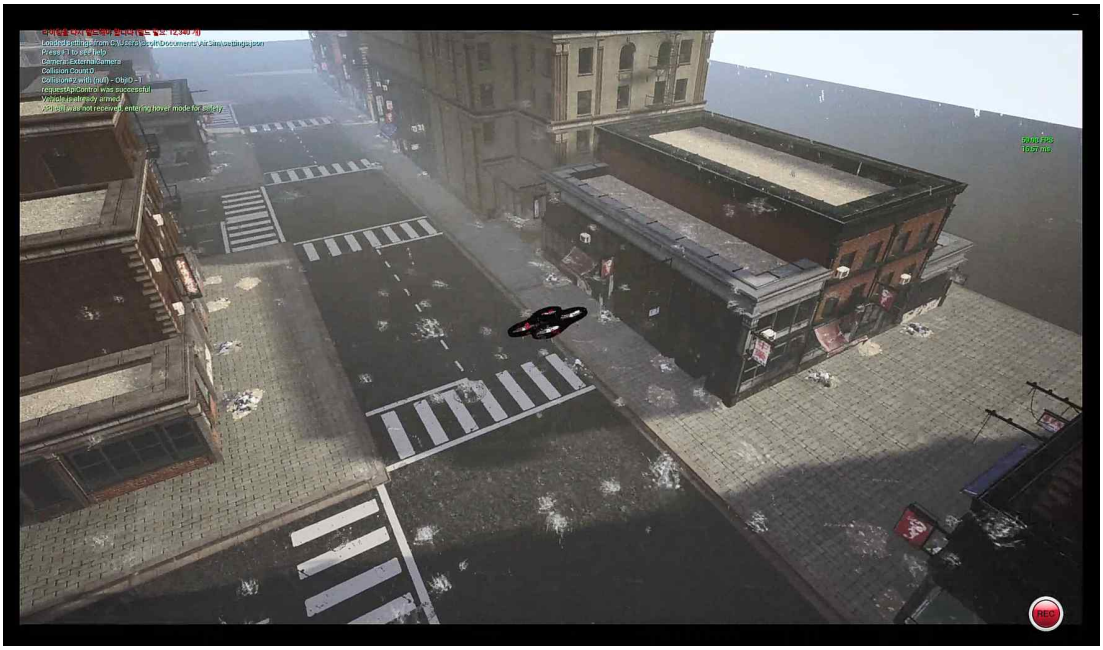

그림 17 도시 환경에서의 날씨 표현(강우)

또한 Airsim 에서는 그림 15와 같이 날씨나 바람 등을 가시화 할 수 있는데,
이는 물리엔진에는 영향을 끼치지 않았다.

### 2.3.1.2 Matlab과 Scilab으로 작성된 datatable을 Blueprint로 구현

Airsim의 경우 여러 가지 편리한 장점이 있으나, 코드의 구조가 너무 복잡하고 수정하기 어려운 단점이 있었으며 드론만을 위한 것이 아닌 차량을 위한 코드와 자율주행을 위한 센싱 코드 때문에 직관성이 떨어졌다.

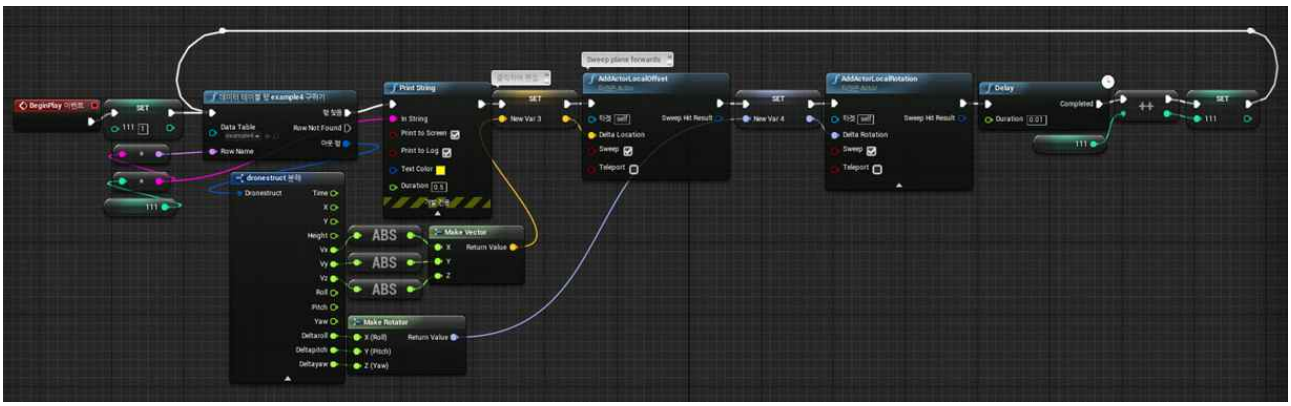


그림 18 unreal 블루프린트

이를 해결하기 위해 Matlab과 Scilab으로 작성된 시뮬레이터 결과를 Excel로 출력하여 Unreal로 표시하도록 하였다. Unreal 에서는 쉽게 따라할 수 있도록 Blueprint 라는 GUI 기반 게임 프로그래밍 툴이 있으며, 이것으로 그림 16과 같이 코드를 작성하였다. Unreal 의 경우 각 객체가 c++ 코드를 기반으로 하는데,

Blueprint는 c++코드를 블록형태로 만들어 사용자가 사용하기 쉽도록 가공해 놓은 형태이며, 이 블록들을 기반으로 graph 형태로 프로그램을 구성 가능하다.

| | time | x | y | height | vx | vy | vz | roll | pitch | yaw | deltaroll | deltapitch | deltayaw |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0.1 | 0 | 1 | 0 | 0 | 0 | 0.174533 | 0.174533 | 0.174533 | 0 | 0 | 0 |
| 2 | 5.39E-07 | 0.1 | -3.35E-12 | 1 | 1.75E-05 | -1.24E-05 | 8.00E-05 | 0.174533 | 0.174533 | 0.174533 | -6.55E-07 | 1.58E-06 | -5.21E-07 |
| 3 | 3.23E-06 | 0.1 | -1.21E-10 | 1 | 0.000105 | -7.46E-05 | 0.00048 | 0.174533 | 0.174533 | 0.174533 | -3.93E-06 | 9.49E-06 | -3.13E-06 |
| 4 | 1.67E-05 | 0.1 | -3.22E-09 | 1 | 0.000541 | -0.00039 | 0.00248 | 0.174533 | 0.174533 | 0.174533 | -2.03E-05 | 4.91E-05 | -1.62E-05 |
| 5 | 8.40E-05 | 0.1 | -8.15E-08 | 1.000001 | 0.002724 | -0.00194 | 0.012478 | 0.174533 | 0.174533 | 0.174533 | -0.0001 | 0.000247 | -8.13E-05 |
| 6 | 0.000421 | 0.100003 | -2.04E-06 | 1.000013 | 0.013629 | -0.0097 | 0.06244 | 0.174533 | 0.174533 | 0.174533 | -0.00051 | 0.001235 | -0.00041 |
| 7 | 0.002104 | 0.100072 | -5.10E-05 | 1.000328 | 0.067998 | -0.04839 | 0.311484 | 0.17453 | 0.174539 | 0.174531 | -0.00255 | 0.00617 | -0.00203 |
| 8 | 0.009278 | 0.101383 | -0.00098 | 1.006332 | 0.296802 | -0.21116 | 1.358595 | 0.174481 | 0.174659 | 0.174491 | -0.01117 | 0.027045 | -0.00891 |
| 9 | 0.016436 | 0.104311 | -0.00307 | 1.019729 | 0.520605 | -0.37012 | 2.38078 | 0.174371 | 0.174926 | 0.174403 | -0.01965 | 0.047622 | -0.01569 |
| 10 | 0.026817 | 0.111368 | -0.00808 | 1.051974 | 0.837663 | -0.59453 | 3.823991 | 0.174104 | 0.175574 | 0.17419 | -0.03173 | 0.077014 | -0.0254 |
| 11 | 0.041127 | 0.126402 | -0.01873 | 1.120497 | 1.261166 | -0.89193 | 5.739272 | 0.173533 | 0.17696 | 0.173732 | -0.04795 | 0.116639 | -0.03854 |
| 12 | 0.060037 | 0.15537 | -0.03915 | 1.251972 | 1.799054 | -1.26363 | 8.143265 | 0.172429 | 0.179649 | 0.172842 | -0.06864 | 0.167385 | -0.0555 |
| 13 | 0.084213 | 0.206853 | -0.0751 | 1.483997 | 2.455172 | -1.7037 | 11.01622 | 0.170463 | 0.184453 | 0.171245 | -0.09384 | 0.229516 | -0.07657 |
| 14 | 0.11434 | 0.292596 | -0.13402 | 1.866121 | 3.230949 | -2.19775 | 14.30051 | 0.167186 | 0.19248 | 0.168553 | -0.12329 | 0.302507 | -0.10195 |
| 15 | 0.151173 | 0.428246 | -0.2249 | 2.46047 | 4.128107 | -2.72248 | 17.90315 | 0.162026 | 0.205163 | 0.164245 | -0.15637 | 0.384884 | -0.13175 |
| 16 | 0.195619 | 0.63462 | -0.35796 | 3.342575 | 5.151805 | -3.2461 | 21.701 | 0.154267 | 0.224292 | 0.157621 | -0.19198 | 0.473998 | -0.16596 |
| 17 | 0.248868 | 0.940037 | -0.54434 | 4.60347 | 6.313564 | -3.72981 | 25.54644 | 0.143043 | 0.252047 | 0.147748 | -0.22851 | 0.565703 | -0.20429 |
| 18 | 0.312607 | 1.384679 | -0.79577 | 6.354746 | 7.633576 | -4.13027 | 29.27134 | 0.127314 | 0.291035 | 0.133373 | -0.26342 | 0.653796 | -0.24579 |
| 19 | 0.389406 | 2.029021 | -1.12471 | 8.739934 | 9.142492 | -4.4034 | 32.68668 | 0.105874 | 0.344344 | 0.112815 | -0.29257 | 0.72904 | -0.28775 |
| 20 | 0.481772 | 2.952639 | -1.53778 | 11.89849 | 10.85235 | -4.50898 | 35.53049 | 0.077966 | 0.414221 | 0.084464 | -0.30814 | 0.77686 | -0.32261 |
| 21 | 0.57987 | 4.10044 | -1.97815 | 15.4811 | 12.54084 | -4.44848 | 37.36188 | 0.047951 | 0.490979 | 0.051974 | -0.29957 | 0.781203 | -0.33485 |
| 22 | 0.6676 | 5.262092 | -2.3627 | 18.79961 | 13.93009 | -4.30994 | 38.19728 | 0.022882 | 0.558347 | 0.02309 | -0.26867 | 0.750117 | -0.3194 |
| 23 | 0.724609 | 6.080278 | -2.60527 | 20.98444 | 14.767 | -4.19855 | 38.41835 | 0.008431 | 0.600157 | 0.005526 | -0.23702 | 0.715087 | -0.29507 |
| 24 | 0.770032 | 6.765428 | -2.79385 | 22.73055 | 15.39524 | -4.1045 | 38.44454 | -0.00164 | 0.631865 | -0.00728 | -0.20592 | 0.680175 | -0.26798 |
| 25 | 0.815455 | 7.47832 | -2.97812 | 24.4752 | 15.98824 | -4.00911 | 38.35624 | -0.0102 | 0.661866 | -0.01872 | -0.17054 | 0.640021 | -0.23487 |
| 26 | 0.871759 | 8.398026 | -3.20054 | 26.62845 | 16.67319 | -3.89238 | 38.11027 | -0.01846 | 0.696356 | -0.03063 | -0.1226 | 0.584306 | -0.18737 |
| 27 | 0.93554 | 9.484362 | -3.44471 | 29.04607 | 17.38073 | -3.76552 | 37.67838 | -0.02448 | 0.73144 | -0.04071 | -0.06644 | 0.515184 | -0.12872 |
| 28 | 0.990799 | 10.46033 | -3.6499 | 31.11518 | 17.9342 | -3.66198 | 37.19564 | -0.02684 | 0.758165 | -0.0464 | -0.01982 | 0.451812 | -0.07774 |
| 29 | 1.023383 | 11.04965 | -3.76827 | 32.32197 | 18.23461 | -3.60398 | 36.87135 | -0.02707 | 0.772264 | -0.04846 | 0.005436 | 0.413517 | -0.04912 |
| 30 | 1.055968 | 11.64845 | -3.88479 | 33.51776 | 18.51587 | -3.54829 | 36.52168 | -0.02651 | 0.785109 | -0.04962 | 0.028313 | 0.374862 | -0.02239 |
| 31 | 1.094454 | 12.36705 | -4.02013 | 34.91492 | 18.82362 | -3.48541 | 36.07998 | -0.02496 | 0.798654 | -0.04992 | 0.05165 | 0.329081 | 0.00601 |

그림 19 Matlab 시뮬레이터의 output file

Matlab 시뮬레이터를 기반으로 하여 그림 17과 같이 Excel 데이터를 추출하였다. 하지만 Blueprint에서 해당 데이터를 사용하기 위하여 index열과 타이틀 행을 추가하였다.

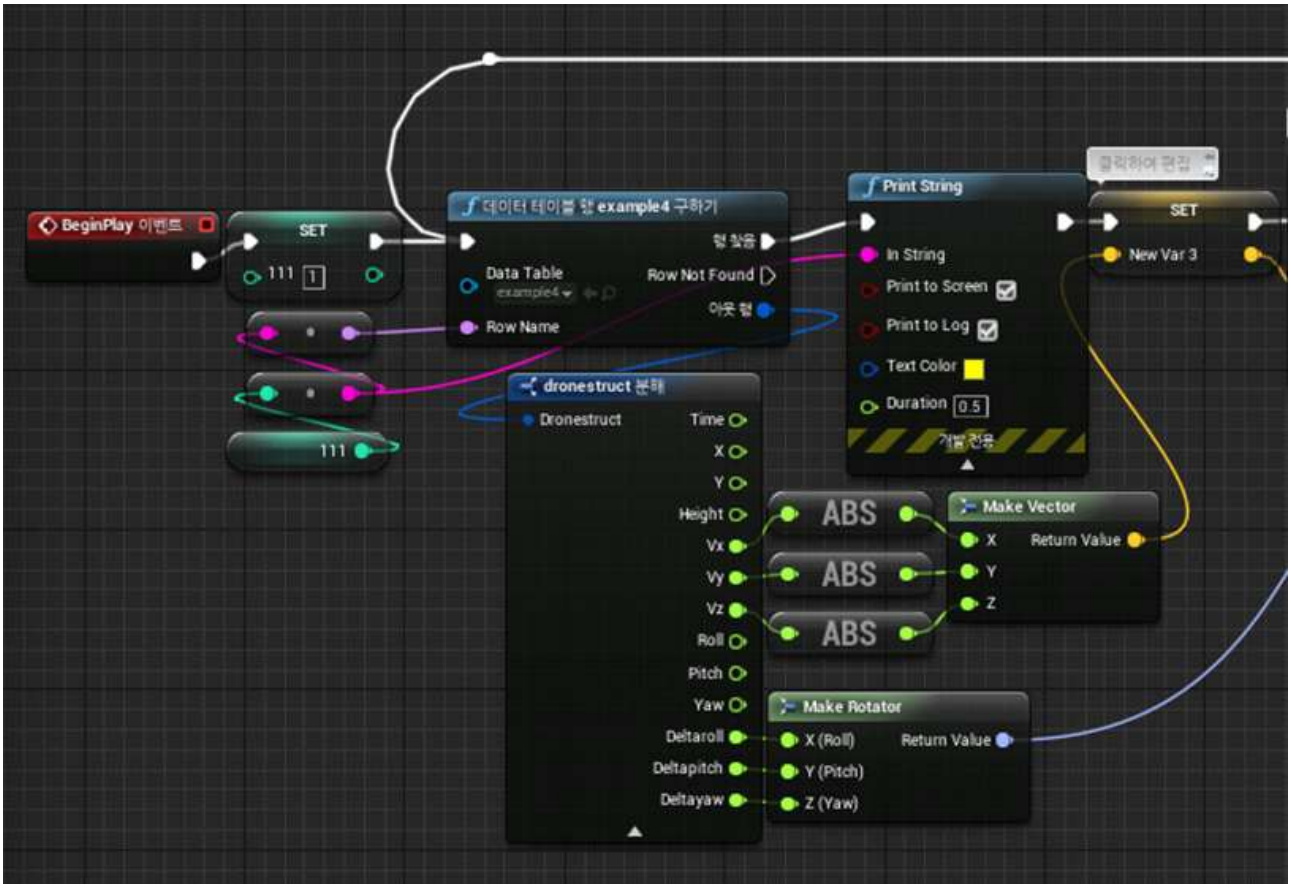


그림 20 Blueprint 상세.

프로그램 start 버튼을 누르게 되면, beginplay 이벤트가 시작되고, index변수가 선언되며 이 index 로부터 datatable의 행을 가져온다. index에 해당하는 행을 각 float 형태의 숫자로 쪼개어 속도와 각속도만을 추출하여 vector와 rotator 형태로 가공한다.

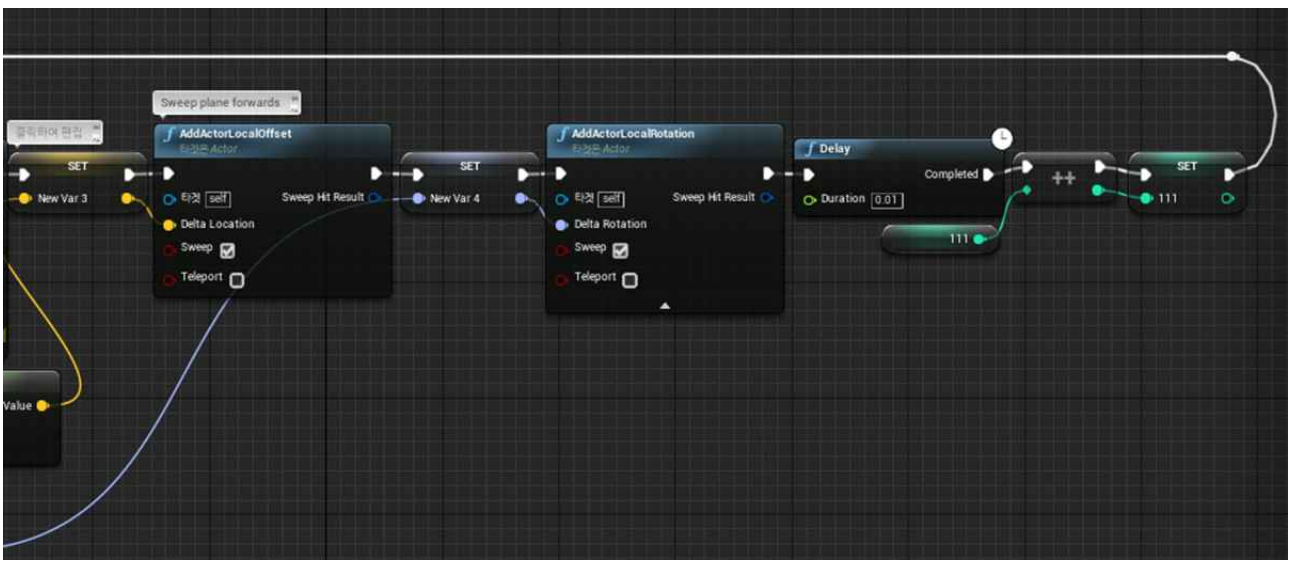


그림 21 Blueprint 상세 2

그 이후는 가공된 vector와 rotator을 드론에 반영한 후 index+1 하여 반복한다.

```
% program for PID control of drone
%t0=initial time, tf=final time
clear;
t0=0;
tf=1000;
%x0=initial value of state variables
x0=[0.1 0. 1 0 0 0 10*pi/180 10*pi/180 10*pi/180 0 0 0]';

%tol=numerical error bound
tol=0.1;
tspan1=[t0 tf/2];
tspan2=[tf/2 tf];
%Numerical analysis by Rouge-Kutta Method, return values=time t,
and state variable x
[t,X]=ode23(@(t,X) DronePIDControl_one(t,X),tspan1,x0);
[t1,X1]=ode23(@(t,X) DronePIDControl_two(t,X),tspan2,X(end,:));
t = [t;t1]
X = [X;X1]
```

DronePIDControl_one

```
phi_d=0*pi/180;
theta_d=40*pi/180;
psi_d=0*pi/180;
phi_d_dot=0*pi/180;
theta_d_dot=0*pi/180;
psi_d_dot=0*pi/180;
z_d=100;
z_d_dot=0;
```

DronePIDControl_two

```
phi_d=0*pi/180;
theta_d=40*pi/180;
psi_d=-180*pi/180;
phi_d_dot=0*pi/180;
theta_d_dot=0*pi/180;
psi_d_dot=0*pi/180;
z_d=0;
z_d_dot=0;
```

그림 22 matlab 시뮬레이터

매트랩 시뮬레이터를 통해 time span을 정해주고 시뮬레이터 함수인 pidcontrol 의 조건을 그림 20과 같이 수정하여 각각의 동작을 수행하도록 변경하여 Blueprint 로 구동하였다.

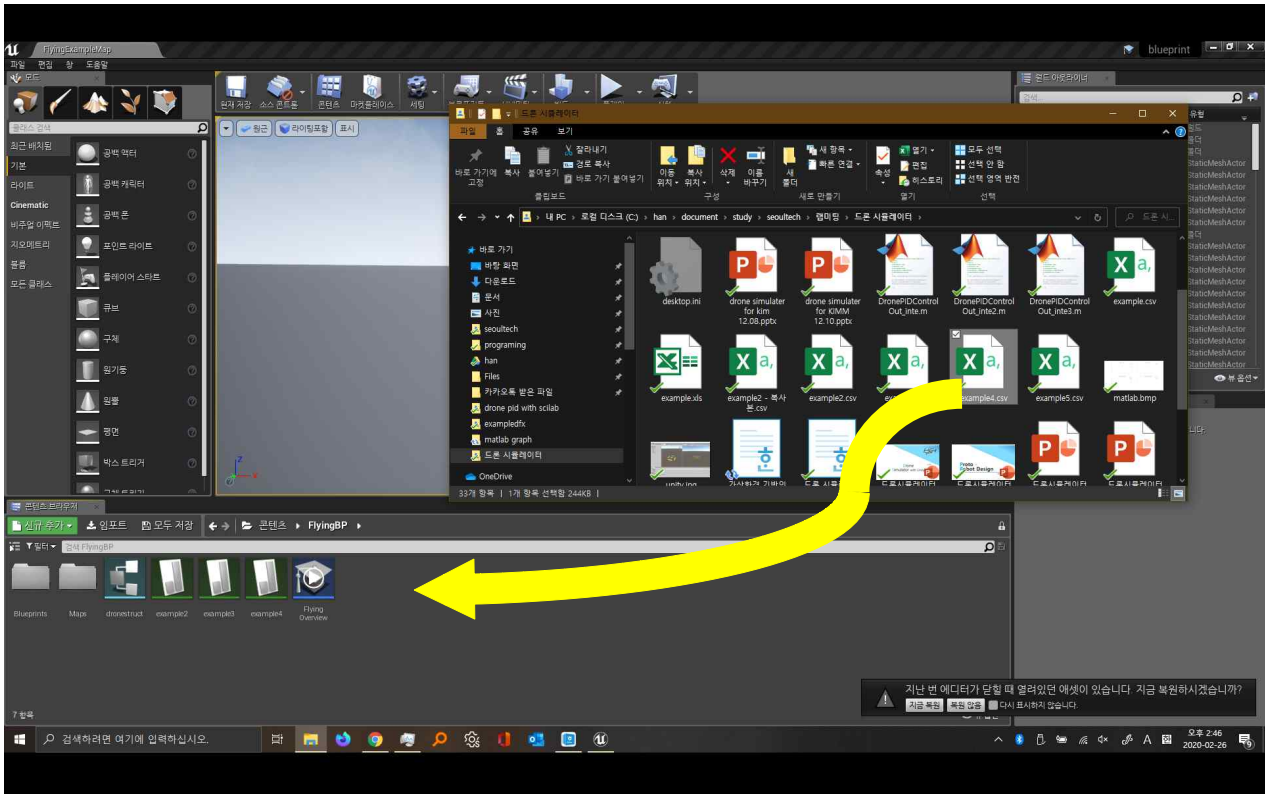

그림 23 matlab 시뮬레이터

그림 21과 같이 언리얼 프로젝트를 열고, matlab에서 생성해낸 그림 17의 데이터를 드래그 앤 드롭으로 넣어 준 후 그림 18에서 datatable을 아까 드래그 앤 드롭으로 넣은 데이터로 이름을 수정해준 후 위의 플레이


그림 24 play 버튼

위에 플레이 버튼을 눌러주게 되면 바로 시뮬레이터가 시작된다.

하지만 이 방식으로는 매번 매트랩 코드를 수정하여 데이터 테이블을 만들어 주어야 하며,
실시간 제어가 불가능한 단점이 있었고,
또한 blueprint의 경우 기초가 되는 코드 블록들을 엮어 만드는 방식이라 코드가 비대해지며,
작동 속도가 느려 적합하지 않았다.
unreal 엔진은 pawn 과 actor 등의 제어의 소유권, 메시 수정시마다 전체 환경을 새로 컴파일 해야 하는 등의
개발하기 어려운 단점이 있어 이를 회피하기 위하여 unity 엔진을 이용하여 접근하게 되었다.

### 2.3.2 Unity 엔진을 사용한 접근

#### 2.3.2.1 시뮬레이터 사용 설명서

Drone Simulator은 프로그램을 시작하면 먼저 그림 21의 UI 화면으로 진입한다. UI에서는 드론의 무게, 동체 지름, 항력계수, 관성 모멘트, 양력계수, 감쇠계수를 입력할 수 있고, 이 값들이 시뮬레이터 내부의 드론의 파라미터로 그대로 삽입되어 자기 자신의 드론이 어떻게 주행할지 시뮬레이트 할 수 있게 하였다. 화면의 상수들은 UI의 기본 물리량인데, 논문에 사용되던 기체의 스펙을 가져온 것이다.


그림 25 드론 시뮬레이터 UI 사용법

마우스와 키보드로 입력을 하거나 원래 값으로 Start 버튼을 눌러 시작을 하면 그림 22의 드론과 3D 지도가 등장한다. 처음에는 지면에 서 있다가, Throttle을 올리면 기체가 상승하게 된다. 화면을 채운 화면은 드론에 하단에 부착되어 보통 드론이 기본적으로 보는 화면이며, 좌측 하단의 화면은 드론의 현재 상태를 3인칭으로 볼 수 있게 한다.

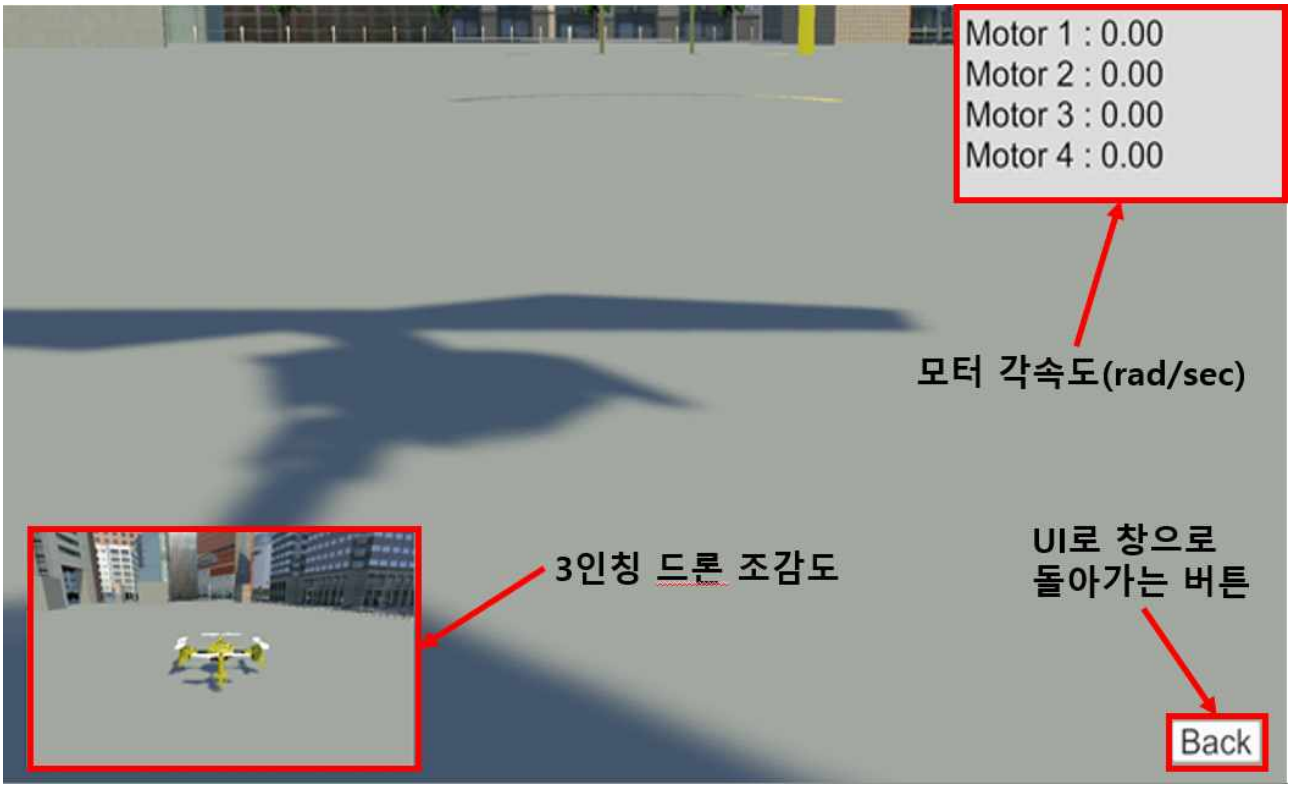


그림 26 UI에서 실행한 후 드론 시뮬레이터의 사용법


그림 27 드론의 조종 방법

드론의 움직임을 위한 입력은 XBOX one 의 조이스틱 2개를 사용하여 조종하며, 그림 23 과 같이 Throttle, Roll, Pitch, Yaw를 사용한다. 이 입력이 들어가면 목표로 하는 높이 $z$, Roll($\phi$), Pitch($\psi$), Yaw($\theta$)가 변하게 된다.

처음 Throttle을 올리는 것으로 이륙을 시작하면 논문의 물리 법칙을 따라 드론이 비행을 하게 된다. 본 시뮬레이터는 해당 상태에서 건물에 부딪히거나, 착륙을 할 때 접촉 트리거가 발생하게 되고, 이후의 물리 법칙은 논문의 비행 계산을 행하지 않고 Unity 내부의 자체 물리 엔진에 의해 그림 24와 같이 지면으로 떨어져 충돌하거나 착륙하게 된다

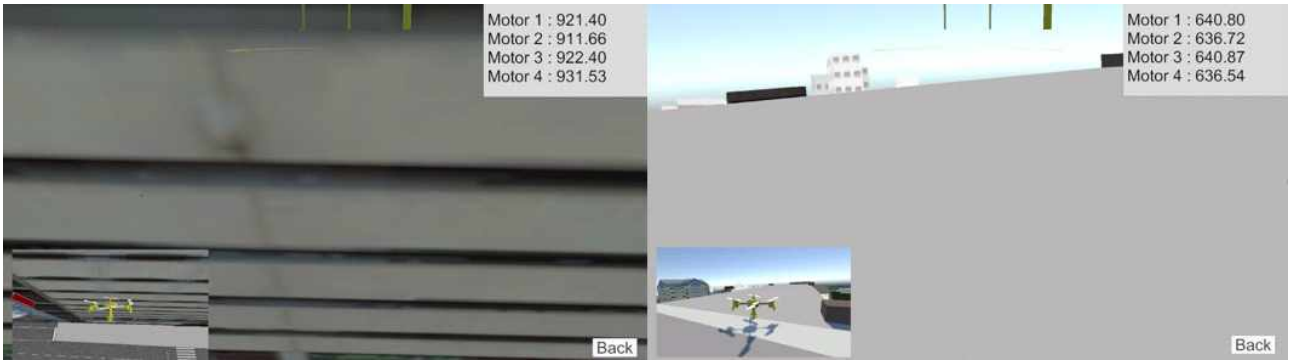


그림 28 충돌 후 낙하 상태와 착륙 상태

### 2.3.2.2 시뮬레이터 구현 및 실험 과정

이번 시뮬레이터에 사용하는 Unity3D는 3D 필드(Scene) 내에 다른 모델들을 받아 물리적 테스트를 진행한다. 또한 외부 에셋으로부터 컨텐츠를 다운로드 받아 사용할 수 있는데, 드론 시뮬레이터 내부의 드론을 구현하기 위해 드론 모델링을 Import 하였고, Unity 내부에서는 Y축과 Z축이 바뀌어 있으므로 그림 24과 같이 반전하였다.

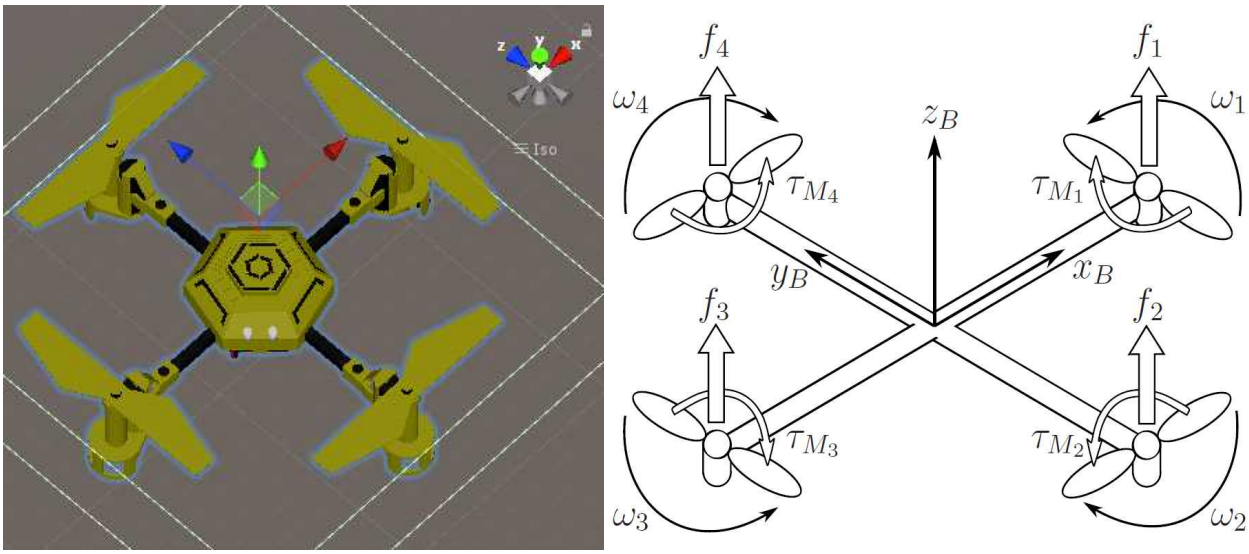


그림 29 시뮬레이터의 드론 모델링과 논문의 좌표 매칭

다음 코드는 Runge-Kutta method로 일반적으로 Unity 소프트웨어의 코드 루프 주기는 frame에 영향을 받아 프로세스에 부하가 걸릴 때 주기에 영향을 받게 되지만 하단 FixedUpdate 내부의 코드는 루프의 주기가 일정하게 되도록 정해져 있고, 동일한 주기를 사용하기 때문에 시간을 사용한 계산식의 사용이 가능하다. 그리하여 하단 코드에서는

```
void FixedUpdate()
    {
        //when we start to control drone
        if (StartTrigger)
        {
            //play runge-kutta method
            t = theIntegrator.RK4Step(theIntegrator.X, t, h);
            //get velocity
            velocity = (pos - previous) / Time.deltaTime;
            previous = pos;
            //verify position, rotation
            pos = gameObject.transform.position;
            rot = gameObject.transform.rotation.eulerAngles;
            pos.x = theIntegrator.X[0];
            pos.z = theIntegrator.X[1];
            pos.y = theIntegrator.X[2];
            rot.x = -theIntegrator.X[6] * 180 / Mathf.PI;
            rot.z = -theIntegrator.X[7] * 180 / Mathf.PI;
            rot.y = -theIntegrator.X[8] * 180 / Mathf.PI;
            rotq = Quaternion.Euler(rot.x, rot.y, rot.z);
            //verify Input
            theIntegrator.z_d += Input.GetAxis("Throttle");
            theIntegrator.phi_d = Input.GetAxis("Roll");
            theIntegrator.theta_d = Input.GetAxis("Pitch");
            theIntegrator.psi_d += Input.GetAxis("Yaw");
        }
    }
```

이 값을 입력받으면 하단의 식 (1), (2), (3), (4)를 따르는 Unity 내부의 코드를 따라 드론의 위치, 각도가 변하게 된다

```
float T = (g + K_zD * (z_d_dot - z_dot) + K_zP * (z_d - z)) * m / (Mathf.Cos(phi) * Mathf.Cos(theta)); // PID Attitude control input
        float tau_phi = (K_phiD * (phi_d_dot - phi_dot) + K_phiP * (phi_d - phi)) * Ixx; // Roll PID control input
        float tau_theta = (K_thetaD * (theta_d_dot - theta_dot) + K_thetaP * (theta_d - theta)) * Iyy;// Pitch PID control input
        float tau_psi = (K_psiD * (psi_d_dot - psi_dot) + K_psiP * (psi_d - psi)) * Izz; // Yaw PID control input
```

다음 코드는 위에 기술하였던 식(9)와 식(10)을 따르는 미분방정식을 구현하였고, X의 값을 연산하여 F에 들어가게 한다. 이 값은 상속 클래스 Integrator에 의해 Runge-Kutta 방법으로 연산되

고, 여기서 연산된 값이 드론의 위치에 그대로 적용된다.

<SHOIntegrator.cs>

```
public class SHOIntegrator : Integrator // Import Runge-Kutta
public override void RatesOfChange(float[] X, float[] F, float t)
    {
        float x = X[0];  // x displacement
        float y = X[1];  // y displacement
        float z = X[2]; // z displacement
        float x_dot = X[3]; // x velocity
        float y_dot = X[4]; // y velocity
        float z_dot = X[5]; // z velocity
        float phi = X[6];  // roll angle
        float theta = X[7]; // pitch angle
        float psi = X[8]; // yaw angle
        float phi_dot = X[9];  //roll angular velocity
        float theta_dot = X[10]; // pitch angular velocity
        float psi_dot = X[11]; // Yaw angular velocity

        float T = (g + K_zD * (z_d_dot - z_dot) + K_zP * (z_d - z)) * m / (Mathf.Cos(phi) *
Mathf.Cos(theta)); // PID Attitude control input ( 1 )
        float tau_phi = (K_phiD * (phi_d_dot - phi_dot) + K_phiP * (phi_d - phi)) * Ixx; // Roll
PID control input ( 2 )
        float tau_theta = (K_thetaD * (theta_d_dot - theta_dot) + K_thetaP * (theta_d - theta)) *
Iyy;// Pitch PID control input ( 3 )
        float tau_psi = (K_psiD * (psi_d_dot - psi_dot) + K_psiP * (psi_d - psi)) * Izz; // Yaw
PID control input ( 4 )

         a1 = Mathf.Sqrt(T / (4 * k) - tau_theta / (2 * k * l) - tau_psi / (4 * b)); // 1st motor
angular velocity(rad/ sec) ( 5 )
         a2 = Mathf.Sqrt(T / (4 * k) - tau_phi / (2 * k * l) + tau_psi / (4 * b)); // 2nd motor
angular velocity(rad/ sec) ( 6 )
         a3 = Mathf.Sqrt(T / (4 * k) + tau_theta / (2 * k * l) - tau_psi / (4 * b));// 3rd motor
angular velocity(rad/ sec) ( 7 )
         a4 = Mathf.Sqrt(T / (4 * k) + tau_phi / (2 * k * l) + tau_psi / (4 * b));// 4th motor
angular velocity(rad/ sec) ( 8 )
        if (!System.Single.IsNaN(a1)&&  !System.Single.IsNaN(a2)&&  !System.Single.IsNaN(a3)&&
!System.Single.IsNaN(a4))
        {
            w1 = a1; w2 = a2; w3 = a3; w4 = a4;
        }
        float  x_ddot  =  T  /  m  *  (Mathf.Cos(psi)  *  Mathf.Sin(theta)  *  Mathf.Cos(phi)  +
```

```
Mathf.Sin(psi) * Mathf.Sin(phi)) - 1 / m * Ax * x_dot;// +f[0]/m; // dx ^ 2 / dt ^ 2; x acceleration( 9  x'' )
        float y_ddot = T / m * (Mathf.Sin(psi) * Mathf.Sin(theta) * Mathf.Cos(phi) - Mathf.Cos(psi) * Mathf.Sin(phi)) - 1 / m * Ay * y_dot;// + f[1] / m; // dy ^ 2 / dt ^ 2; y acceleration( 9  y'' )
        float z_ddot = -g + T / m * (Mathf.Cos(theta) * Mathf.Cos(phi)) - 1 / m * Az * z_dot;// + f[2] / m; // dz ^ 2 / dt ^ 2; z acceleration( 9  z'' )

        Matrix4x4 J = Matrix4x4.identity; ( 11 )
        J[0, 0] = Ixx;
        J[0, 1] = 0;
        J[0, 2] = -Ixx * Mathf.Sin(theta);
        J[1, 0] = 0;
        J[1, 1] = Iyy * Mathf.Pow(Mathf.Cos(phi), 2) + Izz * Mathf.Pow(Mathf.Sin(phi), 2);
        J[1, 2] = (Iyy - Izz) * Mathf.Cos(phi) * Mathf.Sin(phi) * Mathf.Cos(theta);
        J[2, 0] = -Ixx * Mathf.Sin(theta);
        J[2, 1] = (Iyy - Izz) * Mathf.Cos(phi) * Mathf.Sin(phi) * Mathf.Cos(theta);
        J[2, 2] = Ixx * Mathf.Pow(Mathf.Sin(theta), 2) + Iyy * Mathf.Pow(Mathf.Sin(phi), 2) * Mathf.Pow(Mathf.Cos(theta), 2)
    + Izz * Mathf.Pow(Mathf.Cos(phi), 2) * Mathf.Pow(Mathf.Cos(theta), 2);
        Vector4 tau_B = new Vector4(tau_phi, tau_theta, tau_psi, 1);  // roll, pitch, yaw torque

        Vector4 eta_dot = new Vector4(phi_dot, theta_dot, psi_dot, 1); // roll angular velocity, pitch angular velocity, Yaw angular velocity

        //Matrix3x3 c ( 12 )
        Matrix4x4 c = Matrix4x4.identity;
        c[0, 0] = 0;
        c[0, 1] = (Iyy - Izz) * (theta_dot * Mathf.Sin(phi) * Mathf.Sin(phi) + psi_dot * Mathf.Pow(Mathf.Sin(phi), 2) * Mathf.Cos(theta)) + (Izz - Iyy) * psi_dot * Mathf.Pow(Mathf.Cos(phi), 2) * Mathf.Cos(theta) - Ixx * psi_dot * Mathf.Cos(theta);
        c[0, 2] = (Izz - Iyy) * psi_dot * Mathf.Sin(phi) * Mathf.Sin(phi) * Mathf.Pow(Mathf.Cos(theta), 2);
        c[1, 0] = (Iyy - Izz) * (theta_dot * Mathf.Cos(phi) * Mathf.Sin(phi) + psi_dot * Mathf.Pow(Mathf.Sin(phi), 2) * Mathf.Cos(theta)) + (Iyy - Izz) * psi_dot * Mathf.Pow(Mathf.Cos(phi), 2) * Mathf.Cos(theta) + Ixx * psi_dot * Mathf.Cos(theta);
        c[1, 1] = (Izz - Iyy) * phi_dot * Mathf.Sin(phi) * Mathf.Sin(phi);
        c[1, 2] = -Ixx * psi_dot * Mathf.Sin(theta) * Mathf.Cos(theta) + Iyy * psi_dot * Mathf.Pow(Mathf.Sin(phi), 2) * Mathf.Sin(theta) * Mathf.Cos(theta) + Izz * psi_dot * Mathf.Pow(Mathf.Cos(phi), 2) * Mathf.Sin(theta) * Mathf.Cos(theta);
        c[2, 0] = (Iyy - Izz) * psi_dot * Mathf.Pow(Mathf.Cos(theta), 2) * Mathf.Sin(phi) * Mathf.Cos(phi) - Ixx * theta_dot * Mathf.Cos(theta);
```

```
        c[2, 1] = (Izz - Iyy) * (theta_dot * Mathf.Cos(phi) * Mathf.Sin(phi) * Mathf.Sin(theta) + phi_dot * Mathf.Pow(Mathf.Sin(phi), 2) * Mathf.Cos(theta)) + (Iyy - Izz) * phi_dot * Mathf.Pow(Mathf.Cos(phi), 2) * Mathf.Cos(theta) + Ixx * psi_dot * Mathf.Sin(theta) * Mathf.Cos(theta) - Iyy * psi_dot * Mathf.Pow(Mathf.Cos(phi), 2) * Mathf.Cos(theta) * Mathf.Cos(theta) - Izz * psi_dot * Mathf.Pow(Mathf.Cos(phi), 2) * Mathf.Cos(theta) * Mathf.Cos(theta);
        c[2, 2] = (Iyy - Izz) * phi_dot * Mathf.Cos(phi) * Mathf.Sin(phi) * Mathf.Pow(Mathf.Cos(theta), 2) - Iyy * theta_dot * Mathf.Pow(Mathf.Sin(phi), 2) * Mathf.Cos(theta) * Mathf.Sin(theta) - Izz * theta_dot * Mathf.Pow(Mathf.Cos(phi), 2) * Mathf.Cos(theta) * Mathf.Sin(theta) + Ixx * theta_dot * Mathf.Cos(theta) * Mathf.Cos(theta);


        Vector4 eta_ddot = J.inverse * (tau_B - c * eta_dot); // roll, pitch, yaw angular acceleration
        ( 10 [φ'',θ'',ψ''])
        float phi_ddot = eta_ddot[0]+f[3]/Ixx;  // roll angular acceleration (10 φ'')
        float theta_ddot = eta_ddot[1]+f[4] / Iyy; // pitch angular acceleration (10 θ'')
        float psi_ddot = eta_ddot[2]+f[5] / Izz;// Yaw angular acceleration (10 ψ'')

        F[0] = X[3];
        F[1] = X[4];
        F[2] = X[5];
        F[3] = x_ddot;
        F[4] = y_ddot;
        F[5] = z_ddot;
        F[6] = X[9];
        F[7] = X[10];
        F[8] = X[11];
        F[9] = phi_ddot;
        F[10] = theta_ddot;
        F[11] = psi_ddot;
    }
```

위의 RateofChange 함수를 아래 <Integrator.cs>의 추상 선언을 통해 그대로 사용하는 것으로 미분방정식의 구현과 그 미분방정식을 통한 4th Runge-Kutta method 수치해석을 나누어 빠른 연산을 가능하게 했다.

<Integrator.cs>

```
abstract public void RatesOfChange(float[] x, float[] xdot, float t);


public float RK4Step(float[] x, float t, float h)
```

```
{
    RatesOfChange(x, k1, t);
    for (int i = 0; i < nEquations; i++)
    {
        store[i] = x[i] + k1[i] * h / 2.0f;
    }
    RatesOfChange(store, k2, t);
    for (int i = 0; i < nEquations; i++)
    {
        store[i] = x[i] + k2[i] * h / 2.0f;
    }
    RatesOfChange(store, k3, t);
    for (int i = 0; i < nEquations; i++)
    {
        store[i] = x[i] + k3[i] * h;
    }
    RatesOfChange(store, k4, t);
    for (int i = 0; i < nEquations; i++)
    {
        x[i] = x[i] + (k1[i] + 2.0f * k2[i] + 2.0f * k3[i] + k4[i]) * h / 6.0f;
    }
    return t + h;
}
```

그림 25는 위 연산으로 나온 데이터를 드론에 그대로 적용한 후, Matlab과 비교한 것이다. Trajectory를 비교했을 때 약간의 차이가 있다. Matlab의 4$^{th}$ Runge-Kutta method에는 좀 더 실제 값과 유사하도록 미분방정식의 연산 지점의 time 값을 조절하는 기능이 들어있는데, 이것은 그 특성에 의한 오차이다.

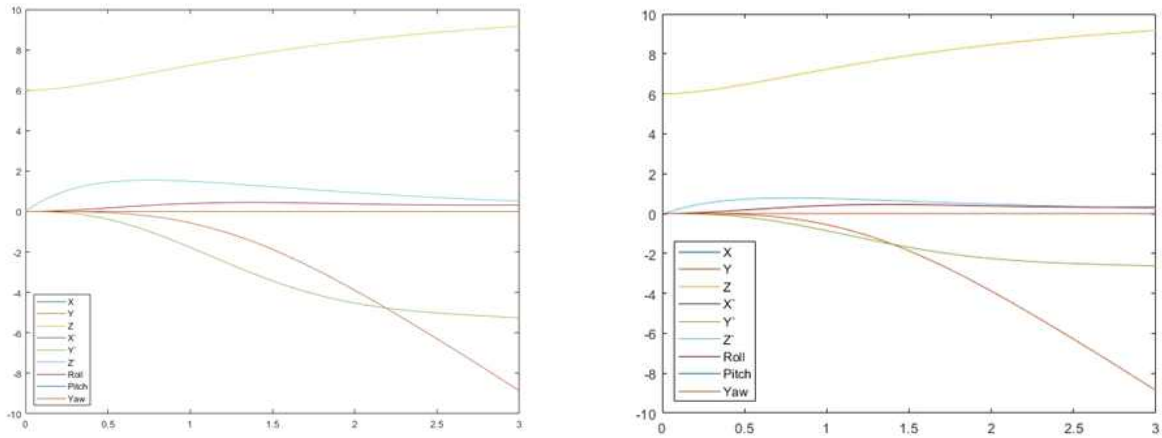


그림 30 Matlab과 Unity의 Trajectory 비교

드론의 운동 규칙을 구현한 후 Unity Asset을 통하여 그림 26과 같이 City를 제작하였다.

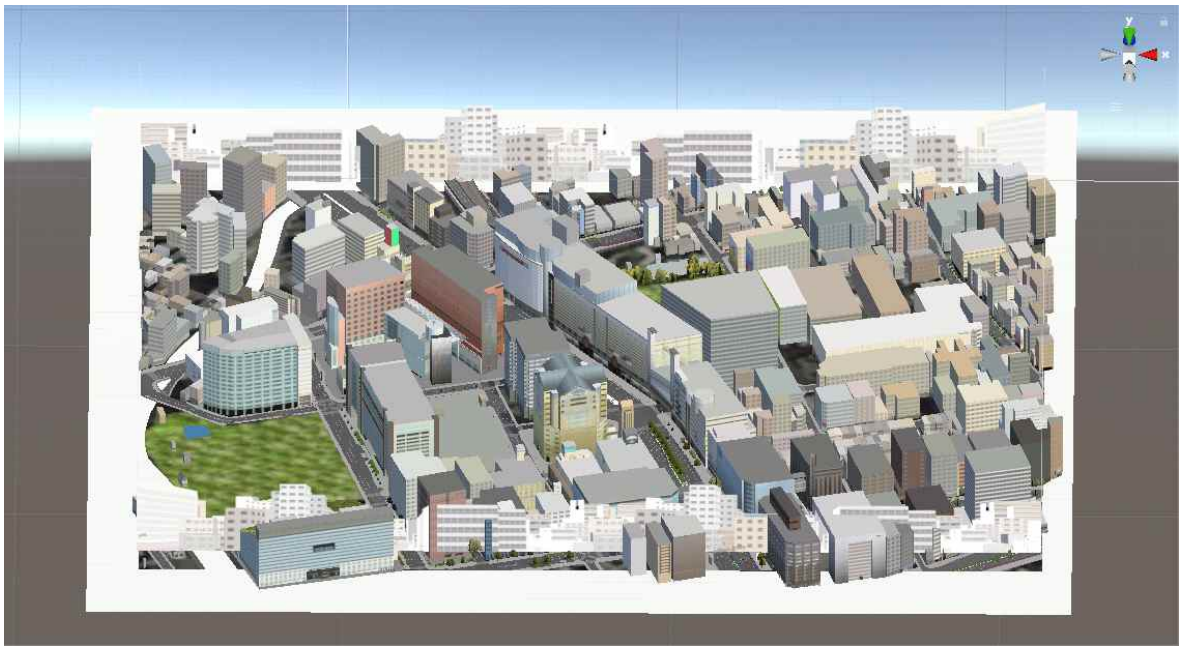

그림 31 Unity상에 구현한 도시 모델

이후 물리 충돌 모델을 구현하기 위해 그림 27과 같이 매쉬에 충돌 설정을 진행했다. 이로서 드론이 착륙하거나 충돌했을 때 Unity 상에서 물리엔진을 구동하여 자연스러운 움직임을 보일 수 있게 되었다.

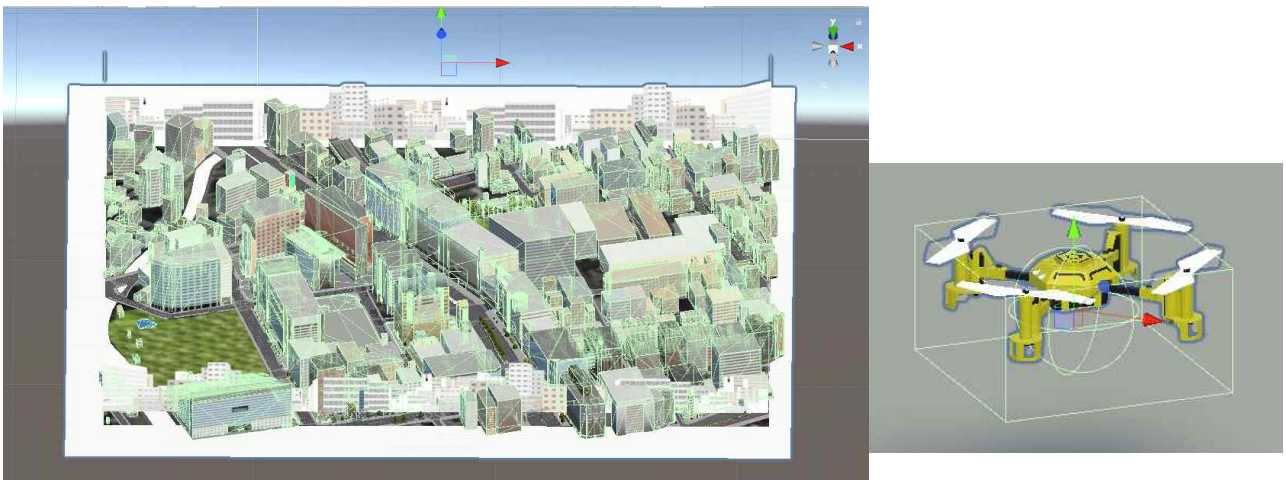

그림 32 Unity상에 구현한 물리 충돌 매쉬

# 제 3 장 결론

## 3.1 Airsim을 통한 접근시도

Airsim의 경우 여러 가지 편리한 장점이 있으나, 코드의 구조가 너무 복잡하고 수정하기 어려운 단점이 있었으며 드론만을 위한 것이 아닌 차량을 위한 코드와 자율주행을 위한 센싱 코드 때문에 직관성이 떨어졌다. 자율주행 이외의 코드를 제거하고 쉽게 시뮬레이터를 사용하기 위해 Blueprint를 사용할 생각을 하게 되었다.

## 3.2 Unreal Engine을 통한 접근시도

Unreal Engine은 사용 언어가 C++과 Blueprint인데, C++언어의 장벽으로 인해 Unreal Engine에서 사용하는 Blueprint를 사용했다. Blueprint는 배우기 쉽고 사용법이 간단하고 작업 진행 속도가 빠르다는 장점이 있다. 그러나 Matlab, Scilab 항목에서 볼 수 있는 미분방정식 등의 복잡한 코드를 구현하기 시작하니 점점 거대해져 코드를 관리하기 힘들게 되었고, 공동 작업이 매우 힘들기 때문에 언어와 엔진이 다른 Unity로 이전하게 되었다.

## 3.3 Unity를 통한 접근시도

Unity는 사용 언어가 mono와 C#인데, 이전부터 사용해왔던 C#을 통해 비교적 쉽게 미분방정식과 $4^{th}$ Runge-Kutta method를 구현할 수 있었고, 이를 베이스로 드론 모델에 스크립트를 삽입해 기본적인 드론의 파라미터 정의와 비행 모션을 구현할 수 있게 되었으며, 여기에서 확장하여 충돌, 착륙 등도 다룰 수 있게 되었다.