function results = simulate(Position, Attitude, parameters, reference, IC, noise)  
  
Position\_W = Position.Position\_W;  
Position\_G = Position.Position\_G;  
Position\_R = Position.Position\_R;  
  
Attitude\_W = Attitude.Attitude\_W;  
Attitude\_G = Attitude.Attitude\_G;  
Attitude\_R = Attitude.Attitude\_R;  
  
dt = parameters.dt;  
grav = parameters.grav;  
t\_f = parameters.t\_f;  
m = parameters.m;  
Ix = parameters.Ix;  
Iy = parameters.Iy;  
Iz = parameters.Iz;  
  
N = t\_f/dt;  
  
% Preallocating variables  
u = zeros(4,N-1); % quadcopter controls [ft tx ty tz]  
u\_noise = zeros(4,N-1); % noisy quadcopter controls  
uxyz = zeros(3,N); % output of position SNAC  
ft\_angles = zeros(3,N); % output of NN [ft phi theta] psi = 0  
torques = zeros(3,N); % output of attitude SNAC  
ft = zeros(1,N); % thrust  
angles = zeros(3,N); % angles  
angles\_ref = zeros(6,N-1); % angles and angular velocities  
Attitude\_error = zeros(6,N); % error  
r\_initial = zeros(3,N-1); % original trajectory  
r\_phi = zeros(1,N); % phi trajectory  
r\_the = zeros(1,N); % theta trajectory  
r\_psi = zeros(1,N); % psi trajectory  
  
Full\_F = @(x,grav,Ix,Iy,Iz) x + dt \* Full\_f\_225(x,grav,Ix,Iy,Iz); % discretized drift dynamics  
Full\_G = @(x,m,Ix,Iy,Iz) dt \* Full\_g\_225(x,m,Ix,Iy,Iz); % discretized control dynamics  
  
x = IC; % defining inital x as the initial condtion (IC)  
  
time = 0:dt:t\_f-dt;  
  
% determining modified reference trajectory based on original trajectory  
for i = 1:length(time)  
 r\_initial(:,i) = reference(time(i));  
end  
smooth\_r\_position = smooth(r\_initial, x(1:3)', x(4:6)', time);  
r\_smooth = [smooth\_r\_position; discrete\_deriv(smooth\_r\_position,dt)];  
  
for i = 1:N-1  
  
 % SNAC controller used to track trajectory - optimal control equation  
 uxyz(:,i) = -Position\_R^-1 \* Position\_G(x(1:6,i)-r\_smooth(:, i))' \* Position\_W(:,:)' \* Basis\_Func\_84(x(1:6,i)-r\_smooth(:, i));  
  
 % NN estimating ft and angles from ux, uy, and uz  
 ft\_angles(:, i) = NN7(uxyz(:, i));  
 ft(i) = ft\_angles(1,i);  
 if ft(i) < 0  
 ft(i) = 0;  
 end  
 angles(:,i) = [ft\_angles(2,i);ft\_angles(3,i); 0];  
  
% Numerically solving for ft and angles from ux, uy, and uz  
% [ft(i), r\_phi(i), r\_the(i)] = system\_solver(g1789(:, i),r\_psi(i),m);  
% angles(:,i) = [r\_phi(i); r\_the(i); r\_psi(i)];  
  
 angles\_ref(:,i) = [angles(:,i); deriv(angles,i,dt)];   
  
 % SNAC controller used to track angles - error regulation and optimal control equation  
 Attitude\_error(:,i) = x(7:12,i) - angles\_ref(:,i);  
 torques(:,i) = -Attitude\_R^-1 \* Attitude\_G(Attitude\_error(:,i))' \* Attitude\_W(:,:)' \* Basis\_Func\_84(Attitude\_error(:,i));  
  
 % Combining controls  
 u(:,i) = [ft(1,i); torques(:, i)];  
 u\_noise(:,i) = u(:,i) .\* (1 + noise.\*(2\*rand(size(u(:,i))) - 1)); % add randomness  
 if u\_noise(1,i) < 0  
 u\_noise(1,i) = 0;  
 end  
  
 % Passing controls though discretized drone dynamics  
 x(:, i+1) = Full\_F(x(:,i),grav,Ix,Iy,Iz) + Full\_G(x(:,i),m,Ix,Iy,Iz) \* u\_noise(:,i);  
end  
  
results.x = x;  
results.u = u\_noise;  
results.r\_smooth = r\_smooth;  
results.r\_initial = r\_initial;  
results.angles\_ref = angles\_ref;  
results.time = time;  
  
function pqr = deriv(angles, i, dt)  
if i == 1  
 pqr = ((angles(:, i) - 0)/dt);  
elseif i > 1  
 pqr = ((angles(:, i) - angles(:, i - 1))/(dt));  
end  
% Limit pqr to be between -5 and 5  
pqr(pqr > 5) = 5;  
pqr(pqr < -5) = -5;  
end  
  
function uvw = discrete\_deriv(x,dt)  
 uvw = ones(size(x));  
 uvw(:, 1) = (x(:, 2) - x(:, 1))/dt;  
 for k = 2:(size(x, 2) - 1)  
 uvw(:, k) = (x(:, k + 1) - x(:, k - 1))/(2\*dt);  
 end  
 uvw(:, end) = (x(:, end) - x(:, end - 1))/dt;  
end  
  
function [ft, r\_phi, r\_theta] = system\_solver(uxyz,r\_psi,m)  
eqns = @(vars) [uxyz(1) - vars(1)/m.\*(sin(vars(2)).\*sin(r\_psi) + cos(vars(2)).\*cos(r\_psi).\*sin(vars(3)));...  
 uxyz(2) - vars(1)/m.\*(cos(vars(2)).\*sin(r\_psi).\*sin(vars(3)) - cos(r\_psi).\*sin(vars(2)));...  
 uxyz(3) - vars(1)/m.\*(cos(vars(2)).\*cos(vars(3)))];  
  
% initial guess for the solution  
x0 = [1; pi/4; pi/6];  
  
% solve the system of equations  
options = optimset('Display','off');  
sol = fsolve(eqns, x0,options);  
  
ft = sol(1);  
r\_phi = sol(2);  
r\_theta = sol(3);  
end  
end

[*Published with MATLAB® R2022a*](https://www.mathworks.com/products/matlab)