```
function [T, X, Y, U, ctrlState] = ClosedLoopSimulation_withnoise2(tspan,x0,D,U,p, ...
    ctrlAlgorithm, simMethod, simModel, observationMethod, ctrlPar,ctrlState0,NK,↙
intensity)
%
% ClosedLoopSimulation()
%
% DESCRIPTION:
% Performs a closed-loop simulation of a model-based control algorithm for
% given time range, initial condition, disturbance variables, insulin
% levels, parameters, control algorithm, simulation model, observation
% model, control parameters, control state, control intervals and intensity
% level.
% Closed-loop is used when the input depends on the
% output; this function is part of the PID-controller.
%
% INPUT:
%   tspan           - boundaries of the control intervals              (dimension:↙
N+1   )
%   x0              - initial state                                    (dimension:↙
nx    )
%   D               - disturbance variables for each control interval  (dimension:↙
nd x N)
%   U               - Insulin levels of both bolus and basal           (dimension nu↙
x N)
%   p               - parameters                                       (dimension:↙
np   )
%   simModel        - simulation model                                 (function↙
handle)
%   ctrlAlgorithm   - control algorithm                                (function↙
handle)
%   ctrlPar         - controller parameters
%   ctrlState0      - initial controller state                         (dimension:↙
nc)
%   simMethod       - simulation method                                (function↙
handle)
%   NK              - Number of steps in each control interval         (scalar)
%   intensity       - The intensity value used for Euler Maruyama      (scalar)
%
% OUTPUT:
%   T - boundaries of control intervals (=tspan)    (dimension:      N+1)
%   X - the states in the simulation model          (dimension: nx x N+1)
%   X - the observed variables                      (dimension: ny x N+1)
%   U - the computed manipulated inputs             (dimension: nu x N  )
%   ctrlState - matrix of controller states         (dimension: nc x N+1)
%
% PROJECT:
% Fagprojekt 2022
% A diabetes case study - Meal detection
%
% GENEREL:
% BSc                       : Mathematics and technology
% University                : The Technical University of Denmark (DTU)
% Department                : Applied Mathematics and Computer Science
%
% AUTHORS:
% Emma Victoria Lind
% Mariana de Sá Madsen
% Mona Saleem
```

```
%
% CONTACT INFORMATION
% s201205@student.dtu.dk
% s191159@student.dtu.dk
% s204226@student.dtu.dk
%

% Initial time
t0 = tspan(1);

% Observed variables of glucose at steady state
y0 = observationMethod(x0,p);

% manipulated inputs calculated
uDummy = ctrlAlgorithm(y0,U(1,1), ctrlPar, ctrlState0);

% Number of each variable
nx = numel(x0);         % states
ny = numel(y0);         % glucose concentration
nu = numel(uDummy);     % manipulated inputs
nc = numel(ctrlState0); % glucose concentration and integral term

% Number of control intervals
N = numel(tspan)-1;

% Number of time steps in each control interval
Nk = NK;

% Initialising Output
T = zeros( 1, N+1);
X = zeros(nx, N+1);
Y = zeros(ny, N+1);
ctrlState = zeros(nc, N+1);

% Storing solution
T(1) = t0;
X(:,1) = x0;
Y(:,1) = y0;
ctrlState(:, 1) = ctrlState0;

% Copying initial condition to another name
tk = t0;
xk = x0;
yk = y0;

for k = 1:N

    %%%% Initializing the loop

    % Time
    tkp1 = tspan(k+1);

    % Time interval
    tspank = linspace(tk, tkp1, Nk+1);

    % Controller state
    ctrlStatek = ctrlState(:, k);
```

```
    % Disturbance variables
    dk = D(:, k);


    %%%% Start computing

    % Compute manipulated inputs after the PID control
    [uk, ctrlStatekp1] = ctrlAlgorithm(yk, U(2 ,k), ctrlPar, ctrlStatek);

    % Solving the differential equation with euler maruyama
    [Tk, Xk] = simMethod(simModel, tspank, xk, uk, dk, p,intensity);

    %%% Overwriting for the next loop

    % States at the next time step
    tkp1 = Tk(end);
    xkp1 = Xk(end,:)';

    % Observed variables at the next time step
    ykp1 = xkp1(7);

    % Update initial condition
    tk = tkp1;
    xk = xkp1;
    yk = ykp1;

    %%%% Storing solution

    % Store solution and updating conditions
    T(   k+1) = tkp1;
    X(:, k+1) = xkp1;
    Y(:, k+1) = ykp1;
    U(:, k  ) = uk;
    ctrlState(:, k+1) = ctrlStatekp1;

end
```