```
function [uk,ctrlstate] = PIDControl2(yk, U, ctrlPar, ctrlState)
%
% PIDControl()
%
% DESCRIPTION:
% This function implements a discretized proportional-integral-derivative
% (PID) controller for controlling the insulin flow rate. This differs from
% PIDControl by additionally having the insulin vector as input
%
% INPUT:
% yk         - Current blood glucose concentration
% U          - Insulin vector of both basal and bolus insulin
%
% ctrlPar    - vector of the following:
%                    * Ts              - Sampling time, 5 min
%                    * Kp              - Proportional gain
%                    * Ki              - Integrator gain
%                    * Kd              - Derivative gain
%                    * ybar            - The traget glucose concentration, y=108
%                    * ubar            - Nominal insulin flow rate
%                    * Ti              - Tuned parameters
%                    * Td              - Tuned parameters
%
% crtlState  - vector of the following:
%                    * Ik              - The integral term (Ik)
%                    * ykm1            - Previous glucose concentration, yk-1
%
% OUTPUT:
%   uk          - a vector of manipulated inputs
%   crtlstate   - the updated controller state
%
% 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
%


% Unpack control parameters
Ts      = ctrlPar(1);  %          Sampling time
Kp      = ctrlPar(2);  %          Proportional gain
%Ki      = ctrlPar(3); %          Integrator gain (not used since we calculate it in the↙
function line 68)
%Kd      = ctrlPar(4); %          Derivative gain (not used since we calculate it in the↙
```

```
function line 69)
ybar    = ctrlPar(5);  %          Target blood glucose concentration
ubar    = ctrlPar(6);  %          Nominal insulin flow rate
Ti      = ctrlPar(7);  %          Tuned parameters
Td      = ctrlPar(8);  %          Tuned parameters

% Unpack control state
Ik = ctrlState(1);    %           Value of integral at previous time step
ykm1 = ctrlState(2);  %           Previous observed glucose concentration

% Computing

ek = yk-ybar;               % Setpoint error

Ki = Kp * Ts/Ti;            % Helps controlling the steady state
Kd = Kp * Td/Ts;            % The top

Pk = Kp * ek;               % Proportional term. Controls how fast the error change

Ikp1 = Ik + Ki * ek;        % Integral term. The area of the error

Dk = Kd * (yk-ykm1);        % Derivative term. The top of the curve

uba = ubar + Pk + Ik + Dk; % Basal insulin flow rate

ubo = U;                    % Bolus insulin flow rate


% OUTPUT

% The controlled manipulated inputs at time step
uk = [uba,ubo];

% Controller state OUTPUT
ctrlstate = [Ikp1; yk];

end
```

```
function [T,X] = OpenLoopSimulation(x0, tspan, U, D, p, simModel, simMethod, NK)
% OpenLoopSimulation()
%
% DESCRIPTION:
% Function peforms an open-loop simulation for given initial condition of
% the state vector, time, intervals, disturbance variables, parameters, and
% simulation model and methods. The open-loop simulation uses the MVPmodel
% and ExplicitEuler to compute both the subcutaneous glucose concentration,
% Gsc(t), the blood glucose concentration G(t) and the statevector x(t) for
% each time step.
%
%
% INPUT:
% x0            - initial state vector                          (dimension: 7)
% tspan         - time interval to integrate over               (dimension N+1)
% U             - bolus and basal insulin (manipulated input)   (dimension nu x N)
% D             - meal rate (disturbance)                       (dimension nd x N)
% p             - parameter values                              (dimension np)
% simModel      - simulation model, MVPmodel                    (function handle)
% simMethod     - simulation method, ExplicitEuler              (function handle)
% NK            - Number of timesteps in each time interval
%
% OUTPUT:
% T - The control state of time for each step                           (dimension:↙
N+1)
% X - The statevector x(t) for each time step stored in a matrix        (dimension: nx↙
x N+1)
%
% PROJECT:
% Fagprojekt 2022
% A diabetes case study - Meal detection
%
% GENERAL:
% 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
%



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

% Number of states
nx = numel(x0);

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

```
% Allocate memory
T = zeros(1, N+1);
X = zeros(nx, N+1);

% Initial condition in each control interval
xk = x0;

% Store solution
T(1) = tspan(1);
X(:,1) = x0;

% Loop for each time step. Computes Gsc(t), G(t), x(t) from k = 0 to N.
for k = 1:N
    % Times
    tk    = tspan(k);
    tkp1  = tspan(k+1);

    % Manipulated inputs and disturbance variables
    uk = U(:,k);
    dk = D(:,k);

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

    % Solve initial value problem
    [Tk, Xk] = simMethod(simModel, tspank, xk, uk, dk, p);

    % Update initial condition
    xk = Xk(end, :)';

    % Store solution
    T(k+1) = Tk(end)';
    X(:, k+1) = Xk(end, :)';

end


end
```

```
function  [ Gfm_vec , filt_prev , flag, zero_one ] = GRID_func( ...
         delta_G , G_vec , tau, tspan , filt_prev , Gmin, Gfm_vec , t_vec, flag)
%
% GRID_func()
%
% DESCRIPTION:
% The function is a part of the GRID algortihm. This is part of the
% dectection logic, the last part of the algortihm.
% The function takes in the glucose measurements, calls the two filter
% functions and finds the derivatives. After this it is able to detect
% weather or not there a meal has been detected. Lastly, it counts down
% such that a meal will not be detected twice within two hours
%
% INPUT:
% delta_G               - The maximum ROC (rate of change)
%
% G_vec                 - Vector consisting of the glucose value and the
%                         two previous glucose measurements.
%                         As follows: [Gm-2, Gm-1, Gm].
%
% tspan                 - The interval step given as a number
%
% filt_prev             - Vector of previous filteret glucose measurements
%                         As follows: [G_{F,NS}(k-1), G_{F}(k-2)].
%                         For equation (1)&(3).
%
% Gmin                  - Vector of minumum glucose measurements
%                         As follows: [G_{min,1},G_{min,2},G_{min,3}].
%                         For equation (4).
%
% Gfm_vec               - Vector of previous derivatives
%                         As follows: [G'_{F}(k-2),G'{F}(k-2)].
%                         For equation (4).
%
% t_vec                 - Vector of sampling time respectively for G
%
% flag                  - For counting the time from last detected meal
%
% OUTPUT:
% Gfm_vec               - The stored new vector of the previous filtered
%                         glucose measurements.
%                         As follows: [G'_{F}(k-1),G'{F}(k)].
%
% G_prev                - The stored new vector of previous glucose
%                         measurements.
%                         As follows: [G_{F,NS}(k-2),G_{F}(k-2)].
%
% zero_one              - 1 or 0 for detected meal.
%
% flag                  - For counting the time from last detected meal
%
% 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
%
% REFERENCE:
% MANGLER FRA ARTIKEL
%

% Inisializing all values

Gfm_m2 = Gfm_vec(1);     % The second previous derivative used in euqation 4
Gfm_m1 = Gfm_vec(2);     % The previous derivative used in equation 4

Gfnsm2_prev = filt_prev(1); % The second previous noise-spike filtered value
                          % used in equation 1
Gfm2_prev   = filt_prev(2); % The second previous low filterd value used in
                          % equation 2

% Minimum values used in equation 4
Gmin1 = Gmin(1);
Gmin2 = Gmin(2);
Gmin3 = Gmin(3);

% The two previous measured gluscose values and the one at control state
Gm2     = G_vec(1);              % The second previous glucose value
Gm1     = G_vec(2);              % The previous glucose value
G       = G_vec(3);              % The glucose value at control state


% COMPUTING

% The noise-spike filter at the 3 sampling times
Gfnsm2 = spikefilt_func(Gm2,Gfnsm2_prev,delta_G);
Gfnsm1 = spikefilt_func(Gm1,Gfnsm2,delta_G);
Gfns   = spikefilt_func(G,Gfnsm1,delta_G);

% The low filter at the 3 sampling times
Gfm2 = lowfilt_func(tau,tspan,Gfnsm2,Gfm2_prev);
Gfm1 = lowfilt_func(tau,tspan,Gfnsm1,Gfm2);
Gf   = lowfilt_func(tau,tspan,Gfns,Gfm1);

% Inisializing input for estimate_lagrange
Gf_vec = [ Gfm2 , Gfm1 , Gf ];

% Computing the first derivative using lagrange
Gfm    = estimate_lagrange(t_vec,Gf_vec); % Returns the derivative

% The detection part from equation 4
if (Gf > Gmin1) && ...
   ((Gfm > Gmin3) && (Gfm_m1 > Gmin3) && (Gfm_m2 > Gmin3) ...
```

```
   || (Gfm > Gmin2) && (Gfm_m2 > Gmin2) )

    zero_one = 1; % A meal has been detected

else

    zero_one = 0; % No meal has been detected

end

% Output
filt_prev = [Gfnsm2,Gfm2]; % Outputting the updated filtered values
Gfm_vec = [Gfm_m1,Gfm]; % Outputting the updated derivative values


% COUNTING PART

if flag > 0

 % flag larger than 0 means that a meal has been detected within 120 min
 % implying that a meal should not be detected again already. So zero_one
 % is set to 0. Therefore, flag is subtracted by -1, such that it will
 % count down so a meal can be detetected again after 120 min.

    flag = flag-1;
    zero_one = 0;

elseif flag == 0 && zero_one == 1

  % flag equal to zero means a meal may be detected again
  % but only is if zero_one equals 1. When this happen flag start over
  % counting down 120 min.

    flag = 120/tspan;

end


end
```

```
function [truepositive,falsepositive] = evaluationfunction(stride,D,D_detected,Ts,N)
%
% detectionrates()
%
% DESCRIPTION:
% This function computes the TP, FP, TN, FN based on a true vector with
% detected meals D, and a computed vector with detected meals D_detected.
% It uses a stride since the meal will be detected a time period after the
% meals was given.
% It uses the indices for the true meals and the detected meals to compare
% if in the stride a meals should be detected or not and vise versa.
%
% INPUT:
% stride            - The maximal time period it takes from the meal to be
% detected
% D                 - The true vector of real meals.
% D_detected        - The estimated vecor of 0 or 1. 1 meaning meals is
% detected
% U                 - Bolus insulin
%
% Ts                - The time between control steps
% N                 - The number of observations
%
% OUTPUT:
% Two outputs being TP, FP
%
% 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
%

% Initializing
falsenegative   = 0;
falsepositive   = 0;
truepositive    = 0;
truemeals       = zeros(1,N);
mealdetec       = zeros(1,N);

% Changing datatype of D to binary
for i = 1:N

    if D(1,i) >= 50/Ts % Not considering the snackmeals
        D(1,i) = 1;
```

```
    else
        D(1,i) = 0;
    end

end

% Finding the indices for the truemeals
for i = 1 : N
    if D(1,i) == 1
        truemeals(i) = i;
    end
end

% Finding the indices for the detected meals
for i = 1 : N
    if D_detected(i) == 1
        mealdetec(i) = i;
    end
end

% Removing all the zeros so there is only the indices left
idxdetecmeals = nonzeros(mealdetec)';
idxtruemeals = nonzeros(truemeals)';

% Examine if there are no true meals where there are detected meals
for i = 1:length(idxdetecmeals)

    % The idx value when meal has been detected
    k = idxdetecmeals(i);

    if (k-stride) < 1
        j = k-1;

        if  sum(D(1,k-j:k)) == 0
        falsepositive = falsepositive + 1;
        end

    elseif sum(D(1,k-stride:k)) == 0
        falsepositive = falsepositive + 1;
    end
end

% Examine if there are true meals where there are detected meals
for i = 1:length(idxdetecmeals)

    % The idx value when meal has been detected
    k = idxdetecmeals(i);

    if (k-stride) < 1
        j = k-1;

        if  sum(D(1,k-j:k)) == 0
        falsepositive = falsepositive + 1;
        end

    elseif sum(D(1,k-stride:k)) == 1
        truepositive = truepositive + 1;
```

```
    end
end


end
```

```
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
```