Automatic Bolus Pancreas

01666 - Fagprojekt

Laurits Fog Balstrup - s200625

Frederik Emil Nagel - s204213

Morten Møller Christensen - s204258

Vejledere:

John Bagterp Jørgensen

Tobias Kasper Skovborg Ritschel

DTU Compute

20/06 - 2022

Contents

1 Abstract

Background

The Automatic Bolus Pancreas (ABP) is an attempt at making a model-free artificial pancreas that can detect meals and give corresponding meal boluses while also continuously administering basal insulin.

Methods

The ABP is tuned and tested by simulating a dynamical system modelling the glucose concentration of a person, with meals ranging from 70-150 g CHO at different times of the day. The final result is based on simulating 100 patients, i.e. with the Monte Carlo method.

Results

Comparing a PID pancreas, which only administers basal insulin using a PID controller, to the ABP on a 4-day simulation, one can observe that the difference in subcutaneous glucose concentrations (CGM measurements) is quite small. While the ABP gets closer than preferred to a critically low level of CGM measurements for short periods of times, it generally performs better than expected. The PID pancreas on the other hand spends more time with higher than preferred CGM measurements, but from the simulations doesn't reach any critically low levels for a significant amount of time.

Conclusion

From this report we can conclude that the ABP does not perform significantly better than the baseline PID pancreas it is compared with. Due to the trial and error method used to find the different parameters for the ABP, it is not certain that the optimal parameters have been used. Further optimization of these parameters might lead to a better performance from the ABP.

2 Introduction

It is estimated that around 1/10 of the worlds population will have diabetes by 2030 [7]. Diabetes does not just impact the individuals, but also families and our society as a whole. When the glucose concentration increases for a non-diabetic person, the pancreas begins to release insulin, which allows the glucose to be used as energy in the cells. For a diabetic, the pancreas does not produce enough, if any insulin, which results in the glucose concentration in the blood not decreasing. If a diabetic forgets to keep an eye on the glucose concentration in their blood it could become life threatening. A problem which an artificial pancreas could help to solve.

This report attempts a digital implementation of an automatic bolus pancreas (ABP), capable of regulating the continuous basal rate, detect meals and give corresponding meal boluses. The implementation is based on elements from control theory and previous work done in the field of artificial pancreas. The implementation is sought to be as general as possible i.e. relying as little as possible on patient specific model parameters. To measure it's performance, it's compared to a baseline consisting of a PID controller.

Note that a list of abbreviation can be found in appendix tab. (9.1).

3 Model

Based on [1], the glucose concentration of a person is modelled with The Medtronic Virtual Patient Model (MVP), which is a system of first order differential equations of the form

$$\dot{x}(t) = \begin{bmatrix} \dot{D}_1(t) \\ \dot{D}_2(t) \\ \dot{I}_{sc}(t) \\ \dot{I}_p(t) \\ \dot{I}_{eff}(t) \\ \dot{G}(t) \\ \dot{G}_{sc}(t) \end{bmatrix} = \begin{bmatrix} d(t) - \frac{D_1(t)}{\tau_m} \\ \frac{D_1(t) - D_2(t)}{\tau_m} \\ \frac{u(t)}{\tau_1 \cdot C_I} - \frac{I_{sc}(t)}{\tau_1} \\ \frac{I_{sc}(t) - I_p(t)}{\tau_2} \\ -p_2 I_{eff}(t) + p_2 S_1 I_p(t) \\ -(GEZI + I_{eff}(t))G(t) + EGP_0 + \frac{1000 \cdot D_2(t)}{V_G \tau_m} \\ \frac{G(t) - G_{sc}(t)}{\tau_{sc}} \end{bmatrix} = f(x(t), u(t), d(t), p_f), \tag{1}$$

where the parameter vector is given by

$$p_f = \begin{bmatrix} \tau_1 & \tau_2 & C_I & p_2 & S_I & GEZI & EGP_0 & V_G & \tau_m & \tau_{sc} \end{bmatrix}^\top. \tag{2}$$

Standard parameter values can be seen in appendix. tab. (4).

Described as a differential equation in state space terms it becomes

$$\dot{x}(t) = f(x(t), u(t), d(t)), \qquad y(t) = h(x(t), u(t)) = G_{sc}(t), \tag{3}$$

with x being the vector of state variables, the input u being the control signal, the input d being a disturbance signal and the output y being the measurement.

The control signal $u(t)$ describes the insulin injection rate in mg/dL at time t consisting of a bolus $u_{bo}(t)$ and basal rate $u_{ba}(t)$ such that $u(t) = u_{ba}(t) + u_{bo}(t)$. The basal insulin is continuously injected adjusting for mostly small disturbances, whereas the bolus insulin is a single injection given with large disturbances. The disturbance signal $d(t)$ describes the meal intake size in carbohydrates pr. minute. It consists of sudden spikes in value corresponding to the patient eating a meal at a given time t. d is therefore fixed whereas u can be adjusted.

Solutions for the optimal injection rate u are obtained in discrete time steps t_k, with $k = 0, 1, 2, ...$ When describing the evolution of u it is therefore in terms of the change between each t_k. The model can therefore be described in terms of difference equations instead of differential equations by

$$x(t_k) = f(x(t_{k-1}), u(t_{k-1}), d(t_{k-1})), \qquad y(t_k) = h(x(t_{k-1}), u(t_{k-1})) = G_{sc}(t_k). \tag{4}$$

3.1 A dynamical system

The system (1) is either open or closed loop depending on u. In the case where u is a predefined function or simply constant the system is open loop meaning that the state vector at a given $x(t_k)$ is not dependent on the previous state vector $x(t_{k-1})$.

In the more relevant case, $u(t_k)$ actually is dependent on $x(t_{k-1})$ resulting in a closed loop system with feedback. The goal is then to compute $u(t_k)$ based on the glucose concentration $G(t_{k-1})$.

In practice the glucose concentration $G(t_k)$ can't be measured continuously as this requires blood samples. Instead the measured subcutaneous glucose concentration $G_{sc}(t_k)$ via a continuous glucose monitoring (CGM) system is used since it correlates closely with the actual glucose concentration $G(t_k)$. It is therefore assumed that optimizing for $G_{sc}(t_k)$ will also optimize for $G(t_k)$. See appendix fig. (15) for at plot comparing CGM measurements with glucose concentrations.

3.2 PID controller

A PID controller is used to adjust the basal rate based on the nominal basal rate $\overline{u}_{ba}$ and the output $y = G_{sc}$. Suppose one chooses an optimal value $\overline{y}$ for this state. At any given time interval the goal is then to minimize the error $e(t_k) = |y(t_k) - \overline{y}|$. With a PID controller this is done using three components: a proportion, an integral and a derivative. The basal rate is then given by

$$u_{ba}(t_{k+1}) = \overline{u}_{ba} + K_P\, e(t_k) + K_I \int_{t_0}^{t_k} e(\tau)\mathrm{d}\tau + K_D \frac{\mathrm{d}e}{\mathrm{d}t_k}. \tag{5}$$

The choice of the constants K_P, K_I and K_D is discussed later. Due to the model working in discrete time steps the computation in practice will look a bit different from (5). The bolus is in practice given by

$$u_{ba}(t_{k+1}) = \overline{u}_{ba} + K_P\, e(t_k) + K_I \sum_{i=0}^{k} e(t_i) + K_D \frac{y(t_k) - y(t_{k-1})}{t_k - t_{k-1}} \tag{6}$$

$$= \overline{u}_{ba} + P_k + I_k + D_k. \tag{7}$$

3.3 Optimal bolus

For larger disturbances in $y = G_{sc}(t)$ such as when eating a meal, the PID controller alone is insufficient.

Let d_k be the meal size, $u_{bo,\,k}$ the bolus size for the meal and $x_k = x(t_k)$ the initial condition, the latter usually being the steady state vector. The goal is now to compute the optimal bolus size $\overline{u}_{bo,\,k}$ based on d_k, x_k and the parameters p_f.

To do this, first consider a glucose penalty function given by

$$\rho(y(t)) = \frac{1}{2}(y(t) - \overline{y})^2 + \frac{\kappa}{2}\max\{(y_{min} - y(t)), 0\}^2, \tag{8}$$

where $\overline{y}$ is the steady state glucose concentration, y_{min} is the lowest allowed glucose concentration and $\kappa = 10^6$ is a penalty scalar. The function $\rho(y(t))$ then describes how optimal the glucose concentration $y(t)$ is at a given time t. This means that

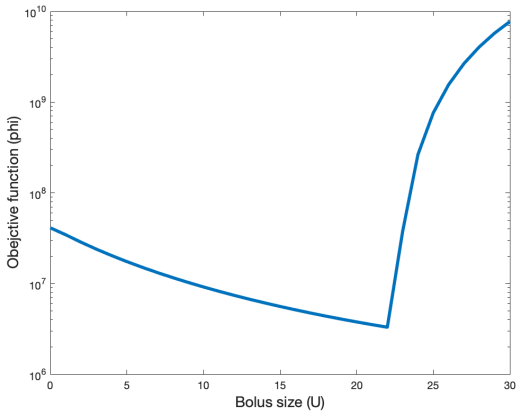
$$\phi = \int_{t_k}^{t_k+T_{bo}} \rho(y(t))dt \tag{9}$$

describes how optimal $y(t)$ has been for $t = t_k...t_k + T_{bo}$. The value T_{bo} is chosen as the time it takes for the glucose concentration to reach a steady state after eating a meal. The optimal bolus size is then given by

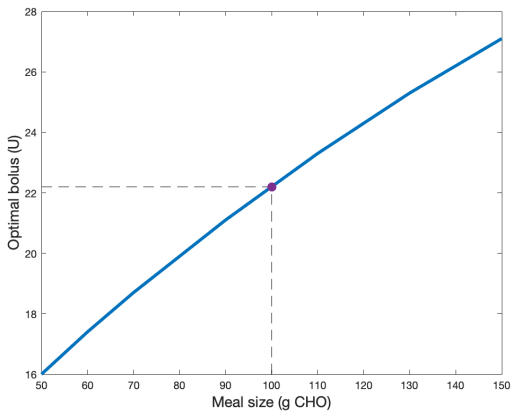
$$\overline{u}_{bo,\,k} = \min_{u_0}\{\phi(x_k, u_{bo,\,k}, d_k, p_f)\}. \tag{10}$$

While the basal injection rate computed with the PID controller is continuous, the optimal bolus size computed with the method above results in a one time injection rate accross a single 5 minute interval.

Shown below is a plot of ϕ with T_{bo} corresponding to 5 hours for $d_k = 100$ and $u_{bo,\,k} = 0, 1, ..., 30$ and of $\overline{u}_{bo,\,k}$ for $d_k = 50, 60, ..., 150$ with the solution for $d_k = 100$ highlighted.



(a) Value of ϕ for different bolus sizes.



(b) Optimal bolus for different meal sizes.

From (a) it is observed that for a meal size of 100g CHO, which is an unrealistically large meal, the optimal bolus size is around 22 U. It is also seen that it is better to give a slightly smaller bolus size than a bigger since the change in the objective function value after the optimal bolus size of around 22 changes quite rapidly for the worse.

Finding the optimal bolus size is a computationally expensive task. From (b) it's seen that there seems to be an almost linear relation between the meal size and optimal bolus, which would make finding the optimal bolus size a lot less expensive.

3.4 Super bolus

Computing the optimal bolus comes with the issue that the estimation is dependant on a model which is tuned to a specific patient. Finding an alternative method is therefore preferred. An idea is to compute the bolus size as the sum of the hypothetical basal which would be given during a time period given by T_{ba} after the meal. This bolus will be called super bolus.

Given a meal $d(t_k)$, the super bolus can thus be computed as

$$u_{bo}(t_k) = \sum_{i=k}^{k+T_{ba}} u_{ba}(t_i). \tag{11}$$

But this solution is actually also model specific because it simulates the virtual patient to estimate the hypothetical future basal rate. An alternative idea is to assume that the basal rate during the time period T_{ba} is equal to the current basal rate and that the optimal bolus correlates positively with the mean of the slope of the CGM measurements Δy during some previous time period given by T_{dG}. Setting

$$\overline{\Delta y}(t_k) = \frac{1}{T_{dG}} \sum_{i=k-T_{dG}}^{k} \Delta y(t_i) \tag{12}$$

thus gives

$$u_{bo}(t_k) = \alpha \, T_{ba} \, y(t_k) \min\{0, \beta + \gamma \overline{\Delta y}(t_k)\}, \tag{13}$$

where $\alpha = 0.2$, $\beta = 0.5$ and $\gamma = 0.1$ are scalar constants adjusted via trial and error. This solution is not model specific but assumes that the optimal bolus is linearly proportional to the current CGM measurement, the average CGM derivative for the past T_{dG} time period and to the length of the time period T_{ba}.

3.5 Combining PID controller and bolus computation

Despite the simple formula to combine basal and bolus rate $u(t) = u_{ba}(t) + u_{bo}(t)$ computing the two in parallel introduces some challenges due the their coupled nature.

Computing the optimal bolus at t_k using (10) requires knowledge of $u_{ba}(t)$ from $t = t_k..t_k + T$ which in turn is dependant on the bolus given at t_k. This type of problem can sometimes be solved using the shooting method but with no guarantee of finding the global minimum. The solution chosen in this report is to simulate the system from $t = t_k..t_k + T$ with a bolus $u_{bo}(t_k) = 0$ and assume that u_{ba} from this simulation is similar to what will be given with a nonzero bolus at t_k. This assumption is obviously false since u_{ba} will be smaller if a bolus is given, and the resulting $u_{bo}(t_k)$ is likely smaller than the true optimal bolus.

Another solution is naturally to use the super bolus described in the previous section. Two methods (11) and (13) are available, the first being model specific and the other general.

Another challenge is tied to the PID controller. When the glucose concentration increases after meal consumption, the integral term in (6) increases substantially. Despite giving a bolus injection at the time of consumption, the glucose concentration will increase due to the latency of the insulin effect. This can cause integrator windup, which can lead to significant overshooting by the PID controller. Since the PID controller is mostly meant for stabilizing the glucose concentration in between meals, it therefore makes sense to halt the integration process for some period T_{halt}, which will be called the halting period, after each meal. Given a meal at t_0, the integral sum in (6) at time t_k is computed as

$$I_k = \begin{cases} I_{k-1} & \text{if} \quad t_k - t_0 \leq T_{halt} \\ I_{k-1} + K_I\, e(t_k) & \text{otherwise.} \end{cases}. \tag{14}$$

Despite the efforts above, the PID controller will still increase the basal rate u_{ba} by some amount after a meal due to the proportion and derivative terms. Ideally, the basal rate should decrease after a bolus injection to prevent overshooting. There are many possible solutions to this problem. One idea is to simply set the basal rate $u_{ba}(t_k) = 0$ during the halting period. Due to the health risks of too fast titration, a ramping function $r : [0, T_{halt}] \to [0, 1]$ is used to gradually increase the basal rate from 0 to the $u_{ba}(t_k)$ given by (6). The basal rate is then given by

$$u_{ba}(t_{k+1}) = \begin{cases} r(t_k)(\overline{u}_{ba} + P_k + I_k + D_k) & \text{if} \quad t_k - t_0 \leq T_{halt} \\ \overline{u}_{ba} + P_k + I_k + D_k & \text{otherwise.} \end{cases}. \tag{15}$$

A ramping function $r(t)$ based on the sigmoid function with an option of adjusting the ramping period T_{ramp} is used. See fig. (16) for an illustration of the sigmoid ramping function.

3.6 Meal detection

3.6.1 The Meal Detection algorithm

For a given timeline of glucose measurements ($G_{sc}(t_k)$), the meal detection algorithm, inspired by [2], uses previous measurements to detect a meal at the current time t_k. Below is a short pseudo code for the meal detection algorithm.

Algorithm 1 Meal Detection at time t_k

Require: $G_{sc}(t), \Delta G, \Delta t, t$

1: $k \leftarrow length(G_{sc}(t))$
2: $GF_{NS} = NoiseSpikeFilter(G_{sc}(t), \Delta G)$ ▷ Preprocessing
3: $GF = LowPassFilter(GF_{NS}, \Delta t, \tau_F, k)$ ▷ τ_F is chosen in [2]
4: $GF' = ThreePointLagrangian(GF, t)$
5: $GRID^+ = LogicDetection(GF, GF', G_{min}, G'_{min,2}, G'_{min,3})$ ▷ $G_{min}, G'_{min,2}, G'_{min,3}$ are chosen in [2]
6: *Return* $GRID^+$ ▷ Returns 1 if a meal is detected otherwise 0

If the reader wants to know what the four functions i.e. "NoiseSpikeFilter", "LowPassFilter", "ThreePointLagrangian" or "LogicDetection", refer to [2] page 308. A brief explanation of the parameters can be seen in the following table:

ΔG	Maximum allowable ROC [mg/dL]
Δt	Sampling period [Min]
τ_F	Filter time constant [Min]
$G_{min}, G'_{min2}, G'_{min3}$	Boundary parameters

The values used for these parameters are inspired by [2] and are as follows: $\tau_F = 6$, $G_{min} = 130, G'_{min2} = 1.6, G'_{min3} = 1.5$, $\Delta G = 15$ and $\Delta t = 5$. Note that [2] has $\Delta G = 3$ and $\Delta t = 1$, but when testing the algorithm, a big difference in using the two different set of parameter values is found. When using 3 and 1 for ΔG and Δt the GRID algorithm is very precise in finding the real meals and does not classify small disturbances in the CGM as meals. Despite this it is very slow at finding the peaks. On the other hand, when using 15 and 5 for ΔG and Δt the GRID algorithms finds a lot of meals in which many of them are false positives, but the average meal detection time is significantly better, which is more preferred, and therefore 15 and 5 is used.

Note that this meal detection algorithm can detect the same meal multiple times in a row, meaning that a possible sequence in the output vector could be:
$[...0, 0, 1, 1, 1, 1, 1, 0, 0, 0, ...]$ where all the ones indicates the find of the same meal. Another thing to consider with the meal detection algorithm is that it runs in real time. Since everything runs in real time, it means a delay before the algorithm detects a meal is inevitable, because the CGM first need to rise for the algorithm to detect it. Thus one of the most important measures to tell whether or not the meal detection algorithm works is the difference between an actual meal and the algorithm detection of the meal.

3.6.2 GRID algorithm filter

One problem with the GRID algorithm's output is the fact that it is prone to detecting the same meal multiple times. This can make it difficult to test and measure the performance. A solution to this problem is to create a filter for the GRID algorithms output.

The filter consists of three steps. Step one is based on merging all the meal predictions which are relatively close together. An assumption is made that if there is a predicted meal within 30 min of another predicted meal, those two predicted meals originates from the same actual meal and should therefore be merged together as one. An example would be if the GRID output is of the form, $[0, 0, 1, 1, 0, 1, 0, 1, 0]$ this would mean the GRID found three different meals. Step one converts this to $[0, 0, 1, 1, 1, 1, 1, 1, 0]$. Step two is based on reducing multiple 1's in a row in the output to a single one. This is done by keeping the first 1 and changing the rest to zeros. In the example the output after step two would be $[0, 0, 1, 0, 0, 0, 0, 0, 0]$. The last step is based on trying to filter out the meal predictions that are caused by noise. These are called false positives. The way a false positive is classified is by looking in a time interval around each meal prediction and if there is no actual meal it is a false positive. This time interval spreads from two and a half hours before the meal prediction and 25 min after the meal prediction.

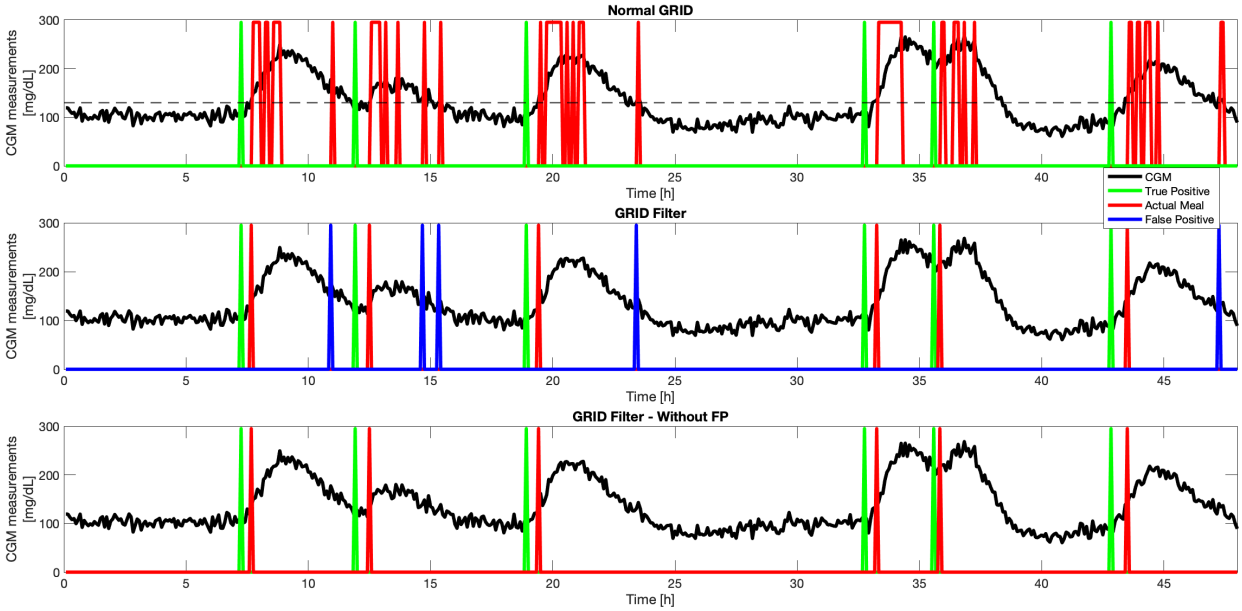


Figure 2: Normal GRID output vs. our GRID filter output, simulated using measurement noise

Figure 2 is a visualisation of how the GRID filter changes the output of the GRID algorithm. In the first plot the normal GRID output is visualised and it is clearly seen that it detects multiple meals, when there is a high incline in the CGM. It is also seen that is has a tendency to predict a meal, when the CGM is on its way down (around 11h and 23h on the plot). This is likely caused by the random noise spikes that happens.

The two last plots are of the GRID filter output. One has the meal predictions that are classified as false positives included. From this it's seen that the false positives are the ones where the CGM measurements are on their way down, which are the ones wanted to be removed. In general it is a lot easier to see where and when a meal is predicted after the GRID filter is applied.

3.6.3 Meal Size Estimation

[1]As stated earlier if the patient forgets to specify a meal it can have severe consequences. The grid algorithm only detects a meal but what about the size of the detected meal. Since neither to much or to little insulin is good for the patient, it indicates that the meal size needs to be considered. The meal size estimation is inspired by [5] equation 10-15, which first tries to detect the meals and then give the detected meals a "weight" depending on the change in measurements.

Meal Size Estimation algorithm

To create a meal size estimator (MSE), the first and second derivatives of $G_F(t)$ needs

[1]This section is optional, since MSE is not used in the final ABP

to be estimated. An estimate to G'_F is found from Harvey [2]. By using the same 3 point Lagrangian estimation algorithm, but replacing $G_F(k)$ with $G'_F(k)$, an approximation for the second derivative is found. Below is the pseudo code to approximate $G'_F(k)$ & $G''_F(k)$:

Algorithm 2 Estimating $G'_F(k)$ & $G''_F(k)$

Require: $G_{sc}(t), t$

1: $k \leftarrow length(G_{sc}(t))$
2: $GF_{NS} = NoiseSpikeFilter(G_{sc}(t), \Delta G)$ ▷ ΔG is, in this case, chosen to be 3.5
3: $GF = LowPassFilter(GF_{NS}, \Delta t, \tau_F, k)$ ▷ τ_F is chosen to be 6 and Δt is, in the case, 1
4: $G'_F(1) = G'_F(2) = G''_F(1) = G''_F(2) = 0$
5: **for** `i=3:k` **do**
6: $T1 \leftarrow (t_i - t_{i-1})/((t_{i-2} - t_{i-1}) \cdot (t_{i-2} - t_i))$
7: $T2 \leftarrow (t_i - t_{i-2})/((t_{i-1} - t_{i-2}) \cdot (t_{i-1} - t_i))$
8: $T3 \leftarrow (2 \cdot t_i - t_{i-2} - t_{i-1})/((t_i - t_{i-1}) \cdot (t_i - t_{i-2}))$
9: $G'_F(i) = T1 \cdot GF_{i-2} + T2 \cdot GF_{i-1} + T3 \cdot GF_i$
10: $G''_F(i) = T1 \cdot GF'_{i-2} + T2 \cdot GF'_{i-1} + T3 \cdot GF'_i$

The MSE algorithm proposed in [5] uses data which is measured ones every minute. This is a problem since the simulated data used and the real data set, only measures once every fifth minute. To avoid using data that's older than 30 minutes, small changes to the algorithm is made. For equation 10 (same as 'Step 1') in the article, the parameter checks for different i's the i's in the used algorithm goes all the way up to 30. This is because it measures every minute. Dividing the max for all i's with 5, the correct intervals are found. e.g. if its i_1 the algorithm would only check for $\exists i_1 = 1...30/5 = 1...6$, same can be said for i_2, i_3, i_4. This changes ensures that the algorithm only looks 30 minutes back.

The idea with the MSE was to implement it into the ABP. Looking at fig: 33 in appendix, one can see that the MSE algorithm gives multiple impulses per meal. Because it gives multiple impulses it means that the ABP would have to give the bolus in many small steps, whereas the ABP trying to be made will give all the bolus at once. Since it is a completely different way the ABP works, the MSE wont be used in the implementation, since it would make the ABP perform worse. A place where the properties of MSE could be to some use, is if the bolus is given in smaller amounts over a larger time interval, like in the paper [6], where the bolus is given over longer periods. Thus the ABP will only use meal detection and not meal detection & MSE as first intended.

3.7 Automatic bolus pancreas

To integrate the GRID algorithm in the PID controller, the GRID algorithm is continuously run with the glucose concentration from the last time period T_{GRID}, i.e. at time t_k with $y(t),\ t = t_k - T_{GRID}...t_k$. To filter out multiple detections of the same meal, a constraint that no meal must've been detected throughout the last T_{meal} time period is implemented. If any nonzero element is found in the resulting $GRID^+$ output and the previously described constraint is satisfied, a meal is detected. The bolus estimation using (13) is then computed and the corresponding PID controller reactions from (14) and (15) are performed. As a last safety measure, a maximum allowed bolus size $U_{bo,\,max}$ [U] is implemented by $u_{bo}(t_k) = \min(U_{bo,\,max}, u_{bo}(t_k))$. The ABP is thereby only relying on the past and current y measurements.

3.8 Control parameter estimation

To search for optimal control parameters K_P, K_I and K_D the penalty function (8) is used as the performance score. A 3D matrix P consisting of all possible combinations of different chosen control parameters are used to simulate a single patient for N number of days with the resulting y being used to compute the penalty score ϕ from (9). The resulting 3D matrix Φ consisting of all penalty scores corresponding to the control parameter combinations from P, can then be used to find the optimal combination by finding the minimum value and it's corresponding index. To further help in the search for optimal control parameters the matrix Φ is visualized as an array of 2D images. Each image corresponds to a K_P-value and each axis of the image corresponds to the K_I- and K_D-values. An example of the method in use can be seen in Appendix sec. (9.6).

4 Data and experiments

4.1 Simulated data

The GRID algorithm is tested via simulation using the MVP model. The solution to the MVP model (1) is found iteratively using the Euler method. To make sure the accuracy is sufficient, the solution to the MVP model from the Euler method is compared to the solution of the Matlab solver ode15s, assuming that the ode15s solution for sure is sufficiently accurate. Based on the result shown in fig. (17), it is concluded that the Euler method is sufficient.

To make the simulations more realistic, two adjustments are made to the simulation method.

For the first adjustment, the measurement inaccuracies of the device measuring the glucose concentration in the blood of the patient is simulated. This is done by adding a normal distributed stochastic variable to the output of the MVP model i.e. to each element from $y = G_{sc}$.

For the second adjustment, the noise from the biology of the patient is added. Things such as stress or exercise can also effect the glucose concentration of the patient. The Euler-Maruyama method is used to simulate this [3].

The two adjustments are only used when testing the GRID algorithm and the final fully automatic pancreas. Otherwise the deterministic model is used.

To further make the simulation as realistic as possible, a meal plan is simulated for the patient. The meal plan consists of a breakfast, lunch, dinner and two snacks, given at different times of the day. When simulating multiple days the meal size and time differ each day and are chosen at random from the intervals given in table 1.

Meal type:	Size Interval	Time of day interval
Breakfast	90g - 150g CHO	$6.30 - 9.00$
Lunch	70g - 100g CHO	$11.30 - 13.30$
Dinner	70g - 100g CHO	$17.30 - 19.00$
Snack 1	18g - 22g CHO	$15.00 - 16.00$
Snack 2	18g - 22g CHO	$20.00 - 22.00$

Table 1: Meal size and time period of consumption for the different types of meals

To finally test the implementation a Monte Carlo method is used by simulating 100 patients. The biological differences and different eating habits of the 100 patients is simulated with the meal plan and via the parameters (2) in the MVP model. The parameters, C_I, S_I and EGP_0 are not dependent on the patient and are therefore held

constant. For the other parameters, the standard parameter values seen in the appendix (Table 4) are used with a normal distributed stochastic variable with mean zero and standard deviation of 10 percent of the parameter value added. This ensures that the noise level is relative to the parameter value.

4.2 Real data

Real data from a diabetic has been provided. It contains measurements of a single patients glucose concentration taken every five minutes over the course of six months. Further the basal rate, meal sizes and time of meal has also been provided. Since the ABP can not be tested on the real data only the first three days will be used to test the GRID algorithm and compare with the results from the simulated data.

4.3 Performance metrics

Two metrics are used to measure the performances of the models. The first is based simply on the penalty score ϕ as described in eq. (9) such that a lower ϕ value means a better score. The second is based on the glucose concentration ranges from [4]:

Category	Range [mg/dL]	Color
Level 2 hyperglycemia	$[250.2, \infty[$	Orange
Level 1 hyperglycemia	$[180.0, 250.2[$	Yellow
Normoglycemia	$[70.20, 180.0[$	Green
Level 1 hypoglycemia	$[54.00, 70.20[$	Pink
Level 2 hypoglycemia	$[0.000, 54.00[$	Red

Table 2: The 5 glycemic ranges and their coloring.

The metric is then based on the percentage of total time spent by the glucose concentration in each of the 5 ranges.

5 Results

5.1 PID and optimal bolus

Throughout this section (5.1) and the next (5.2) the control parameters used are $K_P = 0.15$, $K_I = 0.00015$ and $K_D = 1$. The process of finding these is described in Appendix sec. (3.8).

The effects of a constant basal rate (open loop), basal rate adjusted by a PID controller (closed loop) and constant basal rate with optimal bolus estimations is plotted below. The meal plan is randomly generated using the method described in (4.1).

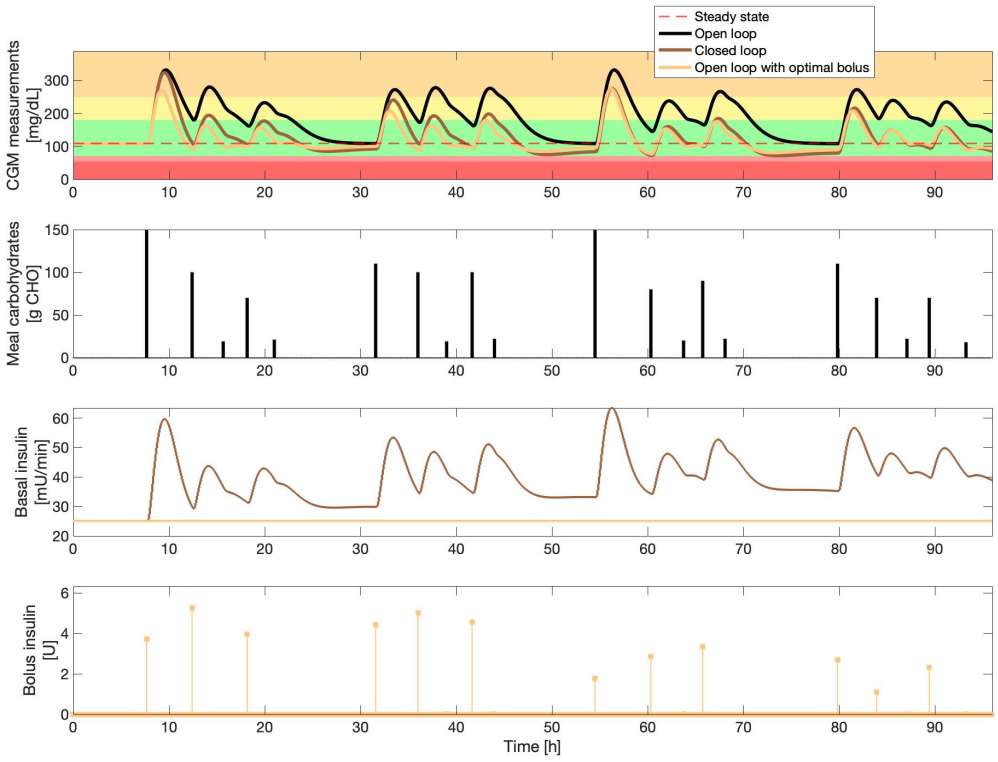


Figure 3: A single patient's CGM measurements, meal carbohydrates, basal and bolus insulin over 3 days using open loop with and without optimal bolus estimation and closed loop.

An illustration of the time in ranges (TIR's), i.e. the percentage of time spent in each of the colored ranges, is shown in (29). The open loop simulation with optimal bolus estimations performs best with 92 % of the time spent in the green range. The constant basal rate open loop simulation and the closed loop simulation spends respectively 48 % and 86 % of the time in the green range. Despite the better performance of the open loop simulation, the result suggests that a combination might be optimal.

Regarding the PID controller and optimal bolus estimation combination, the halting

methods described in (14) and (15) has the effect illustrated below:

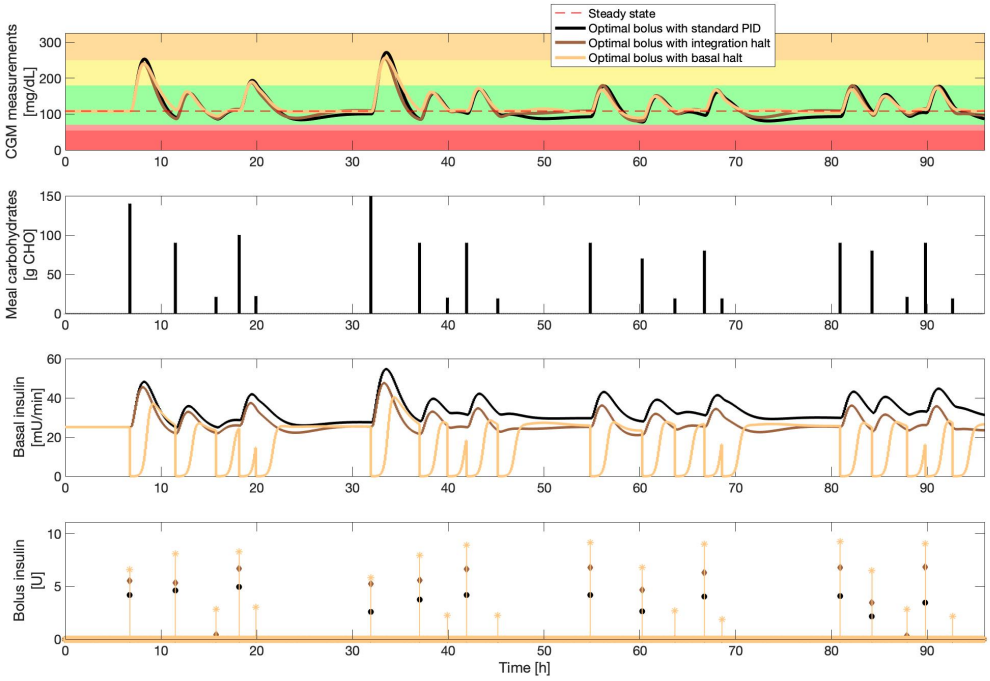


Figure 4: A single patient's CGM measurements, meal carbohydrates, basal and bolus insulin over 4 days with the standard PID, with integration halt for 3 hours and with basal halt for 4 hours.

To have the two methods work together, the PID controller must adjust it's behavior during bolus titration. Especially if the integrator winds up as it happens during day 3 and 4, the optimal bolus becomes very small with the PID taking over most of the control. The integration halt does seem to fix the problem of overshooting due to integrator wind-up but as mentioned in sec. (3.5) the basal rate should ideally decrease during bolus titration. From the TIR's shown in Appendix fig. (28) it can be seen that from this short simulation, the three methods perform almost identically. Note that the necessity of the integration halt is much more prevalent with a larger P_I. With the current parameter values, the proportional and derivative terms K_P and K_D tend to dominate.

5.2 Optimal bolus and super bolus

By now it's been shown that a combination of a PID controller and an optimal bolus estimator is probably optimal and that the PID controller should adjust it's behavior during bolus titration for the two to work together properly. But because of the optimal bolus estimation as described in (10) is model specific, an alternative method is sought. The two methods described in (11) and (13), the latter with $T_{dG} = 6$ corresponding to

30 minutes, are therefore compared to the optimal bolus method.

The method (13) is made to be performed some time after the actual meal has been consumed due to it's reliance on the effect of the meal on the output y signal, i.e. the CGM measurements, and because it will be activated by the GRID algorithm, which generally detects meals with some delay. For this test using the method (13), the system is therefore simulated for 40 minutes once a meal is detected, and the resulting CGM measurements are then used for the bolus size estimation. The resulting plot is shown below.

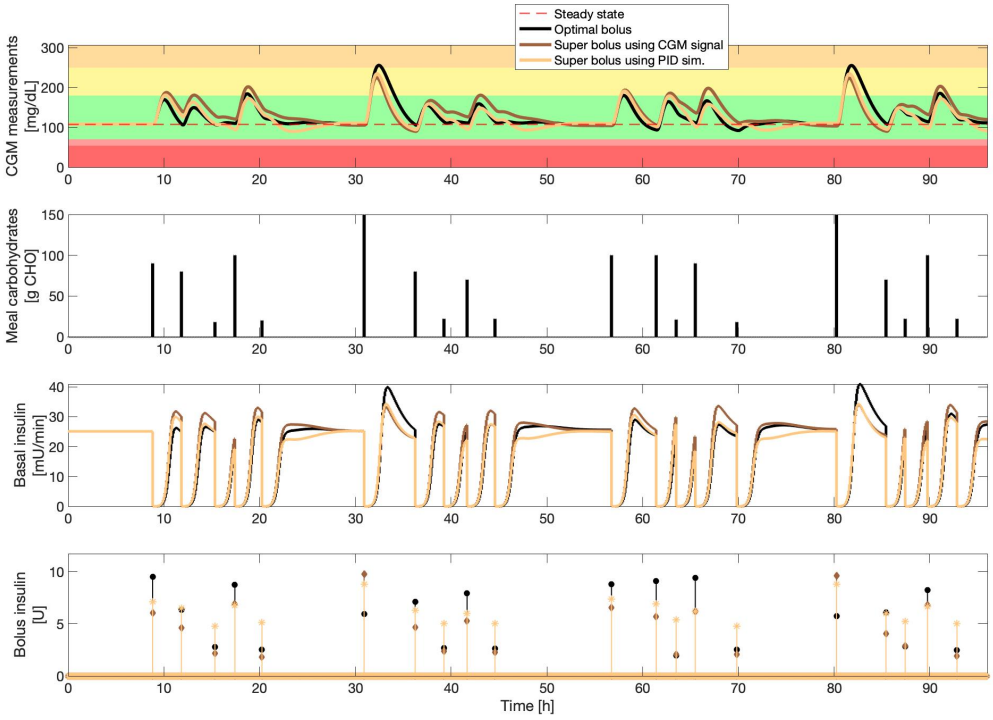


Figure 5: A single patient's CGM measurements, meal carbohydrates, basal and bolus insulin over 3 days using closed loop with optimal bolus and super bolus with and without PID simulation. All three are with basal halt for 3 hours.

All three methods correlate visibly to a surprising degree showing the same variation in the amount of bolus for the meals and with small differences in their TIR's as can be seen in appendix fig. (29). Based on a rough investigation of the correlation between the bolus sizes from the three methods as can be seen in appendix fig. (22), the CGM signal method is regarded as sufficiently accurate.

5.3 Test of GRID algorithm on simulated and real data

Meal detection is essential for our artificial pancreas to work, and we are therefore interested in testing the GRID algorithm on both simulated data and real data. Further

the GRID filter mentioned in section 3.6.2 is applied in all plots.

5.3.1 Simulation with measurement noise

The first test of the GRID algorithm is done on the simulated data, with added measurement noise. We have chosen to include both a simulation of a meal plan with and without snacks. This will test the GRID algorithms capability to detect small meals, and it will show the effects these small meals have on the blood glucose concentration.

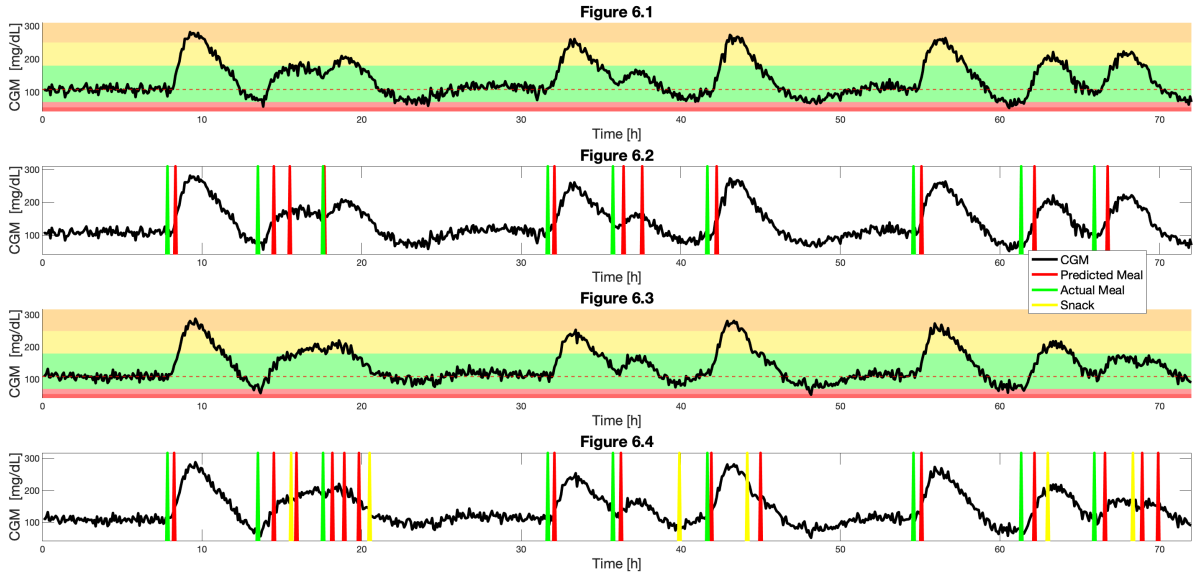


Figure 6: Simulation with measurement noise

Figure 6.1 and 6.3 are the simulated data for one patient over the course of three days. In 6.1 no snacks are added to the meal plan and in 6.3 snacks are included. There is a clear difference in the two plots, most noticeable in the very end of the plots where 6.1 has i high peak and 6.3 has a more flattened out peak. This might be caused by the snack after lunch on the last day giving a extra boost of insulin and therefore flattening out the curve more.

In figure 6.2 and 6.4 the results from the GRID algorithm is plotted along with the times a meal occurs. Generally it is seen that the GRID algorithm preforms well in finding the actual meals and is prone to finding additional predicted meals close to the actual meal. When it comes to finding the snacks we see in figure 6.4 that it misses a few of them and the ones that are found seems to be caused by random noise and not so much the snack itself. The average meal detection time is only 36 minutes for the simulation without snacks and 41 minutes for the one with. The significant difference in the two average times could be explained by the snacks which are not found giving a "false" high meal detection time and thereby influencing the average time.

Further theres a total of three false positives found between the two simulations. A plot of these can be seen in the appendix fig. (31).

5.3.2 Simulation with Euler-Maruyama

The second test of the GRID algorithm is done on the simulated data with Euler-Maruyama mvpmodel solver. Again we have chosen to include a plot with and without snacks to see the difference.

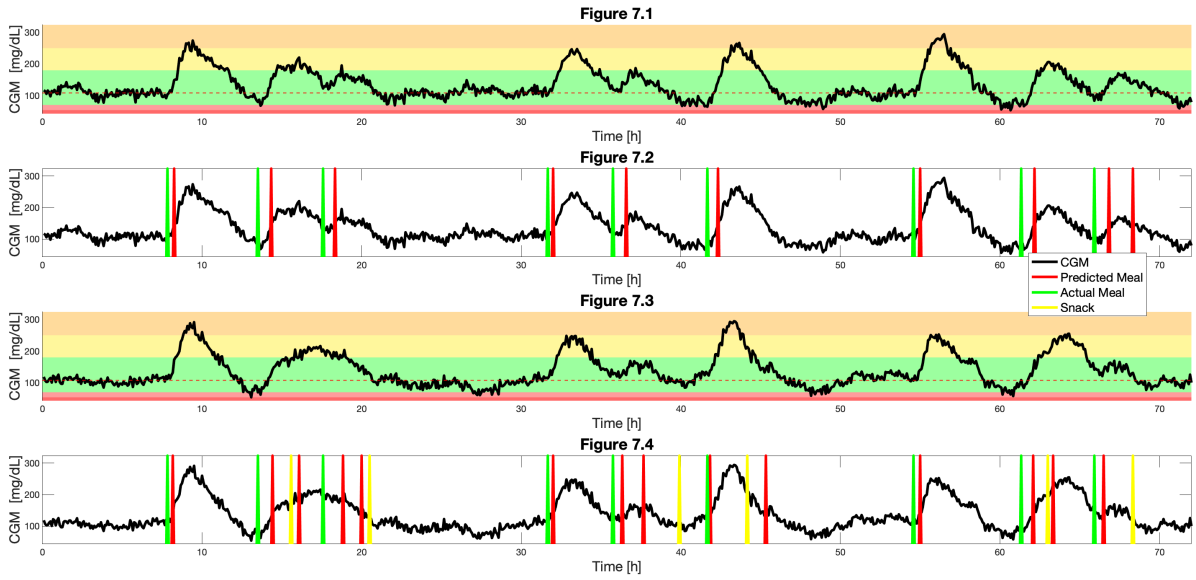


Figure 7: Euler-Maruyama simulation

Again we see a quite big difference in the simulations with and without snacks. From figure 7.2 and 7.4 its generally seen that the GRID algorithm again preforms well in finding the actual meals and it still struggles to find the snacks. The average meal detection time is 40 minutes for the simulation without snacks and 43 minutes for the simulation with snacks. Which means when the Euler-Maruyama method is applied to the simulation the GRID algorithm takes a longer time to find the meals.

Further there is found a total of 6 false positives between the two simulations which is twice as many as for the simulations with measurement noise. A plot of the found false positives can be seen in the appendix fig. (32).

5.3.3 Simulating 100 patients

To get a more general understanding of how well our GRID algorithm performs in finding the meals, we test it on simulated data of 100 different patients during the cause of 31 days. The meal plan used for all the patients is without snacks. For a better

visualisation of the simulation of the 100 patients the first day out of the 31 days are visualized together with the average of the 100 patients CGM measurements.

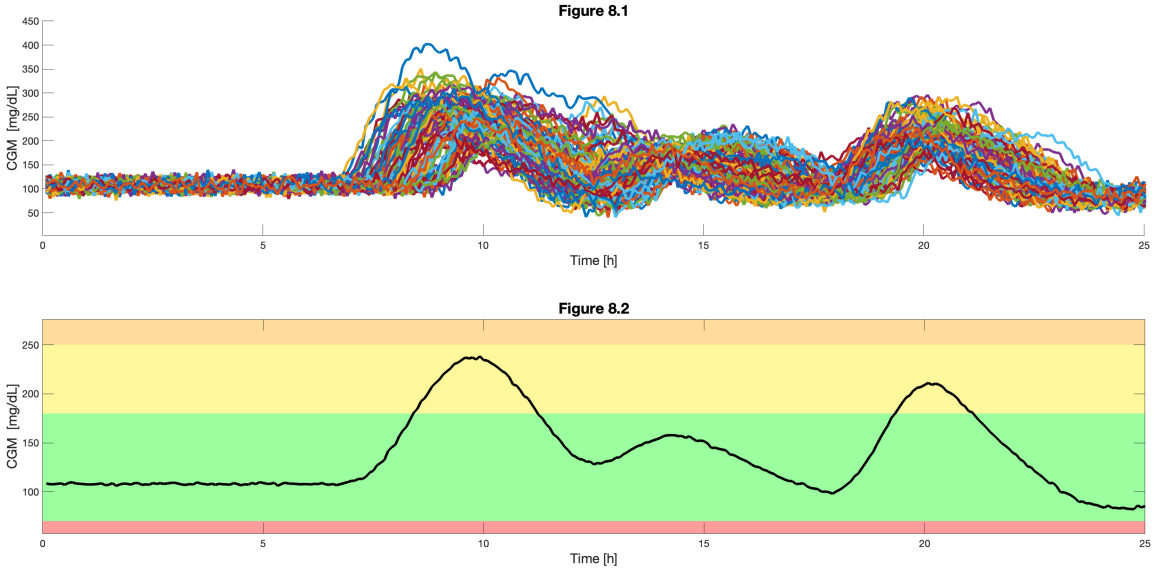


Figure 8: Simulation of 100 patients with added measurement noise

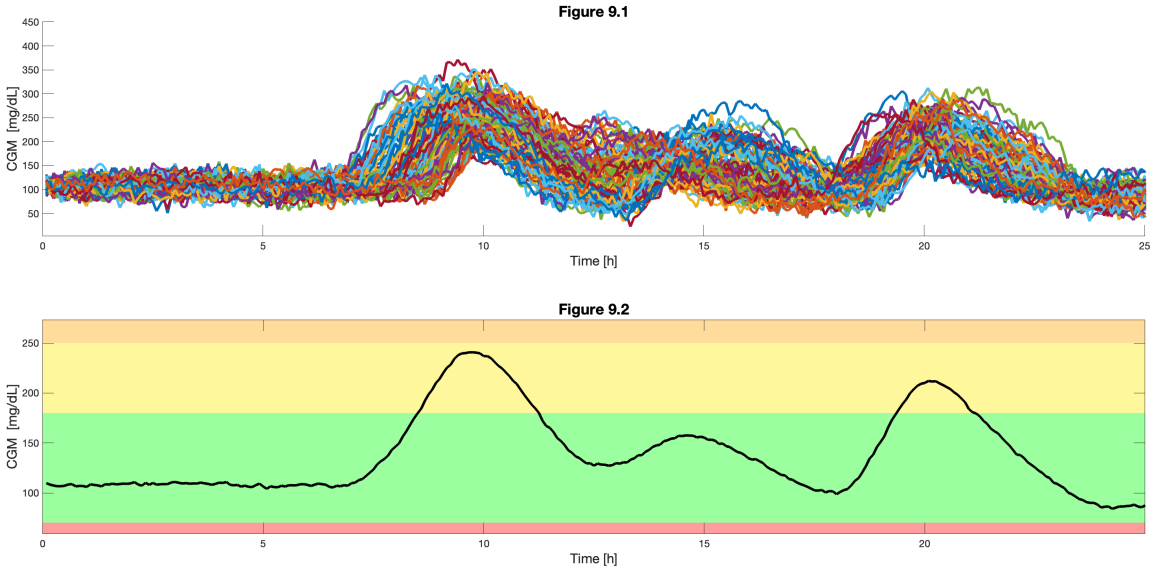


Figure 9: Simulation of 100 patients with Euler-Maruyama method applied

From figure 8.1 and 9.1 it is seen that there is a difference between the two simulation methods. The Euler-Maruyama method produces a higher variance in the CGM, but other than that they seem to follow each other quite well which is also seen in figure 8.2 and 9.2 were the averages are quite identical.

During the 31 days the 100 patients has a total of 9.300 meals. The GRID algorithm

finds 10.796 meals for the simulation with measurement noise and 11.006 for the simulation with the euler-maruyama method, which corresponds to respectively a 116 and 118 percent meal detection rate. Further the average meal detection time for the two were 41 and 41.5 minutes respectively and they found around 4.200 and 6.000 false positives respectively. To summarize the two methods founds roughly the same amount of meals in the same amount of time, though the Euler-Maruyama method found a significantly higher amount of false positives.

5.3.4 GRID on real data

The last test of the GRID algorithm is done on real data. Below is plotted the real data and the GRID algorithms predictions of where the meals are.

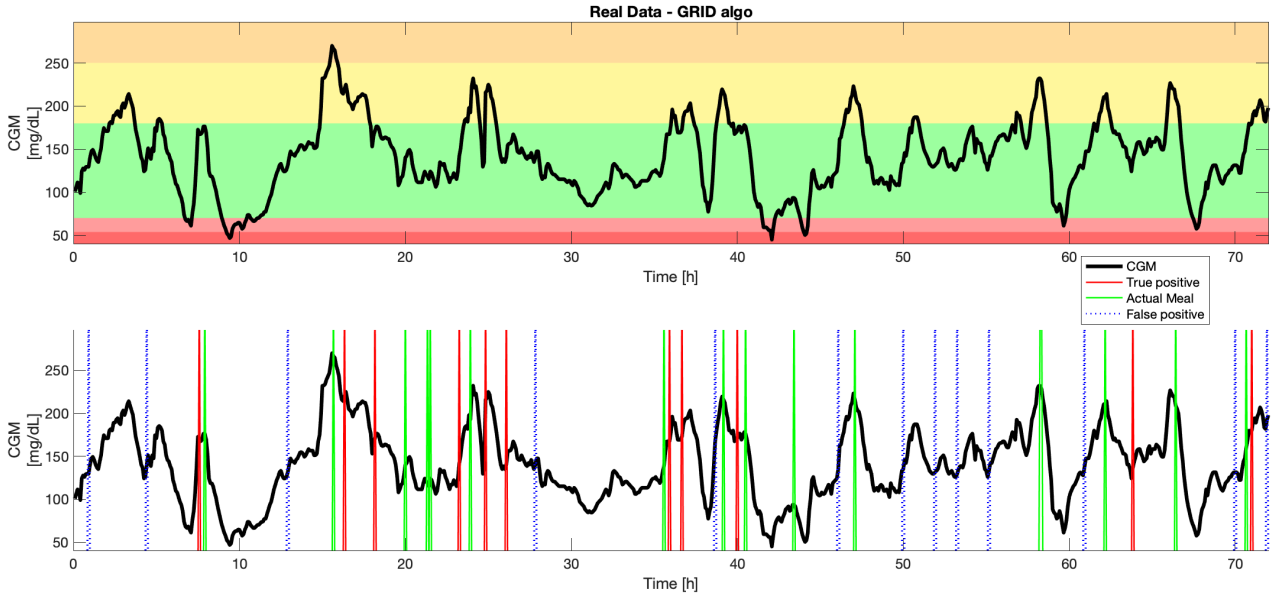


Figure 10: GRID Algorithm applied on real data

From figure 10 (Top) it is clear that the real data differs quite a lot from the simulated data obtained by our models. In the tests on the simulated data a rise in the CGM was a quite good indicator of a meal. Here the peaks seems more randomized and in most cases it is impossible to tell if it raises because of a meal or because of other factors.

In figure 10 (Bottom) the GRID algorithm is applied to the real data. It is clear from the plot that the performance of GRID on real data is worse than it has been on any of the simulations. In total GRID miss predicts 11 meals. The ones that stand out are the two located in the first couple of hours and then again in hour 50 to 55. Observe that these false meals are placed when the CGM rapidly increases. Further observe that the algorithm misses meals, like around hour 20, looking at the top plot, the very small rise could indicate that its two small snacks, which GRID usually doesn't detect.

Although GRID misses some meals, it generally performs well in finding meals. The meals it detects the most consistent is the "big" meals e.g. where the CGM increases rapidly.

Another important thing to note about the real data is that the actual meals is located in peaks on the CGM, which doesn't make sense. A reasonable explanation for this is that the patient has entered the meal after consumption, meaning the CGM has had time to rise. This results in the plot actually being somewhat wrong since meal times shown is later than the real meal times.

5.3.5 Performance statistics

Some fundamental test of the grid algorithm has been performed, to summarize all of the tests a performance table has been created. The table show what the exact number of meals is, how many has been found how long it on average takes to find a meal and last but not least how many false positives have been found. The table can be seen here:

Simulation:	Meals	Meals found	TP	FP	Procent	Average time
Noise (No snack)	9	12	11	1	122.22%	36.11 min
Noise (snack)	15	17	15	2	100%	41.15 min
Euler M (No snack)	9	15	10	5	111.11%	40 min
Euler M (snack)	15	15	14	1	93.33%	42.69 min
Noise 100	9300	15024	10796	4228	116.09%	41.13 min
Euler M 100	9300	17009	11006	6003	118.34%	41.46 min
Real data	16	24	11	13	68,75%	63,13 min

Table 3: Performance statistics for different test scenarios

From the table it can be seen that the performance of the GRID algorithm on real data differs from the test environments, this makes sense since GRID is developed/tuned using the fictional models. Another interesting thing to note is that on average the performance of GRID is nearly identical on the model with noise and the model with Euler-Maruyama.

5.4 Automatic bolus pancreas

To determine the performance of the ABP it's compared to a baseline consisting of a PID controller, which will be called the PID pancreas. First the gains K_P, K_I and K_D are tuned using the method described in sec. (3.8). The ABP gains are first tuned with no measurement noise and with the deterministic MVP model. To take into account the possible significant difference with the stochastic augmentation of the MVP

model, both the ABP and the PID pancreas get a final tuning using the stochastic MVP model. The measurement noise will simply diffuse the clarity of the optimal gains, so no measurement noise is used for the final parameter tuning. Interestingly, the PID pancreas's optimal integration gain K_I is much lower than the optimal K_I using the deterministic model. The other gains K_P and K_D are optimal around the same values as with the deterministic model. The difference between using the deterministic and stochastic model while tuning the ABP gains is more difficult to see. See appendix sec. (9.6) and sec. (9.8). A table of all parameter values used for the following simulations can be seen in appendix tab. (6).

The ABP and PID pancreas simulated for 4 days with measurement noise and the stochastic model is plotted below:

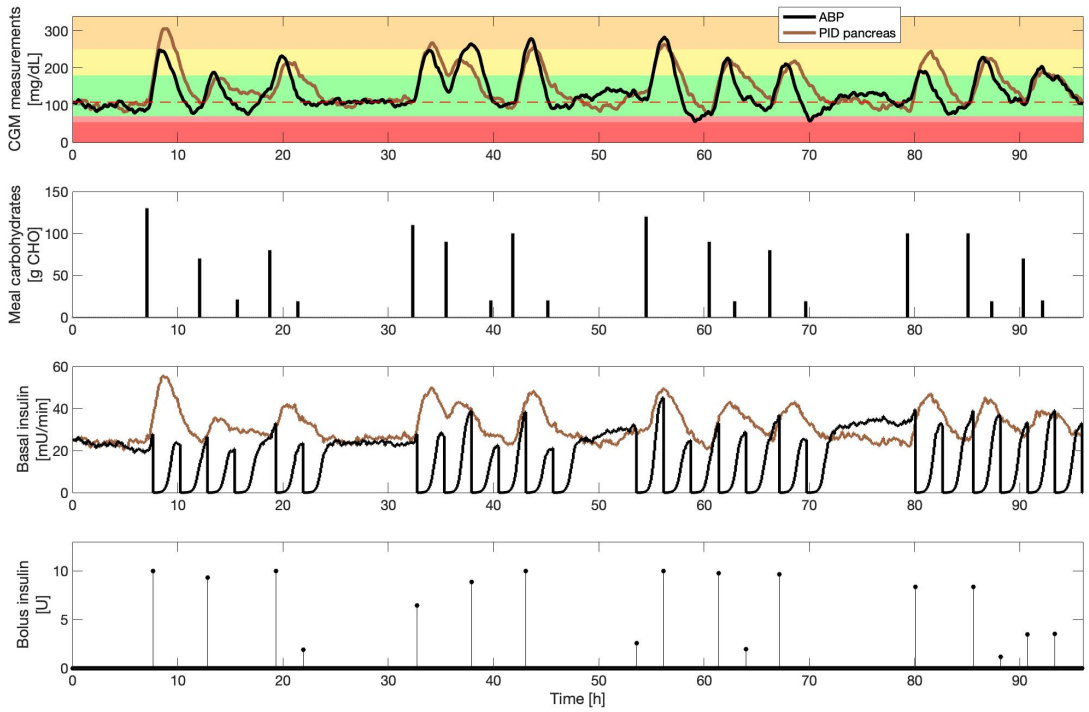


Figure 11: 4 day simulation of a single patient comparing the ABP and PID pancreas.

While the ABP does detect all meals within 30-40 minutes, it also detects a significant amount of false positives. But with the super bolus estimation method eq. (13), these false positive are handled quite effectively. Generally though, the difference in the CGM measurements between the ABP and PID pancreas from this short simulation is quite small.

A Monte Carlo simulation of 100 patients is performed. Due to the stochastic nature of the patient parameters and of the CGM measurements and dynamical system, the

TIR's of the worst case scenario based on the penalty scores is also investigated.

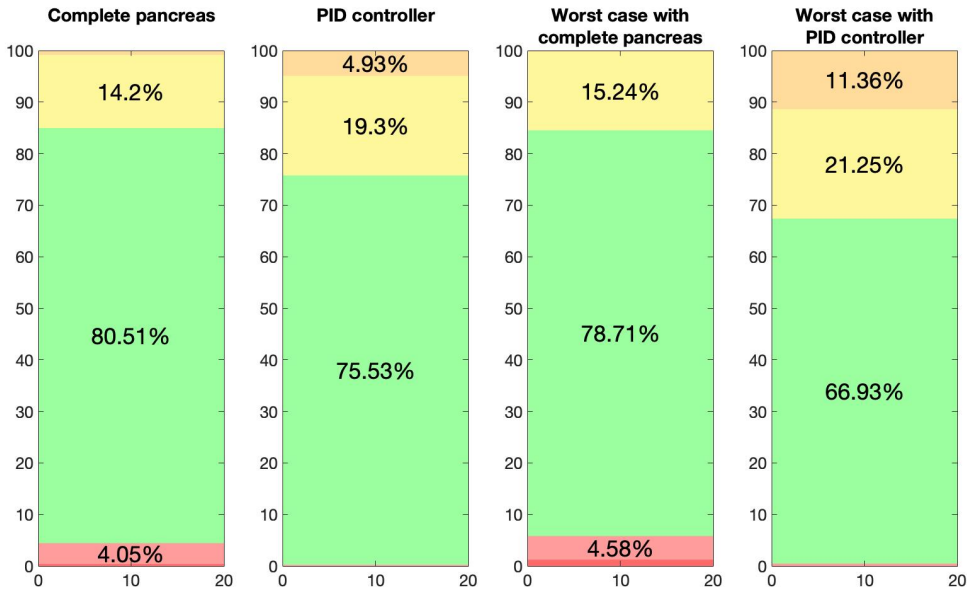


Figure 12: Average and worst case TIR's for the ABP and the PID pancreas from simulating 100 patients.

The PID pancreas generally spends more time in the yellow and orange ranges with practically no time spent in the pink and red ranges whereas the ABP spends less time in the orange and yellow ranges but more time in the pink and red ranges.

The difference between the worst case scenarios is similar to the average TIR's but more pronounced, i.e. the PID pancreas spends more time in the yellow and orange ranges with only slight difference in the pink and red ranges and the ABP spends more time in the pink and red ranges with only slight differences in the yellow and orange ranges.

To better get an understanding of the performance differences, the penalty score distributions is plotted:

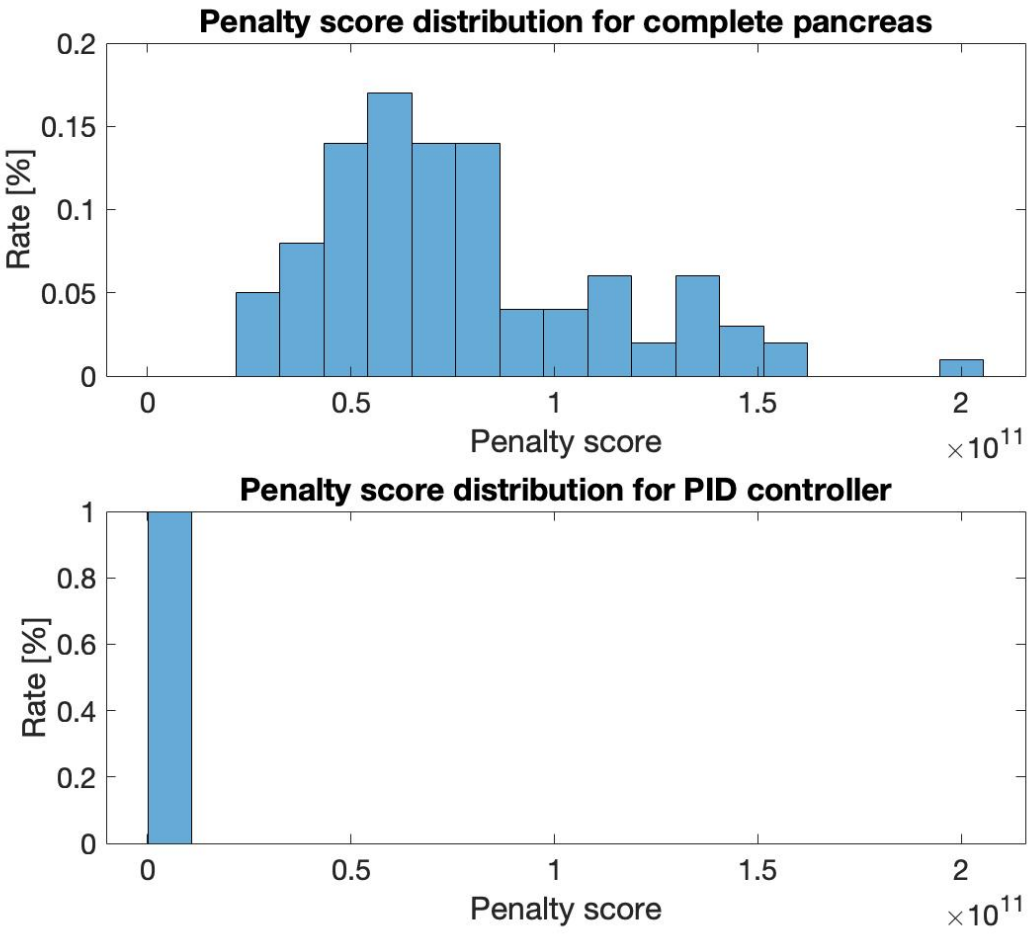


Figure 13: Distribution of the penalty scores for the ABP and the PID pancreas from the 100 simulated patients.

From this it can clearly be seen, that the PID pancreas performs significantly better than the ABP based on the penalty scores. For the purpose of comparison, the x-axis are equal making the penalty score distribution diffult to see in detail. But clearly, the penalty scores are on average much lower than those of the ABP. It is important here to keep in mind, that pink and red ranges result in very large penalty scores such that a few percentage points in those ranges can result in much higher penalty scores. With a slightly different tuning of the ABP, one where less insulin is given either by down-tuning the bolus size estimator eq. (13) parameters or the controller gains, one could possibly improve the penalty scores of the ABP significantly.

Lastly, the distributions of the CGM measurements, basal rates and bolus sizes is plotted:

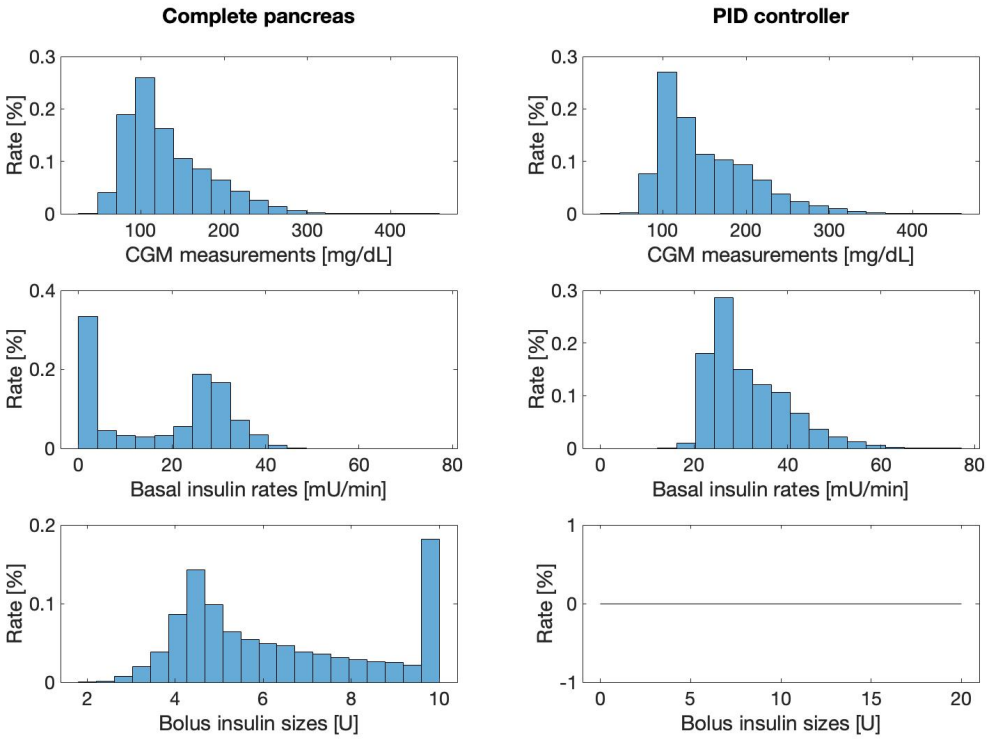


Figure 14: CGM measurement, basal rate and bolus size distributions for the ABP and the PID pancreas from the 100 simulated patients.

Of course the PID pancreas doesn't give any bolus, so the distribution of these is trivial. The ABP looks to be having a bolus size distribution with a peak of 15% at around 4.2 U, but due to $u_{bo,\,max} = 10$, the most frequent bolus size of 18% is 10 U. An expected difference is that the CGM measurements of the ABP are slightly down-shifted compared to the PID pancreas's. This corresponds to the TIR's from fig. (12). The basal rate distributions are also significantly different. The ABP spends around 35% of the time with a basal rate of almost 0 with the basal rate almost never exceeding 50 mU/min. The PID pancreas almost never goes below 15 and reaches basal rates of around $60-70$ mU/min which is more than double the nominal basal rate of 25.01 mU/min. Almost

6 Discussion

Generally, the ABP works better than expected. All the individual elements; PID controller, super bolus estimation and meal detection, does work together with no apparent major complications, i.e. no worst case scenario with some critical malfunction occuring in the simulations performed. Furthermore, the ABP is model-free, i.e. non-reliant on patient specific model parameters, and can therefore be applied to new patients with no elaborate medicinal analysis required.

The ABP, with the parameters used, generally results in the patient having lower glucose concentration levels than with the PID pancreas. The general glucose concentration could probably easily be increased by adjustment of the parameters, such that less insulin is given by the PID controller or via the bolus estimation of the ABP, thus making the PID pancreas and ABP perform close to identically. Also, the number of different parameters to tune introduces a lot of uncertainty as to whether the ABP is tuned optimally. It might be possible to increase it's performance significantly. Such uncertainty isn't as prevalent with the PID pancreas.

Meal detection is one of the major tasks given that a meal detection affects the behavior of the ABP a lot. If the meal detections were perfect, it is expected that the ABP would perform better than the PID pancreas. But it is unfortunately a difficult task, especially when the stochastic model is introduced. The two main factors are meal detection delay and false positives. Given the stochastic model, if no false negatives are wanted it seems impossible to avoid false positives. Regarding the delay of the meal detections, it holds that the faster the meal detection happens, the smaller the signal with which the super bolus estimates the bolus size becomes. So an instant meal detection might therefore not be optimal with the bolus size estimation method used in this report, or at least the bolus size estimation should only happen some time after consumption.

An important thing to keep in mind is that the presented results are based on simulations. Simulations that only account for meals and snacks, which are only consumed in specified time intervals. Many other things such as exercise and stress levels can affect the glucose concentration levels. Simply looking at the glucose concentration levels of the real patient from (10) indicate that real life glucose concentration concentrations often vary in much more complicated ways.

7 Conclusion

With the simulation method used, the ABP performs satisfactorily. Despite not clearly performing better than the PID pancreas, it does come close with the trial and error hand tuning used. And given it's number of parameters it is expected that it can be improved significantly. In the performed simulations specifically, it administers a little too much insulin resulting in lower glucose concentration levels than with the PID pancreas. This can easily be adjusted such that the glucose concentration levels generally would increase, probably coming close to those of the PID pancreas. The results are based on simulations with meals and random noise as the only factors, such that the applicability of the ABP to real patients is difficult to gauge. This should be seen as an attempt at a proof of concept in combining many different techniques in the field of developing an artificial pancreas.

8 References

[1] John Bagterp Jørgensen. Scientific Computing for Systems and Control - A Diabetes Case Study.

[2] Rebecca A. Harvey, Eyal Dassau, Howard Zisser, Dale E. Seborg, and Francis J. Doyle. Design of the Glucose Rate Increase Detector: A Meal Detection Module for the Health Monitoring System. *Journal of Diabetes Science and Technology*, 8(2):307–320, March 2014. URL: http://journals.sagepub.com/doi/10.1177/1932296814523881, doi:10.1177/1932296814523881.

[3] Desmond J Higham. An Algorithmic Introduction to Numerical Simulation of Stochastic Differential Equations. page 22.

[4] Richard I. G. Holt, J. Hans DeVries, Amy Hess-Fischl, Irl B. Hirsch, M. Sue Kirkman, Tomasz Klupa, Barbara Ludwig, Kirsten Nørgaard, Jeremy Pettus, Eric Renard, Jay S. Skyler, Frank J. Snoek, Ruth S. Weinstock, and Anne L. Peters. The management of type 1 diabetes in adults. A consensus report by the American Diabetes Association (ADA) and the European Association for the Study of Diabetes (EASD). *Diabetologia*, 64(12):2609–2652, December 2021. URL: https://link.springer.com/10.1007/s00125-021-05568-3, doi:10.1007/s00125-021-05568-3.

[5] Hyunjin Lee, Bruce A. Buckingham, Darrell M. Wilson, and B. Wayne Bequette. A Closed-Loop Artificial Pancreas Using Model Predictive Control and a Sliding Meal Size Estimator. *Journal of Diabetes Science and Technology*, 3(5):1082–1090, September 2009. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2769914/.

[6] Tobias K. S. Ritschel, Asbjørn Thode Reenberg, Emilie B. Lindkvist, Christian Laugesen, Jannet Svensson, Ajenthen G. Ranjan, Kirsten Nørgaard, Bernd Dammann, and John Bagterp Jørgensen. A one-size-fits-all artificial pancreas for people with type 1 diabetes based on physiological insight and feedback control. February 2022. Number: arXiv:2202.13338 arXiv:2202.13338 [cs, eess, math, q-bio]. URL: http://arxiv.org/abs/2202.13338.

[7] Pouya Saeedi, Inga Petersohn, Paraskevi Salpea, Belma Malanda, Suvi Karuranga, Nigel Unwin, Stephen Colagiuri, Leonor Guariguata, Ayesha A. Motala, Katherine Ogurtsova, Jonathan E. Shaw, Dominic Bright, and Rhys Williams. Global and regional diabetes prevalence estimates for 2019 and projections for 2030 and 2045: Results from the International Diabetes Federation Diabetes Atlas, 9th edition. *Diabetes Research and Clinical Practice*, 157:107843, November 2019. URL: https://www.sciencedirect.com/science/article/pii/S0168822719312306, doi:10.1016/j.diabres.2019.107843.

[8] Karl J. Åström and Richard M. Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton University Press, Princeton, 2008. OCLC: ocn183179623.

9 Appendix

9.1 Abbreviations

The following abbreviations are used:
ABP - Automatic Bolus Pancreas
CGM - Continuous Glucose Monitoring
CLS - Close Loop simulation
GRID - Glucose Rate Increase Detector
MSE - Meal Size Estimate
MVP - Medtronic Virtual Patient
OLS - Open Loop Simulation
ROC - Rate Of Change
TIR - Time In Range

9.2 MVP model parameters

Parameter	Value	Description	Unit
τ_1	49	Time constant	Min
τ_2	47	Time constant	Min
C_I	20.1	Insulin clearance	dL/min
p_2	0.0106	Inverse time constant	1/min
S_I	0.0081	Insulin sensitivity	(dL/mU)/min
GEZI	0.0022	Gluco	1/min
EGP_0	1.33	Endogenous glucose production	(mg/dL)/min
V_G	253	Glucose distibution volume	dL
τ_m	47	Meal absorption time constant	Min
τ_{sc}	5		Min

Table 4: Table of the standard parameters for MVPmodel

9.3 Subcutaneous glucose concentration vs glucose concentration

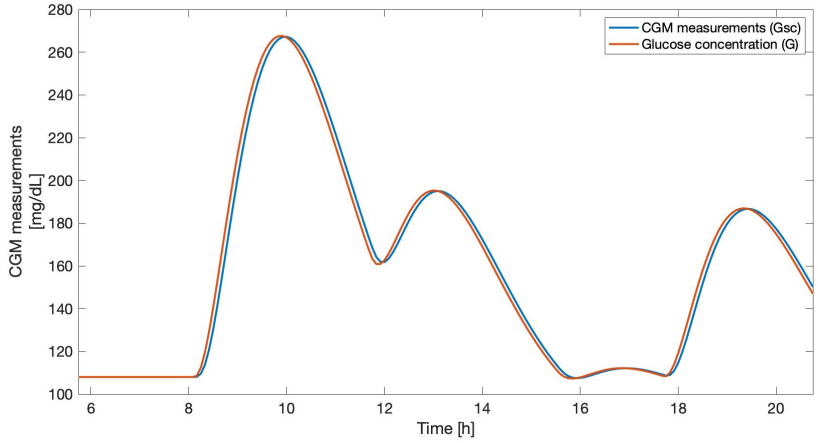


Figure 15: A exempt of CGM measurements and glucose concentrations from a simulation of a single patient. The only visible difference is a very slight delay of about 5 min. in the CGM measurements compared to the glucose concentration.

9.4 Sigmoid ramping function

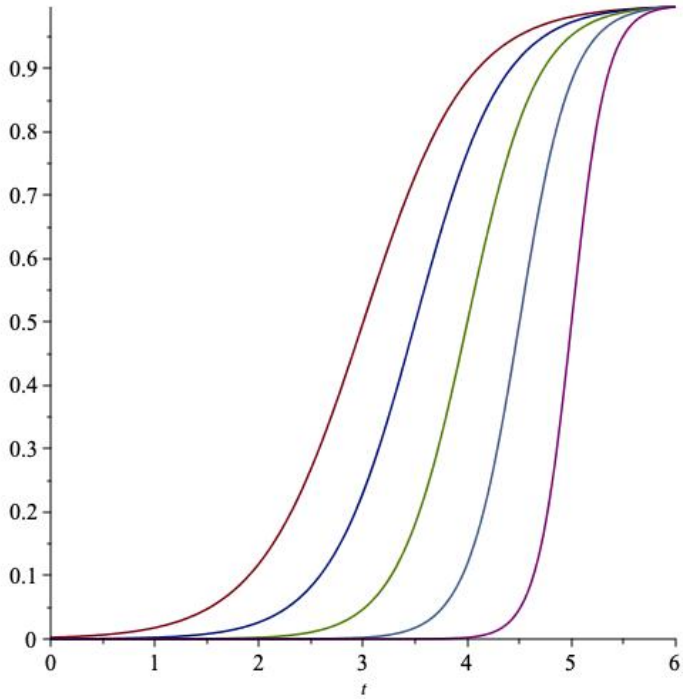


Figure 16: Sigmoid ramp plotted with $T_{halt} = 6$ and different ramping periods.

9.5 Euler method vs ode15s

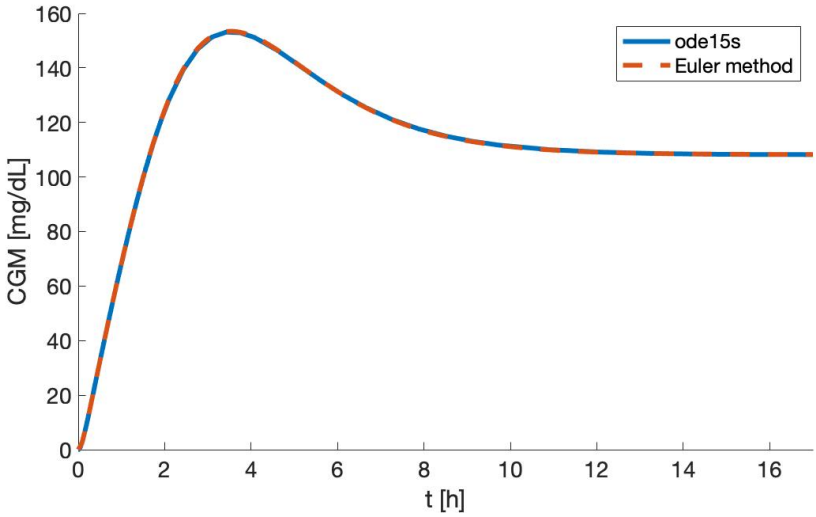


Figure 17: The CGM measurements from solving the MVP model using the Euler method and Matlab's ode15s. The glucose concentration starts at 0, the basal rate is a constant $u_{ba}(t) = 25.04$ and there is no bolus.

9.6 Controller gains for PID controller

The control parameters $K_P = 0.15$, $K_I = 0.00015$ and $K_D = 1$ are found via the method described in sec. (3.8). The images below show the iterative process of narrowing down the space of control parameters that are being tested for. Due to the non-linear behavior of the penalty function, the natural logarithm is taken of the penalty scores to see small differences better.

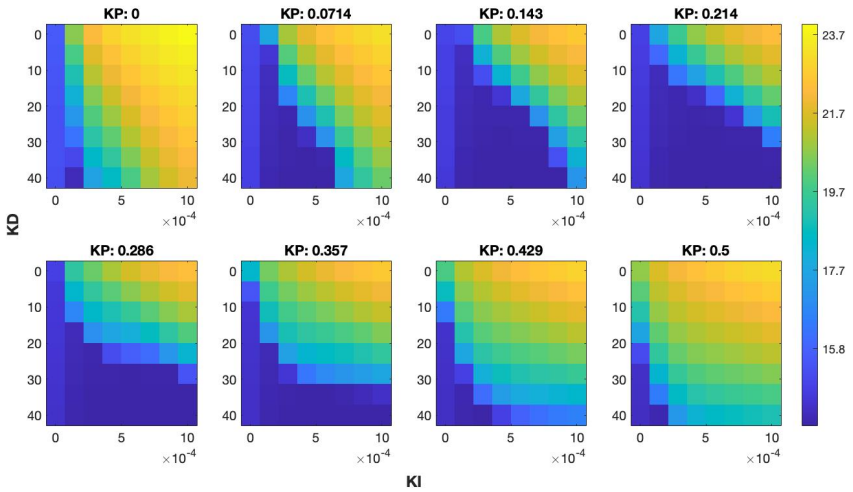


Figure 18: Minimum with penalty score of exp(15.7642) at KP = 0.21429, KI = 0.00057143, KD = 40 with index 4, 8, 5. Simulated for 7 days.

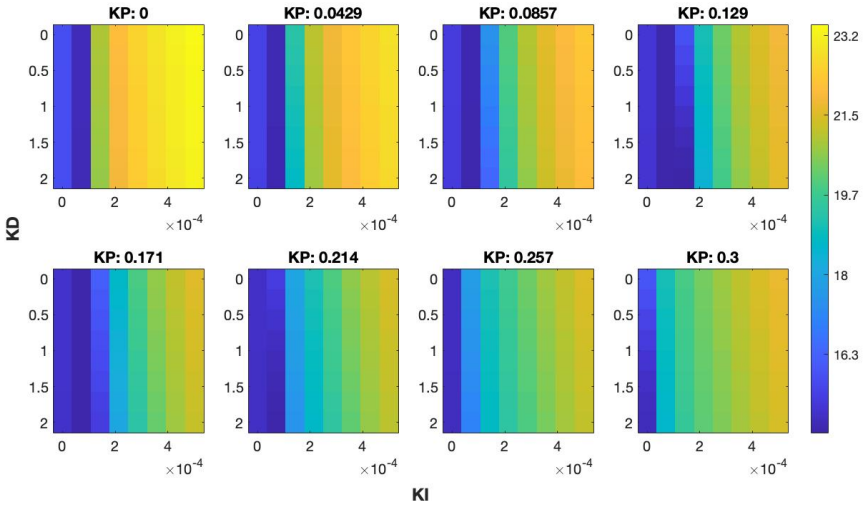


Figure 19: Minimum with penalty score of exp(16.2948) at KP = 0.12857, KI = 0.00014286, KD = 2 with index 4, 8, 3. Simulated for 7 days.

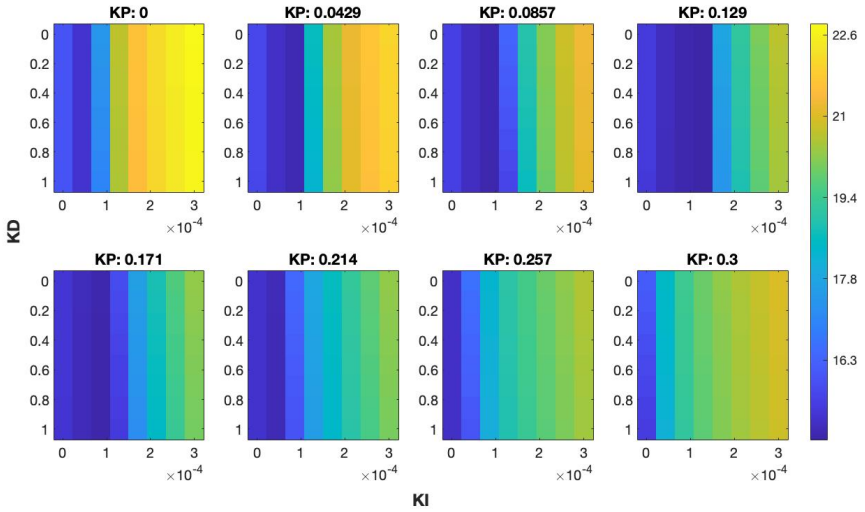


Figure 20: Minimum with penalty score of exp(16.2537) at KP = 0.12857, KI = 0.00012857, KD = 1 with index 4, 8, 4. Simulated for 7 days.

Using the stochastic model without measurement noise yields the following:

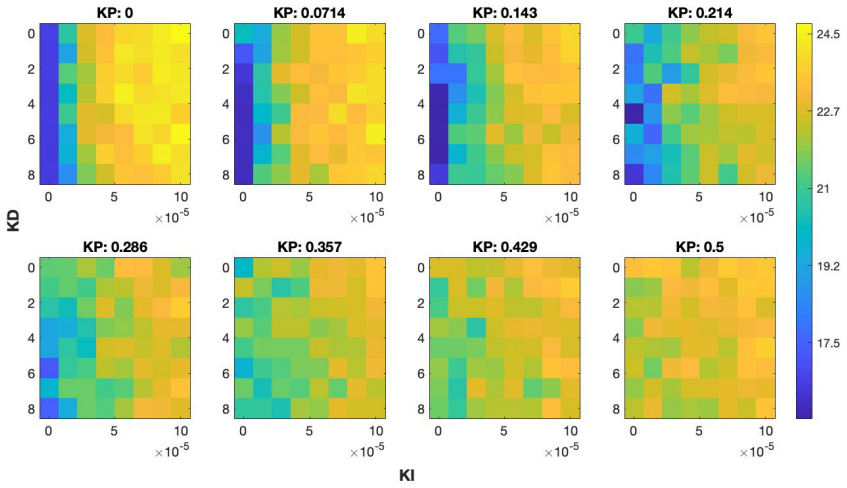
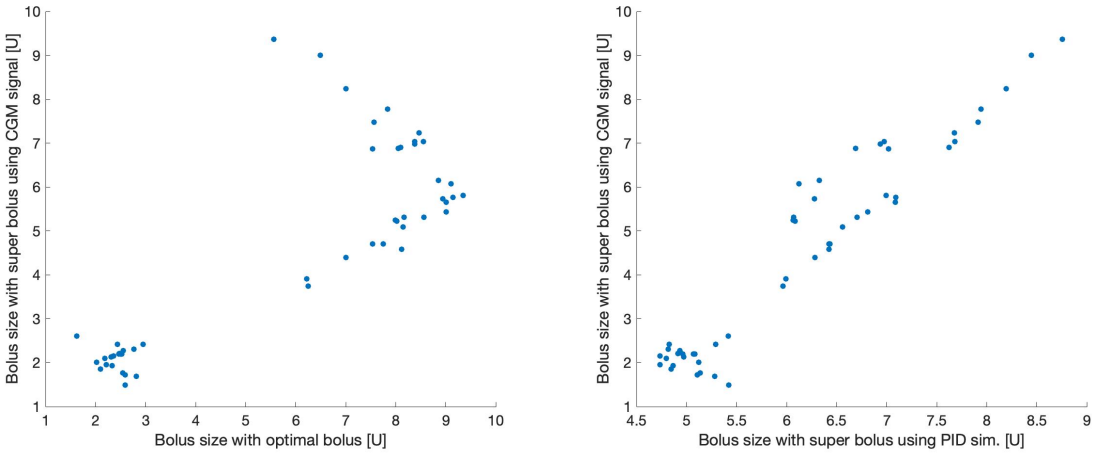


Figure 21: Minimum with penalty score of exp(17.4969) at KP = 0.21429, KI = 0, KD = 4.5714 with index 4, 5, 1. Simulated for 31 days. Simulation with stochastic model.

A clear difference with the integration gain K_I is visible. It looks like it almost needs to be as low as possible.

9.7 Bolus size covariance



(a) Optimal bolus vs CGM signal bolus size estimation.

(b) CGM signal vs PID simulation bolus size estimation.

Figure 22: Scatter plots of bolus size estimation methods.

	Optimal bolus	PID sim. bolus	CGM signal bolus
Optimal bolus	324763	98706	214334
PID sim. bolus	98706	50677	98516
CGM signal bolus	214334	98516	208572

Table 5: Covariance matrix of bolus size estimations.

9.8 Controller gains for ABP

The controller gains $K_P = 0$, $K_I = 0.00021$ and $K_D = 5$ for the ABP are found via inspection of the plots below:

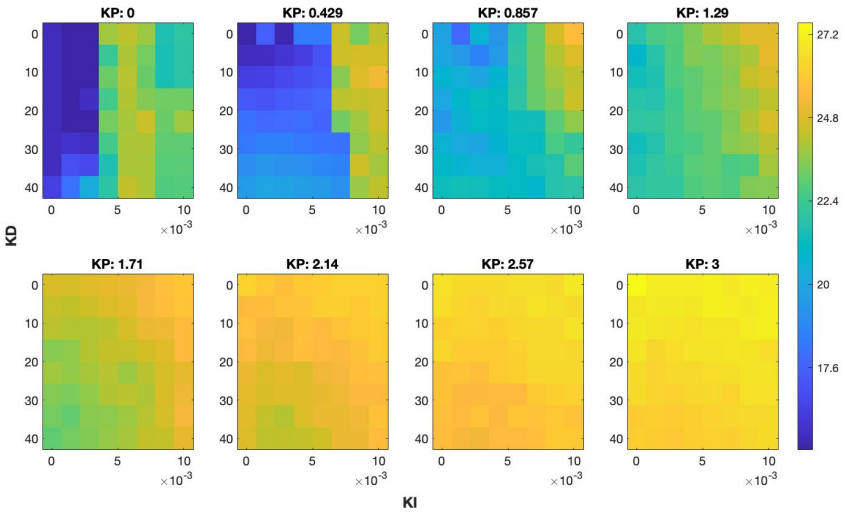


Figure 23: Minimum with penalty score of exp(17.621) at KP = 0, KI = 0.0014286, KD = 22.8571 with index 1, 5, 2. Simulated for 31 days. Simulation with deterministic model.

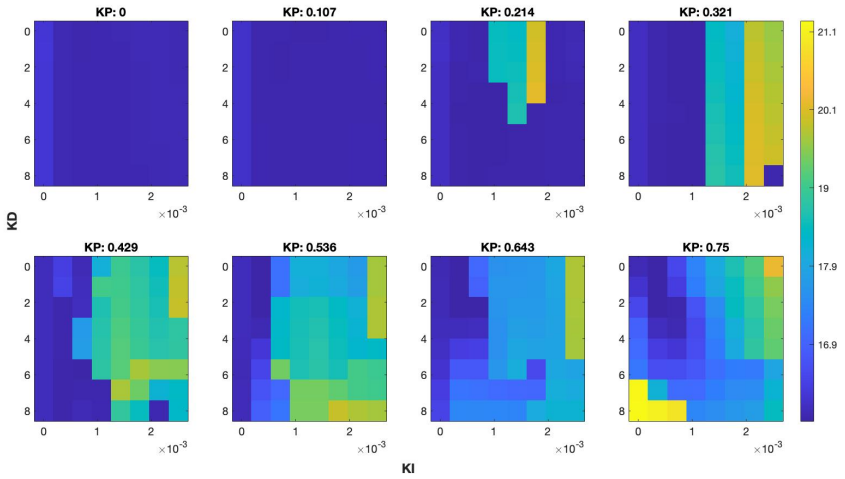


Figure 24: Minimum with penalty score of exp(16.8344) at KP = 0.64286, KI = 0.00071429, KD = 1.4286 with index 7, 3, 2. Simulated for 31 days. Simulation with deterministic model.

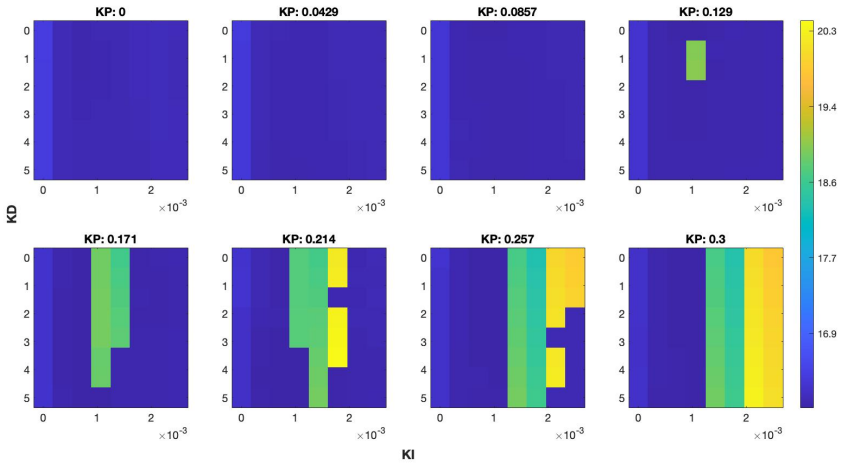


Figure 25: Minimum with penalty score of exp(16.8654) at KP = 0.17143, KI = 0.00071429, KD = 1.4286 with index 5, 3, 3. Simulated for 31 days. Simulation with deterministic model.

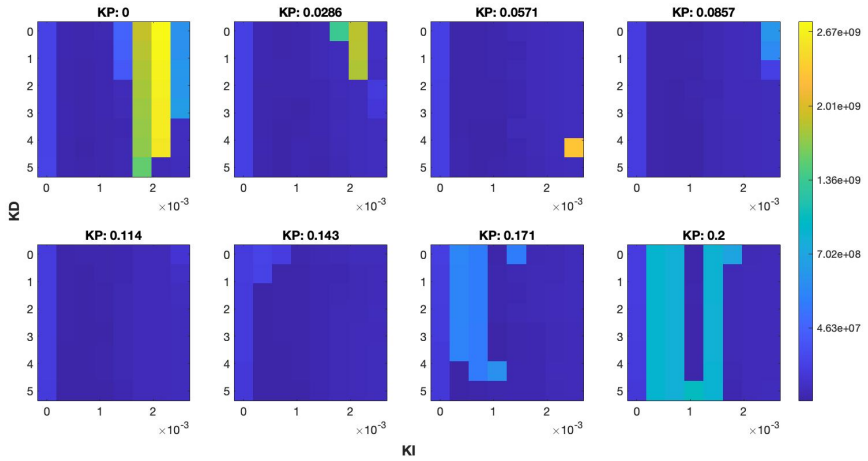


Figure 26: Minimum with penalty score of exp(17.6505) at KP = 0.11429, KI = 0.00071429, KD = 5 with index 5, 8, 3. Simulated for 31 days. Simulation with deterministic model.

Using the stochastic model without measurement noise yields the following:

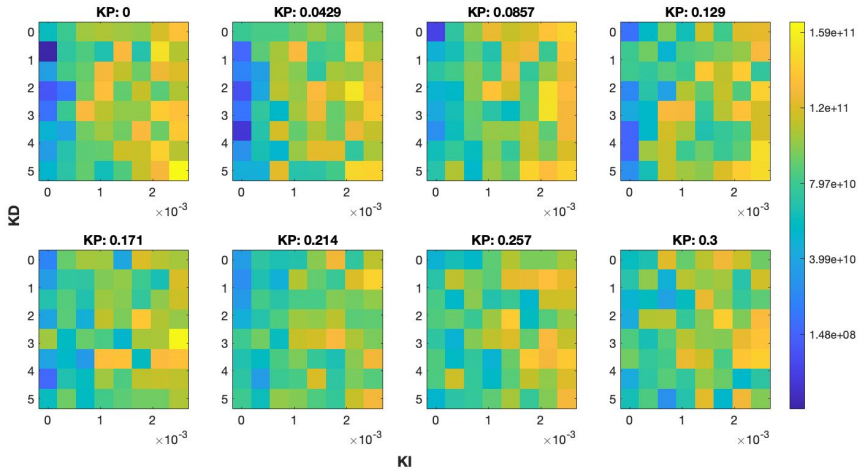


Figure 27: Minimum with penalty score of exp(18.8129) at KP = 0, KI = 0, KD = 0.71429 with index 1, 2, 1. Simulated for 31 days. Simulation with stochastic model.

Difficult to make out any structure apart from K_P and K_I probably needing to be in the low ranges.

9.9 TIR plots

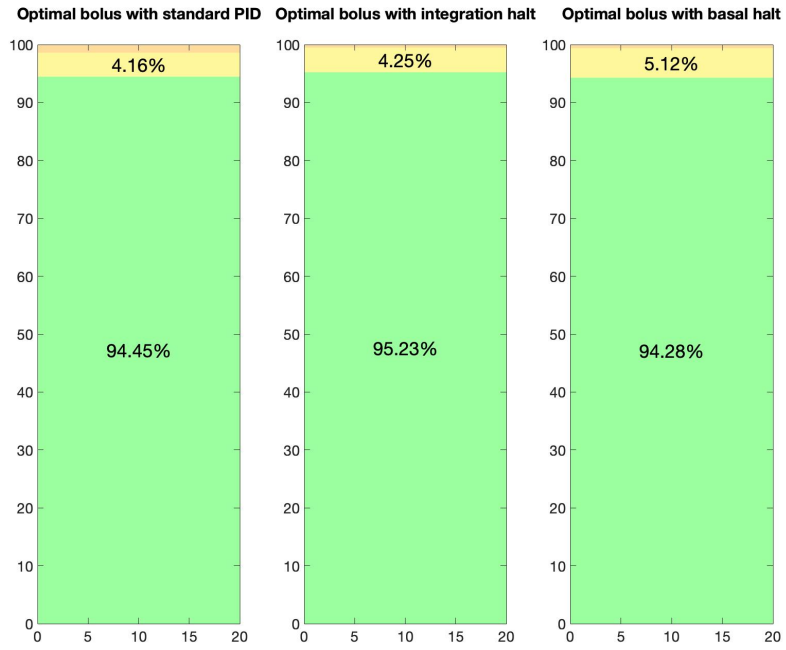


Figure 28: The TIR's of one patient using optimal bolus estimation and PID controller with and without integration halt.

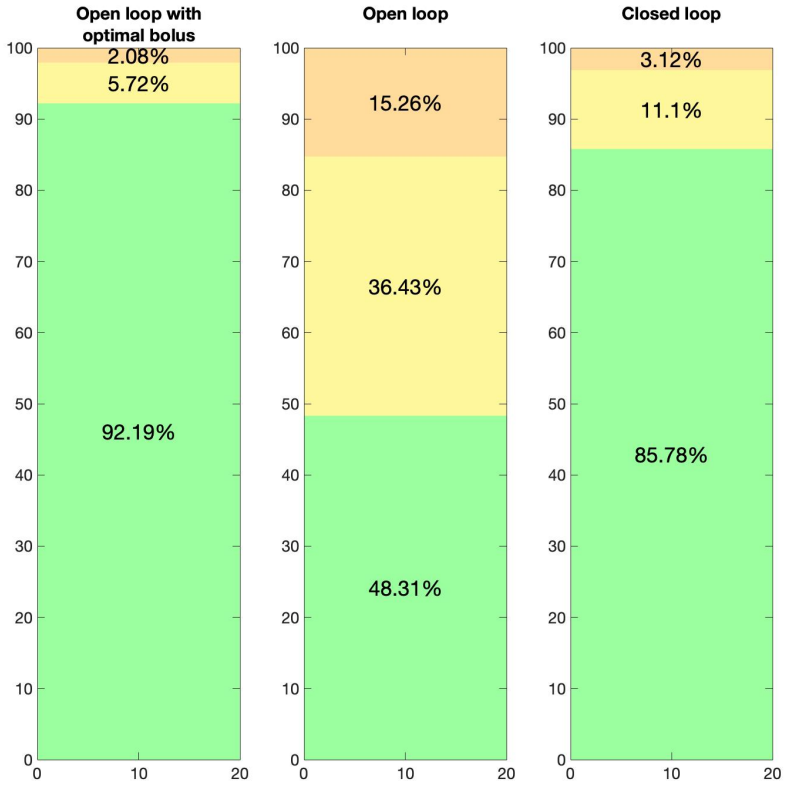


Figure 29: The TIR's of one patient using open loop and closed loop with and without optimal bolus estimation.

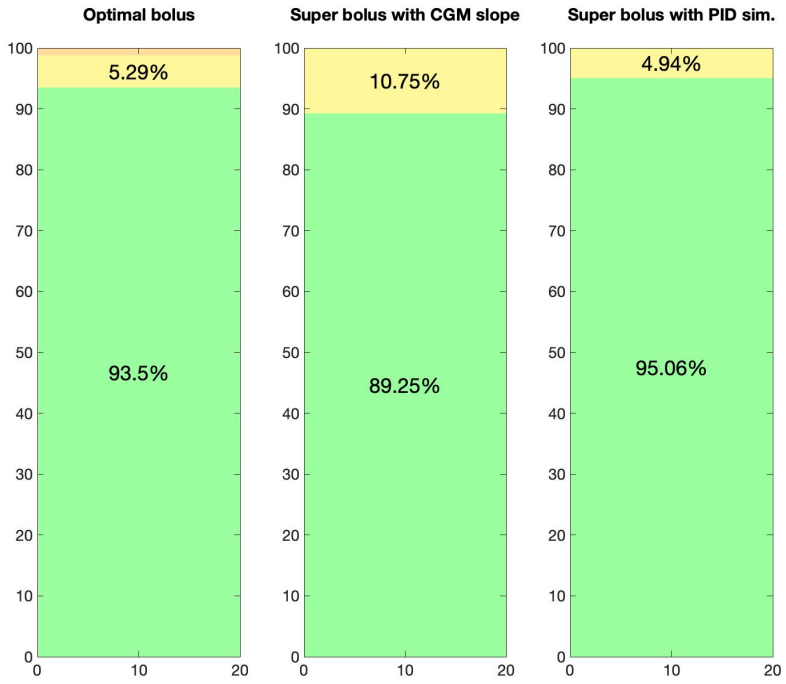


Figure 30: The TIR's of one patient using optimal bolus, super bolus with PID simulation and super bolus with CGM slope.

9.10 Simulations of 31 days

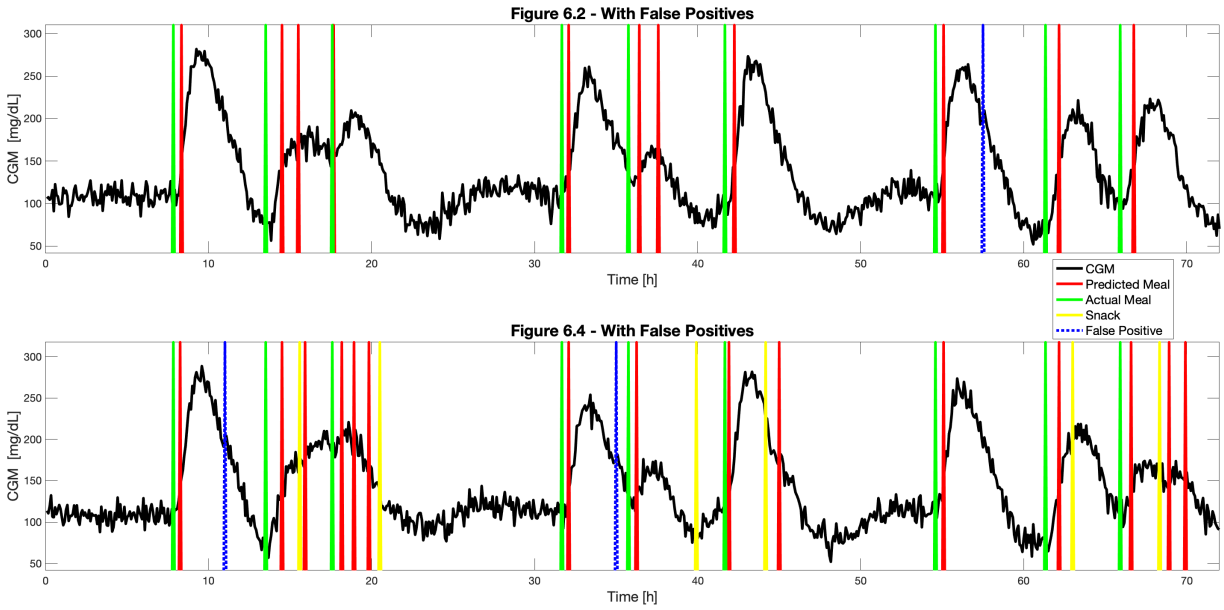


Figure 31: Simulation of 31 days with measurement noise

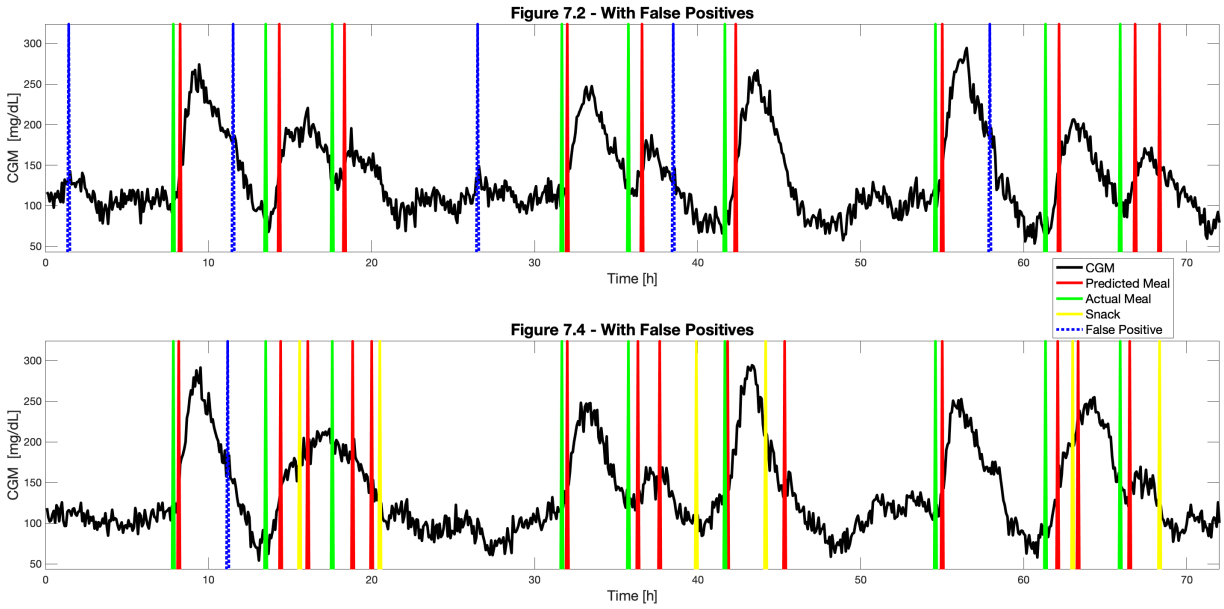


Figure 32: Simulation of 31 days, with Euler-Maruyama

9.11 Meal size estimation test

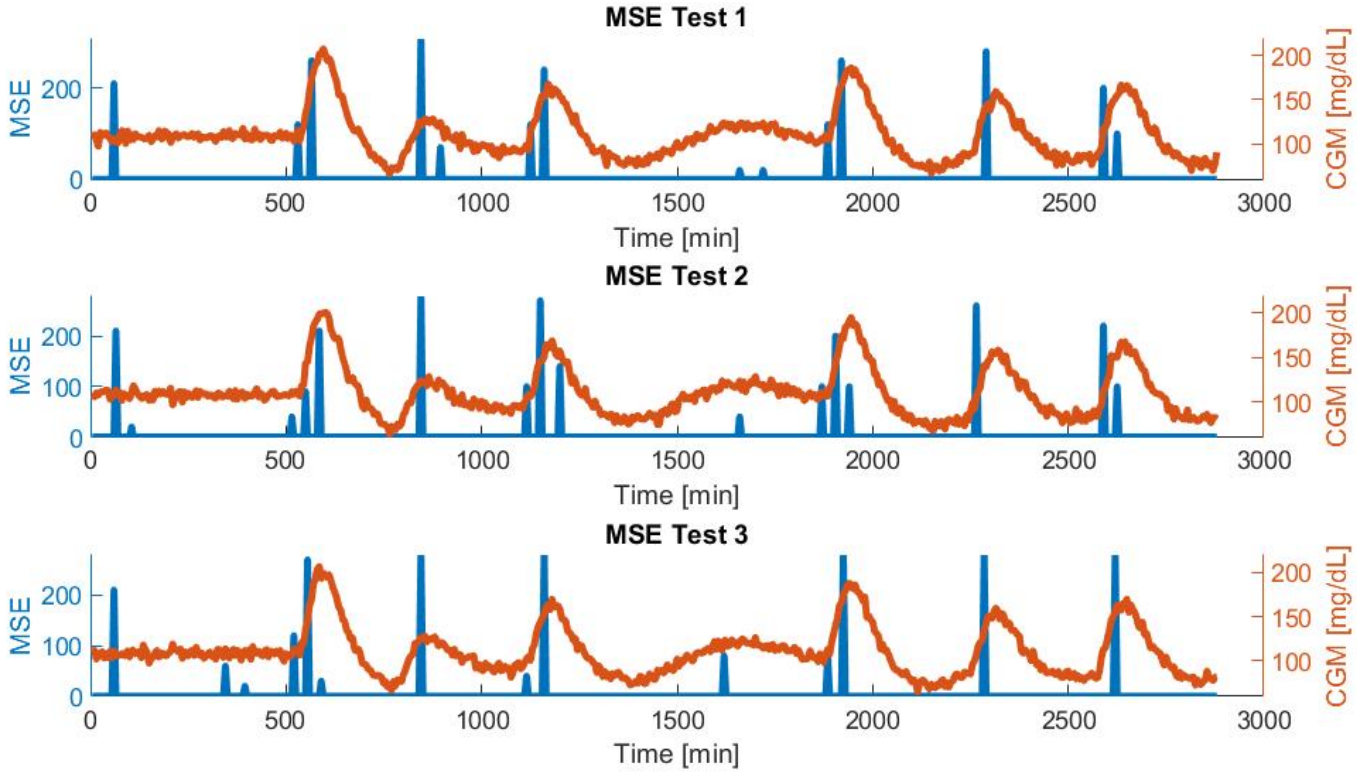


Figure 33: Meal Size Estimation test on three different simulation. One can see Meal estimations and CGM plotted for two days.

9.12 Final parameters

Name	ABP	PID	Unit	Function
K_P	0.1	0.15		Proportional controller gain.
K_I	0.0007	0.000005		Integral controller gain.
K_D	1	1		Derivative controller gain.
$U_{bo,\,max}$	10		U	Maximum allowed bolus size.
T_{ba}	3		h	Time period proportional with the super bolus computation.
T_{GRID}	3		h	Previous time period for which the GRID algorithm is receiving G_{sc} measurements.
ΔG	15			Used in the NoiseSpikeFilter in the GRID algorithm.
Δt	5		min	Used in the LowPassFilter in the GRID algorithm.
T_{meal}	2.5		h	Time period in which only one meal is allowed.
T_{dG}	40		min	Time period used for computing the mean of the ΔG_{sc}.
T_{halt}	3		h	Time period in which the basal rate ramped up from 0 during bolus titration.
T_{ramp}	30		min	Time period in which the basal rate is set to 0 during bolus titration.
α	0.25			Constant used in the super bolus computation.
β	0.5			Constant used in the super bolus computation.
γ	0.1			Constant used in the super bolus computation

Table 6: Parameters used for the simulations comparing the ABP and PID pancreas.

9.13 Code for GRID algorithm

```
function [GF,dGF,GRID]=GridAlgo(Gm,dG,dt,~,t)
%%%%%%%%%%%%%%%
%Function to calculate the Grid algorithm
%% input
%[Vector] Gm Measurement (Length: k)
%[int] dG (deltaG) Maximum allowable ROC
%[int] dt sampling period
%[int] ~ tauF Isn't used since we use specific value for tauF
%[Vector] t timevector (Length: k)
%% Output
%[Vector] GRID with length k either 0 or 1 at the i'th entry.
%% Best values for parameters
tauF=6;%minuttes
% den var 6 før
Gmin=130;%mg/dL
% Var før 130

Gmin3=1.5 ;%mg/dL/min
% den var 1.5 før

Gmin2=1.6;%mg/dL/min
% uændret

%% Inisializing
k=length(Gm);

dGF=zeros(1,k);
%% preprocessing section
GFNS=NoiseSpikeFilter(Gm,dG);

%% Low pass filter

GF=lowpassfilter(GFNS,dt,tauF,k);

%% Lagrangian interpolation

dGF(1)=GF(1);
dGF(2)=GF(2);
% dGF(1)=0;
% dGF(2)=0;
for i=3:k
    T1=(t(i)-t(i-1))/...
        ((t(i-2)-t(i-1))*(t(i-2)-t(i)));
    T2=(t(i)-t(i-2))/...
        ((t(i-1)-t(i-2))*(t(i-1)-t(i)));
    T3=(2*t(i)-t(i-2)-t(i-1))/...
        ((t(i)-t(i-1))*(t(i)-t(i-2)));
    dGF(i)=T1*GF(i-2)...
        +T2*GF(i-1)...
        +T3*GF(i);
end

%% GRID
GRID=zeros(1,k);
for i=3:k
    %if GF(i)>Gmin && (max((dGF(end-2:end)>Gmin3)) || max((dGF(end-1:end)>Gmin2)))
    LastThreeBool=dGF(i-2)>Gmin3&&dGF(i-1)>Gmin3&& dGF(i)>Gmin3;
    LastTwoBool=dGF(i-1)>Gmin2&& dGF(i)>Gmin2;
%     if GF(i)>Gmin && (max((dGF(i-2:i)>Gmin3))...
%             || max((dGF(i-1:i)>Gmin2)))
%         GRID(i)=1;
%     else
```

```
    if GF(i)>Gmin && (LastThreeBool||LastTwoBool)
        GRID(i)=1;
    else
        %Not neccesary since GRID is a vector of zeros.
        GRID(i)=0;
    end
end

end
```

9.14 Code Closed Loop Simulation Complete

```
function [T, X, Y, U, ctrlState] = closedLoopSimulationComplete(x0, tspan, D, p, ...
    simModel, observationModel, ctrlAlgorithm, ...
    ctrlPar, ctrlState0, simMethod, tzero, haltingiter, idxbo, rampingfunction, ...
    dg, dt, gridTime, mealTime, opts)
% CLOSEDLOOPSIMULATION Simulate a closed-loop control algorithm.
%
% DESCRIPTION:
% Perform a closed-loop simulation of a model-based control algorithm for
% given initial condition, control intervals, disturbance variables,
% parameters, simulation model, observation model, and control algorithm
% (including hyperparamters, control model, control model paramters, and
% initial controller state).
%
% REQUIRED PARAMETERS:
%   x0                  - initial state↙
(dimension: nx    )
%   tspan               - boundaries of the control intervals↙
(dimension: N+1   )
%   D                   - disturbance variables for each control interval↙
(dimension: nd x N)
%   p                   - parameters↙
(dimension: np    )
%   simModel            - simulation model          (function handle)
%   observationModel    - observation model         (function handle)
%   ctrlAlgorithm       - control algorithm         (function handle)
%   ctrlPar             - controller parameters
%   ctrlState0          - initial controller state↙
(dimension: nc)
%   simMethod           - simulation method         (function handle)
%   opts                - an options structure (must contain the field Nk)
%   tzero               - time where ramping function is 0
%   haltingiter         - time where integration is halted has basal is ramped
%   dg, dt              - parameters used in GRID
%   gridTime            - time period parsed to GRID at every time step
%   mealTime            - time period in which only 1 meal is allowed
%
% RETURNS:
%   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)
%
% AUTHORS
% Tobias K. S. Ritschel
% Asbjørn Thode Reenberg
% John Bagterp Jørgensen
%
% CHANGED BY
%
% Frederik Nagel
% Laurits Fog Balstrup
% Morten M. Christensen

% Initial time
t0 = tspan(1);

% Observed variables
y0 = observationModel(t0, x0, p);

% Integration halt time during bolus titration
tpause = 0;
```

```
% Determine the number of manipulated inputs
uDummy = ctrlAlgorithm(t0, NaN, NaN, ctrlPar, ctrlState0, tzero, tpause, haltingiter,↙
rampingfunction);

% Number of states and manipulated inputs
nx = numel(x0);
ny = numel(y0);
nu = numel(uDummy);
nc = numel(ctrlState0);

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

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

% Allocate memory
T = zeros( 1, N+1);
X = zeros(nx, N+1);
Y = zeros(ny, N+1);
U = zeros(nu, N  );
ctrlState = zeros(nc, N+1);

% Store initial condition
T(   1) = t0;
X(:, 1) = x0;
Y(:, 1) = y0;
ctrlState(:, 1) = ctrlState0;

% Copy initial condition
tk = t0;
xk = x0;
yk = y0;

% Initialize GRID
GRID = zeros(1,gridTime);

Dest = zeros(1,length(tspan));

% Maximum meal sizes allowed
maxMeal = 2000;

% bolus scalar
alpha = 0.25;
beta = 0.5;
gamma = 0.1;

% time period used for computing Gsc derivative
TdG = 6;

for k = 1:N
    % Times
    tkp1 = tspan(k+1);

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

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

    if k > gridTime
```

```
        [~,~,GRID]=GridAlgo(Y(k-gridTime:k),dg,dt,[],tspan(k-gridTime:k));
    end

    % If meal is detected and no meal has been detected in past
    % gridTime
    if k > mealTime && nnz(GRID) > 0 && nnz(Dest(k-mealTime:k)) == 0
        dkest = 1;
    else
        dkest = 0;
    end

    Dest(k) = dkest;

    % If meal is detected, halt integration for some iterations and
    % estimate bolus
    if dkest ~= 0
        dYmean = mean(Y(k+1-TdG:k)-Y(k-TdG:k-1));

        tpause = haltingiter;

        ubok = min(maxMeal,alpha*Y(k)*haltingiter*max([0,beta+gamma*dYmean]));
    else
        ubok = 0;
    end

    if yk < ctrlPar(5)
        ubok = 0;
    end

    % Decrement tpause until 0
    if tpause ~= 0
        tpause = tpause - 1;
    end

    % Compute manipulated inputs
    [uk, ctrlStatekp1] = ctrlAlgorithm(tk, yk, dk, ctrlPar, ctrlStatek, tzero,↙
tpause, haltingiter, rampingfunction);

    % Set optimal bolus
    uk(idxbo) = ubok;

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

    % Solve initial value problem

    [Tk, Xk] = simMethod(simModel, tspank, xk, [], uk, dk, p);

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

    % Observed variables at the next time step
    ykp1 = observationModel(tkp1, xkp1, p);

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

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