Progetto di

Model Based Software Engineering

Insulin Pump

Daniel Ferro

Edoardo Luciani

Matteo Lazzari

1 Il sistema

Questo documento descrive un possibile modello per Insulin Pump, cioè un sistema che simula il comportamento di pazienti con diabete di tipo due prima, dopo e durante i pasti.
Il microinfusore consente di infondere insulina a velocità variabile durante la giornata permettendo di creare un dose insulinica il più aderente possibile alle reali esigenze di ogni singolo individuo nei diversi momenti della sua vita quotidiana. Questo comporta da un lato la possibilità di controllare in maniera più precisa la glicemia nelle diverse fasi della giornata, migliorando il controllo metabolico globale. Dall'altro, il poter infondere insulina a velocità variabile consente di ridurre il rischio di ipoglicemia perché consente di infondere minor quantità di insulina in quei momenti della giornata dove il fabbisogno è minimo, come ad esempio durante la notte.

Abbiamo simulato e modellato il sistema attraverso le seguenti componenti in modo da verificare piu' casi possibili:

- Paziente, modella le caratteristiche di una persona in base all'insulina ed il glucosio presenti;
- Ambiente, modella il cibo ingerito dal paziente e i vari intervalli tra i pasti in una giornata;
- Pompa di insulina, modella il sistema che controlla l'insulina iniettata sulla base del glucosio del paziente, cercando di minimizzarne l'uso e tenere il paziente lontano da situazioni di ipo e iperglicemia.

2 Scenari Operativi

Gli scenari operativi per questo sistema sono tutte le possibili combinazioni delle caratteristiche di una persona, quali altezza, peso, età, genere; gli scenari variano poichè la pompa è pensata per essere utilizzata da una persona qualsiasi, cioè senza delle caratteristiche particolari.

2.1 Modellazione attraverso Modelica

Lo script verify.py simula il sistema della pompa generando in maniere pseudo-intelligente delle possibili caratteristiche che rappresentano un paziente come: età,altezza,peso,sesso.
Le caratteristiche quindi non vengono generate in maniera totalmente random. Abbiamo sviluppato una funzione che generà età,altezza,peso,sesso in funzione dell'età cosi da tagliare fuori dai test tutti quei modelli di pazienti che nellà realtà non sarebbero possibili.
Come: Età 50, Peso, 10Kg

Qui viene rappresentata la funzione di cui parliamo:

```
def calcola_(age,intervallo,pesi_1,pesi_2,altezze_1,altezze_2):
    if abs(age-intervallo[0])<abs(age-intervallo[1]): # calcolo a quale eta si avvicina di piu
        peso=int(np.random.uniform(pesi_1[0],pesi_1[1])) #assegno un random preso nella prima fascia di peso
        altezza=int(np.random.uniform(altezze_1[0],altezze_1[1])) #assegno un random preso nella prima fascia di altezze
    else:
        peso=int(np.random.uniform(pesi_2[0],pesi_2[1]))
        altezza=int(np.random.uniform(altezze_2[0],altezze_2[1]))
    return(peso,altezza)
```

3 Architettura del sistema

Abbiamo modellato le seguenti componenti:

- **mealgen.mo**, questa componente gestisce l'ingestione di cibo che verrà data in pasto al modello del Paziente. Il paziente effettuerà un pasto ogni 8 ore, di cui ogni pasto dura un'ora.
- **Pump.mo**, questa componente gestisce la strategia con la quale viene iniettata l'insulina al paziente. Controlla il livello di glucosio attualmente presente nel paziente e con delle formule calcola la dose di insulina da iniettare.
- **patient.mo**, Il paziente ha in input la dose di insulina da pump.mo e la dose di cibo da mealgen.mo e calcola il glucosio nel sangue attraverso delle equazioni.
- **Monitor.mo**, E' il monitor che controlla se il glucosio del paziente non scenda mai sotto i 50 mg/dL o superi i 200 mg/dL altrimenti il monitor dichiara il paziente morto e il test fallisce.

- **Connectors.mo**, E’ un modello che permette di creare il tipo che connette gli input agli output.
- **Ragmeal.mo**, Modella il rate appareance glucose con un valore delta compreso tra 10 e 30.

4 Requisiti del sistema

- **Requisiti Funzionali**
  - Il glucosio non dovrebbe mai scendere sotto i 50 mg/dL;
  - Il glucosio dovrebbe stare piu' vicino possibile ai 100 mg/dL.
  - La pompa di insulina deve continuare a funzionare anche con una percentuale di rumore

4.1 Requisiti non Funzionali

- L'insulina iniettata deve essere minimizzata;
- Il sampling time della pompa di insulina deve essere massimizzato.

4.2 Modellazione requisiti con Modelica

Abbiamo modellato i requisiti funzionali servendoci principalmente di tre modelli:
- Il modello del paziente, che attraverso le formule del paper calcola in ogni momento il livello di glucosio nel sangue.
- Il modello della pompa di insulina, che in base a quanto glucosio è già presente nel sangue del paziente, eroga insulina.
- Il monitor, che controlla se i requisiti di liveness e safety sono violati.

Per quanto riguarda i requisiti non funzionali, abbiamo creato uno script sinth.py per cercare quali parametri della strategia della pompa è possibile variare per ottenere una quantità di insulina da iniettare minima, ma tenendo comunque il paziente nei range ragionevoli di glucosio.

Nel nostro sinth.py iteriamo simulando il modello con una combinazione dei parametri "a" e "ref" della pompa di insulina (ControlloGlu). Nello script si prova a simulare variando la "a" tra 0,5 e 1 e "ref" tra 90 e 115 per il log_100 e "a" tra 0,1 e 1.2 e "ref" tra 60 e 160 per log_1000.

Con la variazione di questi parametri possiamo osservare le diverse quantità di insulina iniettate ad un paziente con la stesse caratteristiche.

5 Extra

Abbiamo introdotto una percentuale di rumore che va ad influire direttamente sui valori del paziente. Questo rumore influenza i dati per un intervallo che oscilla tra -10% e +10% del valore stesso.

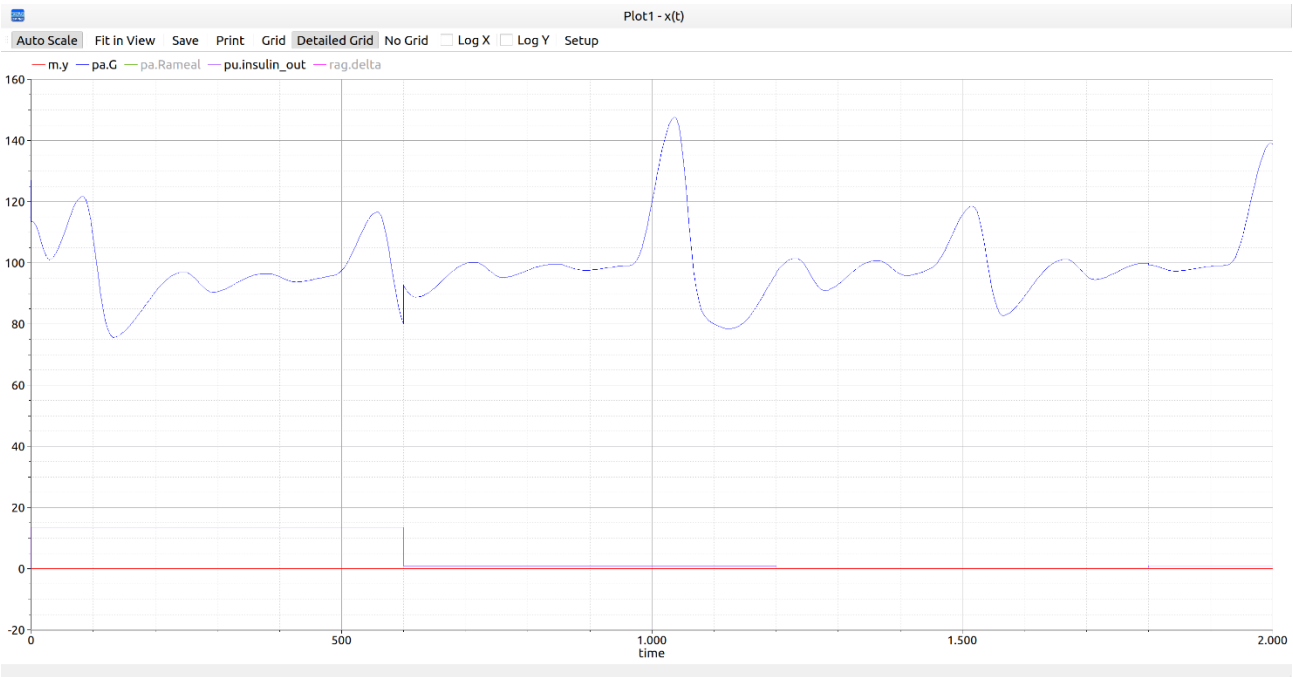
Per simulare che la pompa continui a funzionare nonostante il rumore abbiamo randomizzato i valori nel file verify.py e testato su 100/1000/10000 pazienti.

Dai risultati che abbiamo allegato in fondo siamo in grado di affermare che la pompa continui a funzionare correttamente.

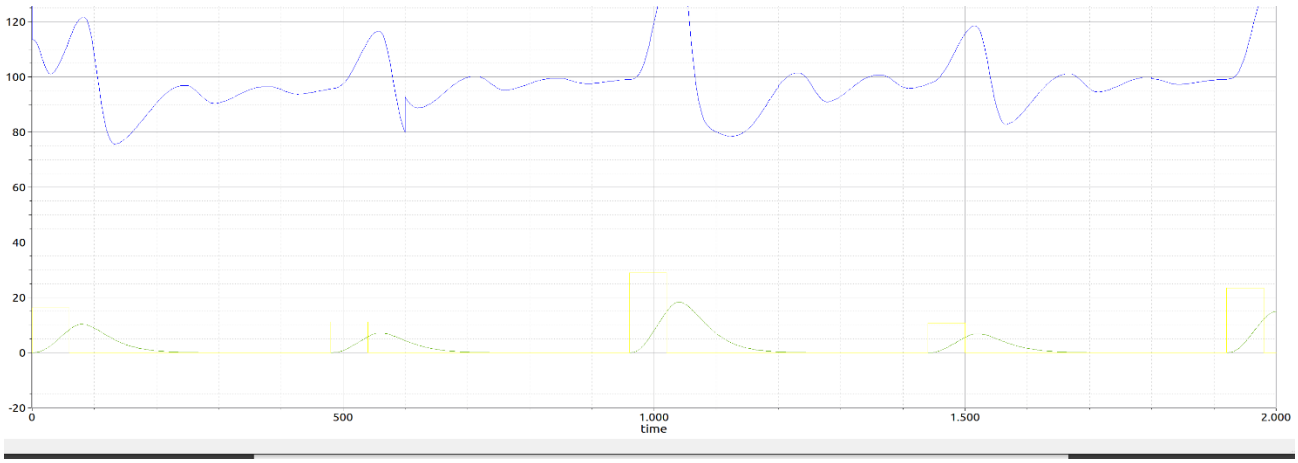
6 Risultati Sperimentali

6.1 run.mos

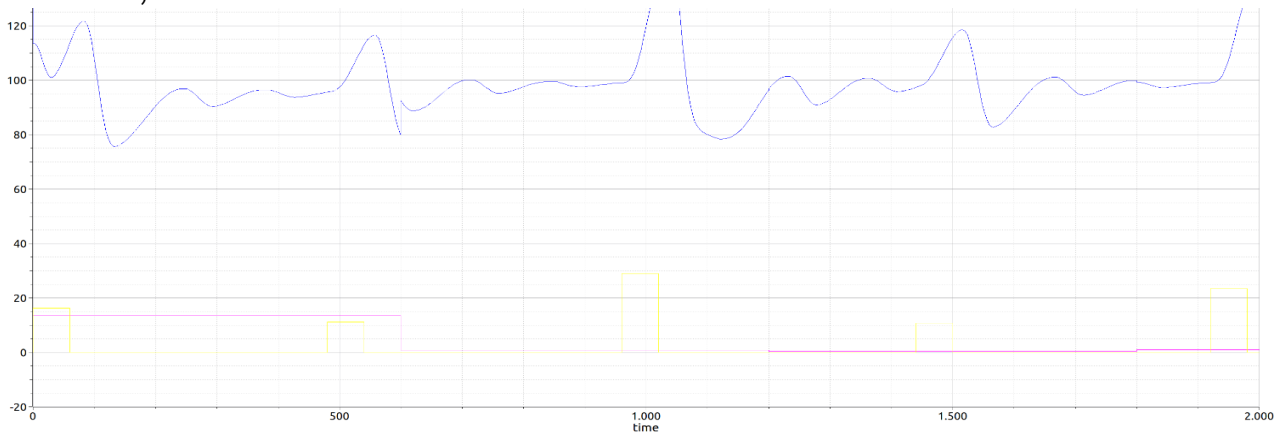
Glucosia e insulina



Glucosio, Rameal e Delta:



Glucosio,Insulina e Delta:



```
timeBackend = 0.029184446,
timeSimCode = 0.006862470000000001,
timeTemplates = 0.017243257,
timeCompile = 0.698598567,
timeSimulation = 0.06683794600000001,
timeTotal = 0.882036053
```

6.2 verify.py

Abbiamo effettuato test utilizzato il nostro verify.py eseguendolo su 100, 1000 e 10000 samples.

- 100 samples:

```
num pass = 100.0 , num fail = 0 , total tests = 100.0
pass prob = 1.0 , fail prob = 0.0
CPU Time = 5.098126173019409
```

- 1000 samples:

```
num pass = 1000.0 , num fail = 0 , total tests = 1000.0
pass prob = 1.0 , fail prob = 0.0
CPU Time = 53.52352952957153
```

- 10000 samples:

```
num pass = 10000.0 , num fail = 0 , total tests = 10000.0
pass prob = 1.0 , fail prob = 0.0
CPU Time = 523.3590009212494
```

In allegato i file contenenti i risultati, rispettivamente 100_log, 1000_log, 10000_log

6.3 synth.py

Abbiamo effettuato test utilizzato il nostro synth.py eseguendolo su 100, 1000 samples.

- 100 samples:

```
num pass = 100.0 , num fail = 0 , total tests = 100.0
pass prob = 1.0 , fail prob = 0.0
CPU Time = 878.5463573932648
```

- 1000 samples:

```
num pass = 1100.0 , num fail = 0 , total tests = 1100.0
pass prob = 1.0 , fail prob = 0.0
CPU Time = 8529.42873764038
```

In allegato i file contenenti i risultati, rispettivamente 100_synth_log, 1100_synth_log