Open-Source Project Plan:
User-Friendly Smart and Connected IoT for Engineers and DIY Hobbyists/Patients

Yong-Jun Shin, MD, PhD
Assistant Professor of Biomedical Engineering at the University of Connecticut

- Director of the Computational and Systems Medicine Lab https://csml.uconn.edu/
- Received MD and PhD in electrical engineering
- Received the National Science Foundation Smart and Connected Health Award titled "Distributed and Adaptive Personalized Medicine"
- Received the Microsoft Azure Research Award (in 2014 and 2015)
- Selected Publications
  1. Shin YJ. **"Edge Computing for Personalized Adaptive Artificial Pancreas"**. Advanced Technologies and Treatments for Diabetes (ATTD) 2019.
  2. Shin YJ. **"Digital Signal Processing and Control for the Study of Gene Networks"**. Scientific Reports. 2016; 6:24733. PubMed PMID: 27102828.
  3. Shin YJ. "**Parallel Computing for Adaptive Multi-Cellular Gene Network Modeling"**, Proceedings of the 1st IEEE Global Conference on Signal and Information Processing. 2013; 103-4 (invited article).
  4. Shin YJ, Sayed AH, Shen X. **"Using an Adaptive Gene Network Model for Self-Organizing Multi-Cellular Behavior"**. Conference Proceedings of IEEE Engineering in Medicine and Biology Society. 2012;5449-53. PubMed PMID: 23367162.

Purpose

Democratize engineering algorithms by providing a user-friendly smart and connected IoT platform.

Summary

Most IoT (the Internet of Things) platforms available today focus on sending, receiving, storing, analyzing, and visualizing data between things and the cloud. However, the true challenge of IoT comes from (1) how these things can interact with each other to form a complex, interconnected system, and (2) how each thing or entire connected system can dynamically learn and adapt to changes in the presence of uncertainty. In short, engineers and DIY (Do-It-Yourself) hobbyists want to build smart and connected IoT applications. However, this often requires learning cloud-native services that have a steep learning curve. I propose a user-friendly, nanoservice-enabled computing paradigm to build smart and connected IoT applications. API (Application programming interface) and SDK (Software Development Kit) will be developed so that engineers and DIY hobbyists can use either their favorite programming environment (e.g., MATLAB and LabVIEW) or a chatbot (e.g., Skype) to easily build smart and connected IoT applications. The proposed approach will also provide a variety of well-tested engineering algorithms, for instance the PID (Proportional-Integral-Derivative) and adaptive MPC (Model Predictive Control) algorithms, as function nanoservice (e.g., Azure Function) available anytime anywhere. This "democratize engineering algorithms" approach will also benefit proactive diabetic patients who are currently building their own DIY artificial pancreas to meet individual needs. Creating a safe and secure platform that also enables customization and sharing is critically needed for these patients. In this regard, the approach developed for smart and connected IoT applications in general can be further extended to smart and connected personalized artificial pancreas and other healthcare IoT applications.

Motivation and Background

Computing plays a critical role in engineering. It is used to simulate physics (e.g., fluid dynamics) and to execute signal processing and control algorithms (e.g., robot control). Engineers and DIY (Do-It-Yourself) hobbyists interested in engineering applications like home automation often do not write code from scratch. They make use of ready-made code and user-friendly computing environments such as MATLAB [1], LabVIEW [2], or Arduino IDE (integrated development environment) [3]. Although these environments have been extremely useful, they are facing a new challenge recently. As they are designed for isolated, unconnected applications such as embedded systems, they do not fit properly into the paradigm of today's open and connected world often referred to as the Internet of Things or IoT [4].

Most IoT platforms available today such as Amazon Web Services (AWS) IoT [5], Microsoft Azure IoT Hub [6], and Google IoT Core [7] focus on receiving or sending data, from or to things across the internet in a

dependable and scalable way. The data can also be stored, analyzed, or visualized. However, the true challenge of IoT comes from (1) how these things can interact with each other to form a complex, interconnected system, and (2) how each thing or entire connected system can readily learn and adapt to changes over time. In other words, engineers and DIY hobbyists want to build smart and connected IoT applications. This requires not only currently available IoT service but also integrating other cloud-native services to achieve intelligence and connectedness. Learning these cloud-native services is not easy because cloud computing is a new and fast-evolving technology often not taught at schools [8]. Furthermore, since it is a computer science (CS)-oriented subject, it has a steep learning curve for non-CS engineers and DIY hobbyists. In this regard, I propose a user-friendly, nanoservice-enabled computing paradigm to build smart and connected IoT applications. API (Application programming interface) and SDK (Software Development Kit) will be developed so that engineers and DIY hobbyists can use either their favorite programming environment (e.g., MATLAB) or a chatbot (e.g., Skype) to easily build smart and connected IoT applications (**Figure 1**).

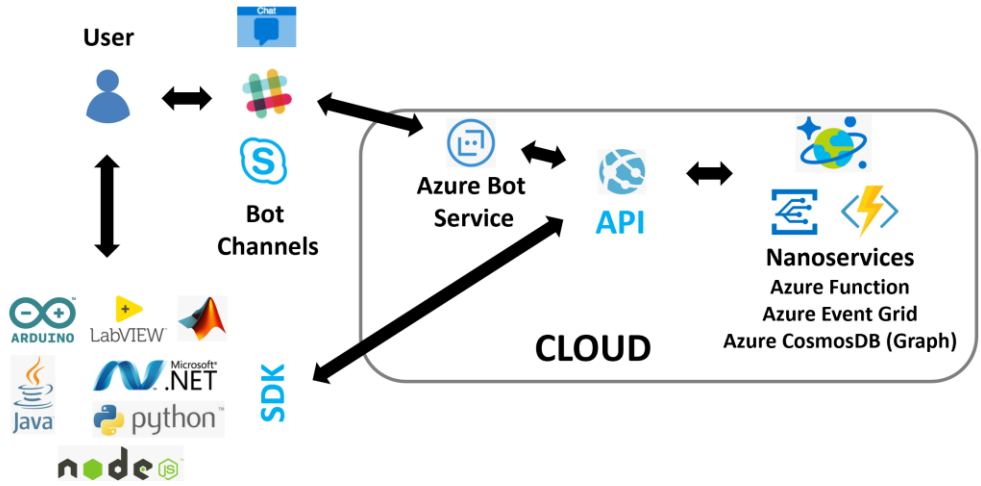

Figure 1. A user-friendly, nanoservice-enabled computing paradigm to build smart and connected IoT applications.

Smart and connected IoT applications demand high data granularity. For instance, consider the array of data that an autonomous car would generate. This data might need to be cross-referenced with other cars, pedestrians, street lamps, weather, traffic, and even internal datasets to understand relationships between energy efficiency and time of the day. This granularity of the data could even enable measuring the performance of a braking system during specific weather conditions. Although the state-of-the-art microservice computing paradigm enables increased flexibility and scalability compared to traditional monolithic approaches [10, 11], the fundamental components of computing, (1) variables (that store data), (2) functions (that updates data), and (3) event-handling (that couples variable updates with functions) are still tightly-coupled within each microservice. I propose to use a cloud-native graph database to provide data nanoservice (e.g., Azure CosmosDB graph database) that achieves desired granularity. A platform where engineers can make well-tested engineering algorithms available as function nanoservice (e.g., Azure Function) will be created that can be accessed anytime anywhere. Data and function nanoservices can be coupled using cloud-native event-handling nanoservice (e.g., Azure Event Grid). Function nanoservice can also be deployed to a local device using edge computing (Azure IoT Edge) [12] so that it can be executed even when the device is offline or has intermittent connectivity to the cloud. Function nanoservice-enabled engineering algorithms can be used by DIY hobbyists with proper guidance and education even if they do not have a deep understanding of these algorithms. It is like how a layperson without in-depth knowledge of mechanical engineering can drive a car after completing driving lessons. This “democratize engineering algorithms” approach can also benefit proactive patients who are currently building their own DIY artificial pancreas (the Open Artificial Pancreas System Project [13]) to meet individual needs. DIY artificial pancreas raises safety and security concerns because it is often used without supervising clinician or researcher. Therefore, creating a safe and secure platform that also enables customization and sharing is critically needed. In this regard, the approach developed for smart and connected IoT applications in general can be applied to medical IoT (IoMT) applications, especially smart and connected artificial pancreas. The proposed “barrier-lowering” approach will also be augmented by developing online MOOCs (Massive Open Learning Courses) freely available to the public.

Microservice/5G/edge computing-enabled IoT

Microservices are a state-of-the-art software development technique to decompose software into independent, loosely-coupled small (micro) services, which can easily be changed or replaced [10, 11]. In contrast, monolithic applications are difficult to maintain or revise as their components are tightly coupled. Microservices use modern web service technology. This means that they are interoperable across the internet and can talk to each other regardless of programming language, operating system, or development environment used. Microservices can be reliable and scalable when managed by orchestration platforms such as Kubernetes [14] and Microsoft Azure Service Fabric [15] that use the latest distributed computing technoloiges. These cloud-native platforms can be exploited to build an IoT platform and multiple cloud providers are already providing microservice-enabled, fully-managed, reliable, and scalable IoT platforms.

In addition to microservices, a new emgering communication technology 5G may also drive the future innovations of IoT. Multiple carriers plan to launch 5G service as soon as early 2019 [16] and it will provide unprecedented high data rate (peak data rate: 20 Gb/s), reduced latency (less than 1 millisecond), and device connectivity (1 million connections/km^2) for IoT. However, it is important to realize that many engineering applications such as chemical processes are operated at a slow pace and high computation/communication speed and sub-millisecond latency are not always required. For example, the basal insulin infusion rate of the artifical pancreas needs to be computed every 5 to 10 minutes for diabetic patients and this still turns out to be much more effective than maunal adjustments made by patients a few times a day. Furthermore, IoT edge computing can extend the power of cloud computing to local edge devices and these devices can run even when they are offline or have intermittent connectivity to the cloud, thus resolving communication and potential security issues.

Nanoservice-enabled IoT

Computing involves variables that store data and functions that update the values of these variables. Functions can also interact with each other in an event-driven fashion. For example, when “buttonClicked” function is triggered by a button-clicking event, “sendEmail” function can be executed. Modern smart and connected systems (e.g., smart and connected health, smart grid, etc.) should perform real-time learning, decision-making, and control. However, often the variables and functions of individual system components and their relationships cannot be clearly defined in advance. In addition, the event-driven interactions among these components may dynamically change over time. Therefore, hard-coding or tight-coupling variables, functions, and event-handling of individual system components becomes cumbersome in the presence of such uncertainty. Although a microservice approach enables increased flexibility and scalability compared to traditional monolithic approaches, variables, functions, and event-handling are still tightly-coupled within each microservice. In this context, I am exploring serverless nanoservices as a new computing paradigm for smart and connected IoT, which provides desired granularity and flexibility. The term “serverless” indicates that these services are fully-managed by cloud service providers and the burden of cloud infrastructure (server) management and capacity planning decisions are completely unloaded from the developer [19].

1. Data nanoservice: A graph database is a database designed to treat the relationships between individual data as important as the data itself [20]. Unlike other types of databases such as a relational (table) or document database, the data of a graph database is represented by a graph consists of vertices and edges. A vertex is an entity such as a thing, person, place, object, or relevant piece of data that can have its own properties. An edge shows the relationship between two vertices and can also have properties like vertices. Graph databases are ideal for smart and connected IoT as they are well suited to represent heavily inter-connected things with dynamically changing relationships and a high degree of granularity. Fully-managed cloud graph database services such as Azure Cosmos DB [21] and AWS Neptune [22] can provide serverless data nanoservice that can store variable data as independent vertices with flexible edges. Data nanoservice can be consumed to dynamically add, update, or remove IoT data in real time.
2. Function nanoservice: Azure Functions [23], AWS Lambda [24], Google Cloud Functions [25], and IBM Cloud Functions [26] are the examples of cloud services that can provide function nanoservice. These cloud services enable developers to run serverless functions (e.g., a control algorithm) that scale with high availability. These functions can be automatically triggered from other functions or services like event-handling nanoservice. Developers pay only for the compute time functions consume and there is no charge when they are not running.

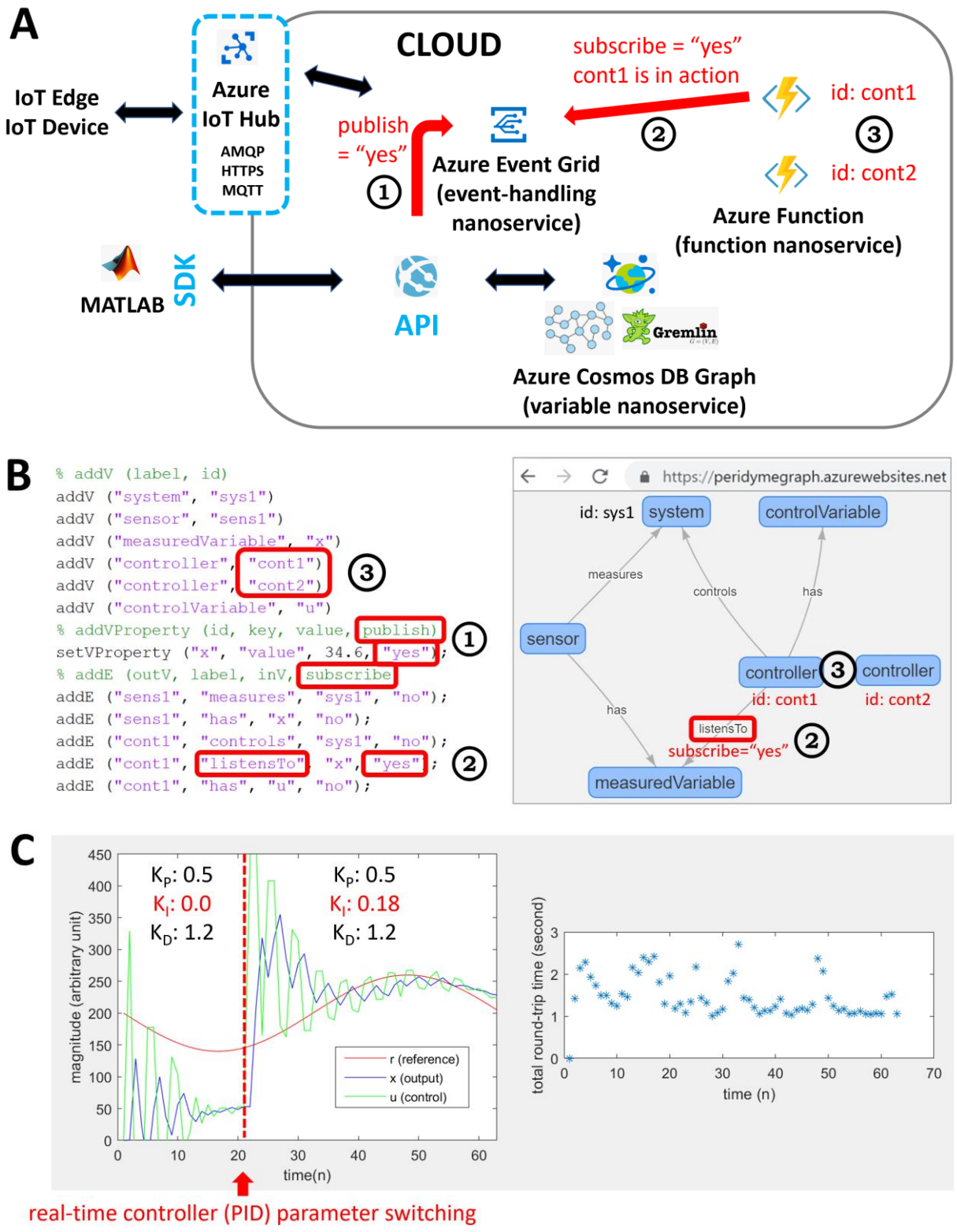

Figure 2. Nanoservice-enabled real-time control switching using MATLAB. (A) **peridyme API** and **peridyme SDK** enable MATLAB code to easily create cloud-native graphs that are coupled with function and event-handling nanoservices. (B) MATLAB code and GraphExplorer webpage example (web browser used: Google Chrome version 70.0.3538.110). (C) Controller parameter switching simulation. When the PID (proportional-integral-derivative) controller (id: cont1-PID) parameters $K_P/K_I/K_D$ are switched from 0.5/0.0/1.2 to 0.5/0.18/1.2 at around time 20, performance is improved (less steady-state error) due to increased K_I value.

3. Event-handling nanoservice: Azure Event Grid [27], Amazon Simple Notification Service [28], Google Cloud Pub/Sub [29] are the examples of fully-managed event-routing service that can provide event-handling

nanoservice at massive scale. Event publishers are decoupled from event subscribers using a publish/subscribe model.

Why now?

The three serverless nanoservices from Microsoft became generally available recently:

- Azure Cosmos DB graph database service became generally available in February 2018.
- Azure Event Grid became generally available in January 2018.
- Azure Functions became generally available in November 2016.

Competition / alternatives

As mentioned earlier, most IoT platforms available today such as Microsoft Azure IoT Hub, AWS IoT, and Google Cloud IoT focus on receiving, sending, storing, visualizing, and analyzing data between things and the cloud. As the focus is not on "connectedness" among things and integration of engineering algorithms, it is hard for engineers and DIY hobbyists to build smart and connected IoT applications using these platforms.

Intellectual property

This is an open source project available under the MIT license and no intellectual property will be claimed. Community-driven fast adaptation and constant innovation will sustain competitive advantage.

Preliminary work #1: Using MATLAB to consume nanoservices

Azure Cosmos DB is a Microsoft's globally distributed, serverless graph database service. It supports open source Apache Tinkerpop's graph traversal language, Gremlin [30], to create graph entities and to perform graph query operations. However, this means that engineers and DIY hobbyists need to learn (1) Azure Cosmos DB and (2) Gremlin. In this preliminary work, it is shown how proposed **API** and **SDK** enable MATLAB code to easily create cloud-native graphs that are coupled with function and event-handling nanoservices.

Figure 2A shows Azure IoT Hub, a cloud platform service that can securely connect, monitor, and manage billions of devices to develop IoT applications. It supports multiple communication protocols, including AMQP, HTTPS (HTTP over an encrypted SSL/TLS connection), and MQTT. Using Azure IoT Hub and **peridyme API**, MATLAB code can talk to other cloud-native services such as Azure Cosmos DB. **peridyme API** developed for this preliminary work to create graph entities and to perform graph query operations can also be directly accessed at https://peridymeapi2.azurewebsites.net/ (**Figure 3**). Currently, there are 11 services available as shown in the figure and each service (e.g., addV) can be tried on the web page. When Azure IoT Hub is not used for simplicity, MATLAB code can talk to **peridyme API** directly. **Figure 2B** shows how MATLAB code can add a vertex with a label of "system" and a unique identifier of "sys1" to the graph already created. Label can be considered as classifier or type. Addition of new vertices and edges can be visualized using a web browser by accessing GraphExplorer available at https://peridymegraph.azurewebsites.net/ (**Figure 2B)**. There are 6 vertices in the graph: sys1 (system), sens1 (sensor), x (measuredVariable), cont1 (controller), cont2 (controller), and u (controlVariable). There are also 5 edges with labels (shown within double quotation marks): (1) sensor "measures" system, (2) sensor "has" measuredVariable, (3) controller "controls" system, (4) controller "has" controlVariable, and (5) controller "listensTo" measuredVariable. Finally, a vertex property "value" with 34.6 is added to measuredVariable vertex (id:x) to store the value of the variable x. For this operation "setVProperty" is used and if one of the parameters "publish" is "yes" (① in **Figure 2A**) then the x value is published to Azure Event Grid. "addE" is used to add an edge between two vertices and the four required parameters are outV (starting vertex), label (edge label), inV (ending vertex), and subscribe. **Figures 2A** and **2B** show that controller "listens to" any change in published x value by making one of the available controllers ("cont1" and "cont2") coupled with corresponding Azure Functions (③ in the figures) to subscribe to Azure Event Grid (② in the figures). In summary, when sensor data x is updated Azure Function "cont1" will be triggered and compute the new value of u (controlVariable).

When controlling systems whose dynamics change with time or operating condition, (1) a controller with different gains (or controller parameters) or (2) a set of diverse types of controllers might be needed. The proposed cloud-native nanoservice approach can provide virtually an unlimited number of control options while control switching can be done in real time without down time. **Figure 2C** shows an example of control simulation

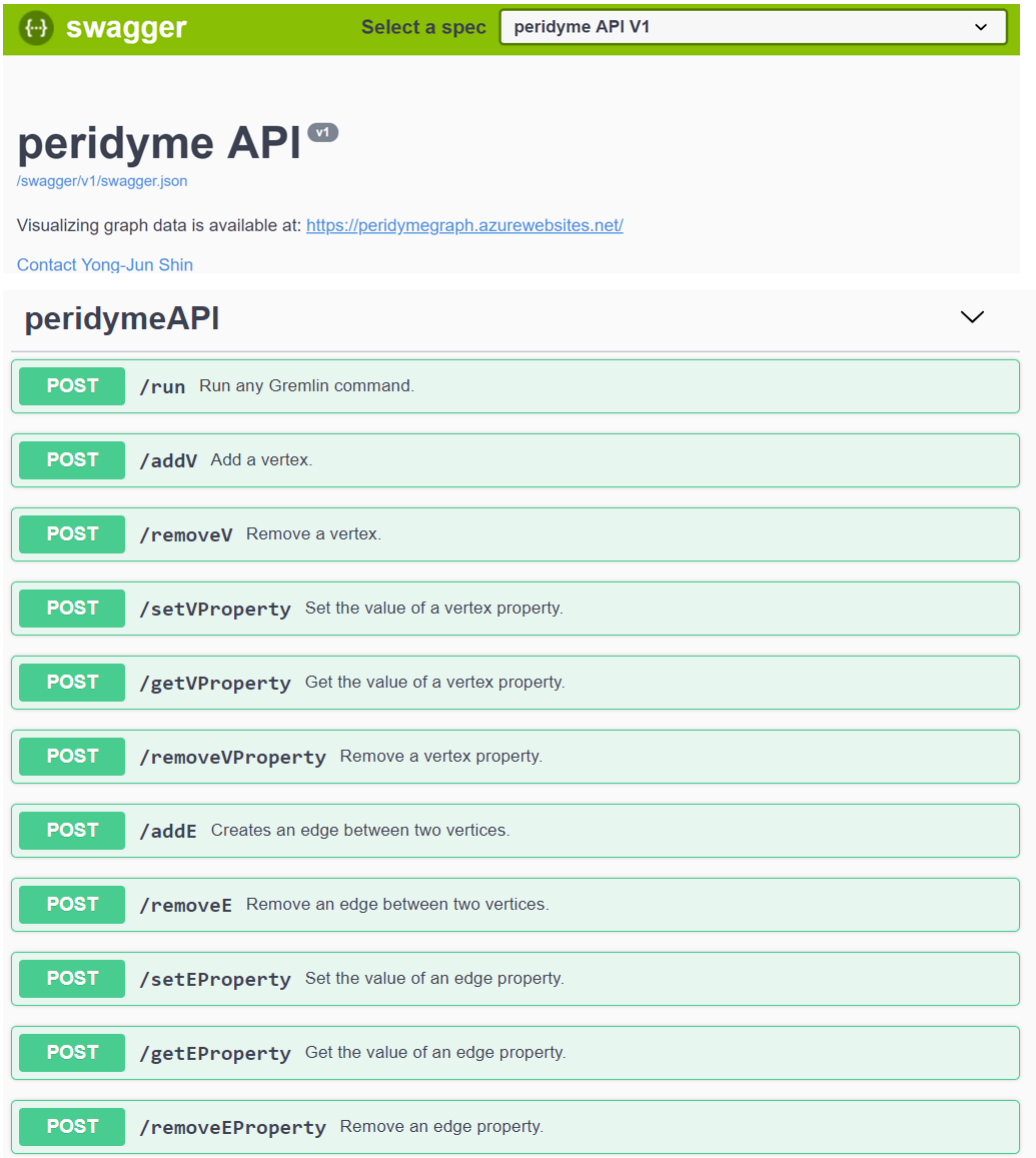

Figure 3. peridyme API developed for the preliminary work. Each service (e.g., addV) can be tried on the web page (web browser used: Google Chrome version 70.0.3538.110).

using the first approach (r: reference, x: measured output, u: control signal). When the PID (proportional-integral-derivative) controller (id: cont1-PID) parameters $K_P/K_I/K_D$ are switched from 0.5/0.0/1.2 to 0.5/0.18/1.2 at around time 20, performance is improved (less steady-state error) due to increased K_I value. Because "PID" is included in the name of id "cont1-PID", pre-built Azure Function with the PID control algorithm is automatically coupled with measured variable x during edge addition. K_P, K_I, and K_D are the properties of the vertex "cont1-PID" which can be updated using "setVProperty" in real time. The PID control algorithm is coded using Visual Studio 2017 and published to Azure as function nanoservice (Azure Function). This simulation makes 3 round-trips per iteration between the local computer (region: Mansfield Center, CT) and the cloud (region: Azure Data Center East US) and the average total round-trip time per iteration is 1.4810 seconds (or 0.4937 seond per round-trip) (**Figure 2C**). HTTPS is used for this preliminary work but if delivery latency is a concern, MQTT or AMQP are the best protocols to use. The PID control algorithm is widely used in engineering and it is estimated that over 90% of industrial loops are closed with PID controllers.

Preliminary work #2: Nanoservice-enabled automatic lighting control

Appropriate lighting controls ensure that the lighting system produces the right amount of light where and when it is needed. Benefits of a good control design include reduced energy costs and flexibility, which can support the user's visual needs and create a desired mood or ambience. **Figure 4** shows how nanoservices can be used

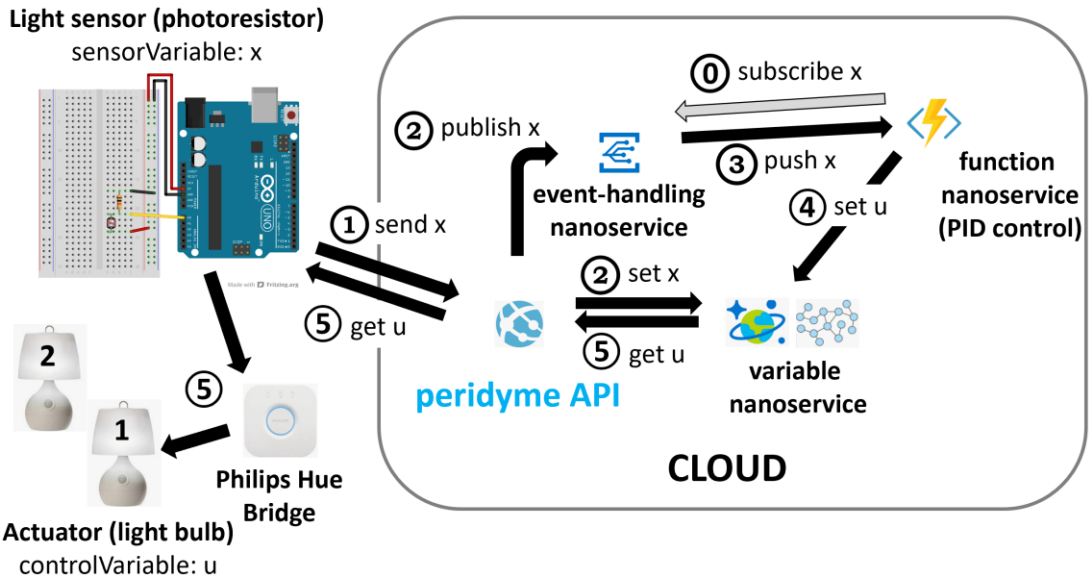

Figure 4. Nanoservice-enabled automatic lighting control.

to enable automatic lighting control using the PID control algorithm. The brightness of the room (sensorVariable x) is measured by the Arduino-enabled photoresistor and sent to **peridyme API** (① in the figure). **peridyme API** publishes x to event-handing nanoservice (Azure Event Grid) while setting the value of x using data nanoservice (Azure Cosmos DB Graph) (② in the figure). Event-handing nanoservice pushes x to PID control function nanoservice (③ in the figure) previously subscribed. The function nanoservice computes the value of the controlVariable u using the PID control algorithm and sets the value of u using data nanoservice (Azure Cosmos DB Graph) (④ in the figure). The value of u is then pulled by the Arduino and used to update the brightness of the light bulb (Philips Hue) through the Philips Hue Bridge (⑤ in the figure). Using the proposed nanoservice approach, different light bulbs (1 and 2 in the figure) can be coupled with the photoresistor by simply re-wiring relevant edges in the graph.

Using a chatbot to consume nanoservices

Chatting is a natural and convenient form of communication for humans and a chatbot is a computer program that can conduct a conversation with humans via sound or text methods. Recently, building, connecting, deploying, and managing intelligent online chatbots to interact naturally with humans on websites or apps have become popular, thanks to fully-managed bot services like Microsoft Azure Bot Service [31]. Chatbots can enable engineers and DIY hobbyists to create graph entities and to perform graph query operations without writing a single line of code. Since these graphs are integrated with pre-built function and event handling nanoservices, it is possible that they can build smart and connected IoT applications using voice or text only. An example conversation is shown in **Figure 5A**. Although not directly related to this project, a prototype chatbot project titled "A logic-based clinical decision support chatbot for patients" can be accessed at https://www.peridyme.com/prototype-chatbot (**Figure 5B**). It was made using Azure Bot Service, Azure Language Understanding (LUIS) [32], Azure Service Fabric [15], Azure Function, and Azure Event Grid. Azure Language Understanding is a machine learning-based service to build natural language understanding into chatbots, apps, and even IoT devices.

Nanoservice-enabled smart and connected personalized artificial pancreas

Smart and connected personalized artificial pancreas is a good example of distributed (connected) and adaptive (smart) IoT applications. Diabetes mellitus is a disease of chronically elevated blood glucose concentrations [33]. The serious complications and high prevalence make the disease a major public health concern. The primary treatment in all patients with type 1 diabetes is insulin since these patients suffer from insulin deficiency. However, those trying to achieve better blood glucose control through intensive insulin therapy often faced an increased

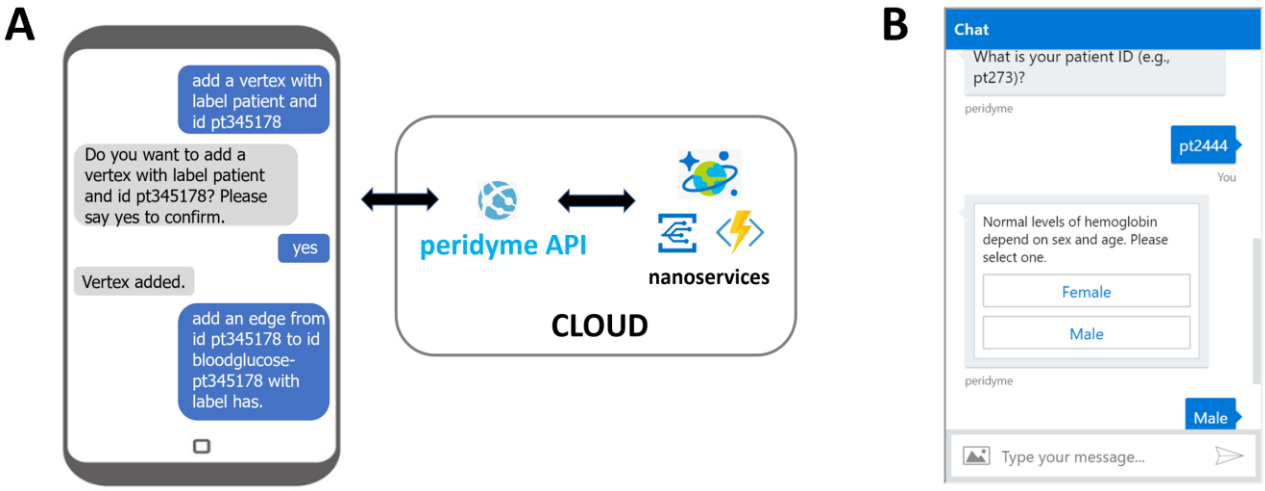

Figure 5. Chatbots. (A) An example of using a chatbot to consume nanoservices. (B) Previously developed prototype chatbot for "logic-based clinical decision support" (https://www.peridyme.com/prototype-chatbot). Azure Bot Service and Azure Language Understanding were used for the development.

risk of hypoglycemia (abnormally low blood glucose concentration) with severe consequences to heart and brain. In this context, keeping optimal blood glucose control while reducing the risk of hypoglycemia became a main thrust for the development of new insulin delivery strategies. In recent years, automation of insulin delivery, also known as the artificial pancreas, has been explored as a promising technology to achieve the goal.

The artificial pancreas consists of three parts: 1) continuous subcutaneous glucose sensor, 2) computer, and 3) continuous subcutaneous insulin pump [34]. The computer receives glucose levels at regular intervals (e.g., 5 minutes) from the sensor and computes the amount of insulin to be administered per unit time *via* the pump. The adequacy of the sensor accuracy and the consideration of the time delay between subcutaneous glucose readings and actual blood concentrations are critical. Like a healthy pancreas, an insulin pump can provide both basal and bolus insulin. Basal insulin is the background insulin continuously released in small amounts throughout the day. In contrast, bolus insulin refers to the extra-large amount of insulin released for a brief period in response to glucose-elevating events like food intake. Since there is a time delay between subcutaneous insulin injection and insulin action, bolus insulin is typically given from 10 to 20 minutes prior to a meal. In September 2016, the U.S. Food and Drug Administration (FDA) approved first artificial pancreas (the MiniMed 670G system from Medtronic) that automatically checks glucose and provides proper basal insulin in people 14 years of age and older with type 1 diabetes [35]. The basal rate is automated using the proportional-integral-derivative (PID) control algorithm with insulin-on-board (IOB) feedback (insulin-on-board is the calculation that tells patients how much insulin is still active from earlier bolus doses). This artificial pancreas system is not fully automated because it still depends on the user to enter meal information for bolus insulin. Although the system is a historic achievement, it has been criticized for the lack of customizability because automating insulin delivery should consider the wide variations between people and a myriad of other factors such as stress, sleep, and medication that may dynamically change over time. Although multiple clinical studies have been testing more advanced artificial pancreas systems [36], following limitations are not still addressed from a patient's perspective:

1. Lack of personalization: Due to limited resources, they often focus on a specific approach (e.g., control algorithm) and force end users to be locked in. However, patients may need to try different approaches before finding the one that meets individual needs.
2. Lack of adaptation: A single approach may not work all the time. Patients may want to use different approaches as needed to manage on-going changes.
3. Lack of collective learning: Diabetic patients are distributed all over the world and "close" patients with similar age, weight, height, living style, etc., may want to share with others what they learn from their artificial pancreas system.

As a result, proactive patients are currently building their own Do-It-Yourself artificial pancreas (the Open Artificial Pancreas Project [13]). This is often done without any supervising clinician or researcher, raising safety and security concerns.

Similarly, researchers are also facing challenges that need to be addressed. Deploying simple forms of approach such as the PID control algorithm in a single device may not be so demanding. The challenges arise from:

1. how the approach can readily adapt to changes in underlying medical knowledge or clinical finding.
2. how the approach can form an interconnected process with other modules in a flexible fashion.
3. how the approach can be widely available and scalable.
4. how the approach can be interoperable (i.e., language and platform independent) so that distributed artificial pancreas systems can share what they learn.

In this regard, creating a safe and secure platform that also enables customization and knowledge sharing is critically needed for DIY patients. Eventually, it will be a platform where engineers supply proven control approaches as integrated nanoservices while DIY patients can try different approaches on an individual basis, under the guidance of physicians and researchers. This project will investigate smart and connected personalized artificial pancreas to find out the right nanoservice approach during Phase I. In Phase II, a platform specifically designed for smart and connected personalized artificial pancreas will be developed based on the Phase I study.

Control objectives of the artificial pancreas

The control objectives may include maximization of time spent within a desired glucose range, minimization of hypoglycemic events, prevention of postprandial hyperglycemia, and minimization of patient intervention. These objectives need to be tailored to individual patients. For example, children are considered a high-risk group and their artificial pancreas should satisfy more stringent safety requirements. Given a set of control objectives, a suitable controller can be designed. Possible design approaches are shown in **Figure 6** [37]. For each component in the figure, there are multiple design options. The first component is desired (target) glucose level that can be either a specific value or a range. Desired glucose level can also be fixed or change over time (time-varying). The controller receives measured glucose level and computes a new insulin basal infusion rate after comparing it to desired concentration. Several control algorithms have been evaluated, including the proportional-integral-derivative (PID) control, model predictive control (MPC), and fuzzy logic control (FL). These algorithms may also calculate glucagon and/or other medications and additional modules can be added to the controller, including hypoglycemia prediction, hyperglycemia prediction, meal detection, exercise detection, stress detection, etc (**Figure 6**). Model predictive control is a model-based control approach that uses adaptive system identification discussed in more detail later. While the PID control is simpler and widely used in science and engineering, model predictive control is more suitable for advanced multivariable artificial pancreas that uses not only glucose sensor data but also other physiological information such as heart rate and skin temperature, but at the expense of increased computation [37]. These sensor data can be processed by filters and/or fault detection algorithms to increase the quality of the data before being sent to the controller. The patient may be involved in this control loop. For example, the patient may be required to make a meal announcement so that the controller can deliver appropriate bolus insulin as in the case of the first FDA-approved artificial pancreas. However, whenever a human is involved, safety concern is increased because of the unpredictable nature of human behavior.

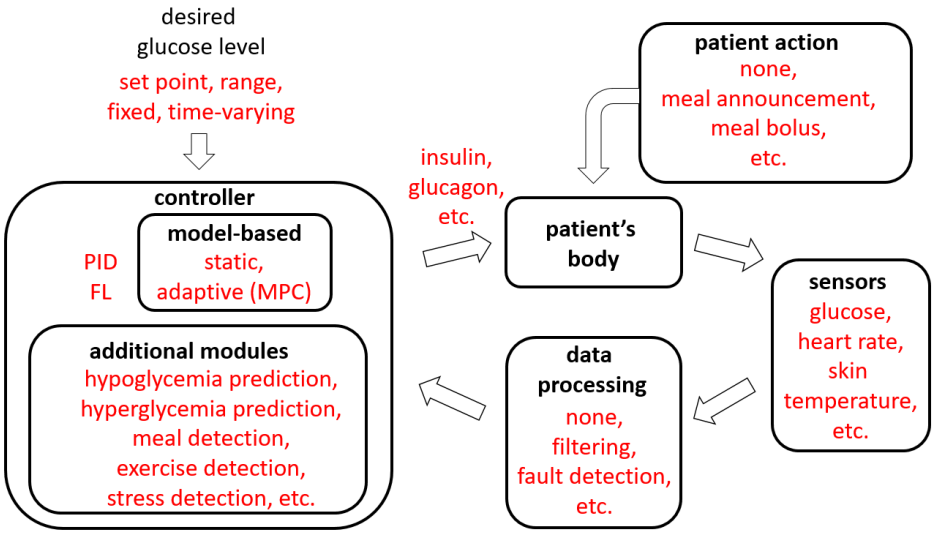

Figure 6. Possible design options for the artificial pancreas.

Nanoservices can be used to implement heterogeneous components described in this section and studying efficient implementation will be one of the focuses of Phase I.

Smart (adaptive) multivariable system identification for personalized artificial pancreas

High-throughput streaming data are increasingly available thanks to advances in technologies such as continuous glucose monitoring, enabling real-time system identification geared towards personalized artificial pancreas. In this section, an adaptive multivariable system identification approach will be introduced that can analyze the effect of *n* multiple physiologic variables (independent continuous random variables) x_1,…, x_n, such as heart rate and skin temperature, on glucose level y (a continuous dependent random variable) using the data $y, x_1, \ldots, x_n$. Considering the fact that any current state is the result of previous states, it can be assumed that the current value of y depends on M previous values of y, N_1 previous values of x_1, …, and N_n previous values of x_n:

$$y[i] = \beta_0[i] + \sum_{j=1}^{M} \alpha_j[i]y[i-j] + \sum_{j=1}^{N_1} \beta_{1,j}[i]x_1[i-j] + \cdots + \sum_{j=1}^{N_n} \beta_{n,j}[i]x_n[i-j] + v[i] \quad [1]$$

where i is the discrete-time index (integer), $\alpha_1[i], \ldots, \alpha_M[i]$ are the time-varying coefficients for M previous y terms and $\beta_{n,1}[i]$, …, $\beta_{n,N_n}[i]$ are the time-varying coefficients for N_n previous x_n terms. $v[i]$ is a continuous random variable representing the measurement noise. Measured data $y[i], \ldots, y[i-M], x_1[i-1], \ldots, x_1[i-N_1], \ldots, x_n[i-1]$, …, $x_n[i-N_n]$ and all β coefficients are not fixed and can change over time. Eq. 1 can be compactly put into matrix form as:

$$y[i] = \boldsymbol{u}[i]^T \boldsymbol{w}[i] + v[i] \quad [2]$$

where:

$$\boldsymbol{u}[i] = \left[1, y[i-1], \cdots, y[i-M], x_1[i-1], \cdots, x_1[i-N_1], \ldots, x_n[i-1], \cdots, x_n[i-N_n]\right]^T$$

$$\boldsymbol{w}[i] = \left[\beta_0[i], \alpha_1[i], \cdots, \alpha_M[i], \beta_{1,1}[i], \cdots, \beta_{1,N_1}[i], \cdots, \beta_{n,1}[i], \cdots, \beta_{n,N_n}[i]\right]^T \quad [3]$$

By denoting estimated $\boldsymbol{w}$ as $\widehat{\boldsymbol{w}}$, estimated $y[i]$ or $\hat{y}[i]$ can be expressed as:

$$\hat{y}[i] = \boldsymbol{u}[i]^T \widehat{\boldsymbol{w}}[i] \quad [4]$$

The estimation error or the difference between $y[i]$ and $\hat{y}[i]$ can be expressed as:

$$e[i] = y[i] - \hat{y}[i] = y[i] - \boldsymbol{u}[i]^T \widehat{\boldsymbol{w}}[i] \quad [5]$$

Given $y[i]$ and $\boldsymbol{u}[i]$ which are measured data, $\widehat{\boldsymbol{w}}[i+1]$ can be estimated by minimizing the expected value of $e[i]^2$ or $E\{e[i]^2\}$. Using the stochastic-gradient recursion, $E\{e[i]^2\}$ can be approximated and a recursive relation for updating $\widehat{\boldsymbol{w}}[i]$ can be shown as:

$$\widehat{\boldsymbol{w}}[i+1] = \widehat{\boldsymbol{w}}[i] + \mu \boldsymbol{u}[i]e[i] \quad [6]$$

where μ is a controllable step-size parameter. There is a trade-off between the estimation error and coefficient convergence rate and μ is often tuned to achieve desired performance**.** Eq. 6 is the Least Mean Squares (LMS) algorithm, one of adaptive system identification techniques used in science and engineering to identify dynamic physical systems (e.g., Unmanned Aerial Vehicle (UAV), Global Positioning System (GPS), etc.) in the presence of uncertainty [38]. Although simple in implementation, LMS is capable of delivering high performance through iterative adaptation or "learning" over time. I previously demonstrated that the dynamic behavior of the p53-Mdm2 gene network in individual cells can be tracked using adaptive system identification algorithms (normalized LMS, RLS (Recursive Least Squares), and the Kalman filter) and the resulting time-varying models were able to approximate experimental measurements more accurately than time-invariant models [39]. These adaptive methods can be considered as variants of recurrent neural networks studied in deep learning which exhibit dynamic temporal behavior. The outcome of Eq. 1 can also be used to predict future y value $\hat{y}[i+1]$:

$$\hat{y}[i+1] = \boldsymbol{u}[i+1]^T \widehat{\boldsymbol{w}}[i+1] \quad [7]$$

where $\boldsymbol{u}[i+1]$ includes only the data measured before $i+1$ and $\widehat{\boldsymbol{w}}[i+1]$ is computed using $\widehat{\boldsymbol{w}}[i]$ and previous data ($y[i]$ and $\boldsymbol{u}[i]$). This predictive capability can be used for disease treatment and/or prevention.

Described adaptive and personalized multivariable system identification approach shows following characteristics:

1. It enables a continuously "learning" system through adaptation to changes. The coefficient estimate vector $\widehat{\boldsymbol{w}}$ provides customized correlation values for a single patient. The estimation error can be large initially, but it decreases as the systems learns more ($\widehat{\boldsymbol{w}}$ converges over time).
2. y and $\boldsymbol{u}$ are measured over time and $\widehat{\boldsymbol{w}}$ adapts to their fluctuations. In some cases, $\widehat{\boldsymbol{w}}$ "tracks" them in real time and constantly attempts to converge to new values. Unlike conventional multiple linear regression which requires batch analysis, this adaptive approach uses only immediate previous y and $\boldsymbol{u}$ for estimation reducing memory usage.

3. Since $\widehat{\boldsymbol{w}}$ can change over time, this variability can capture the non-linear dynamics of the relationships between dependent ($y[i]$) and independent ($y[i-1], \dots, y[i-M], x_1[i-1], \dots, x_1[i-N_1], \dots, x_n[i-1]$, …, $x_n[i-N_n]$) variables which is assumed to be linear when the coefficients are fixed.
4. Unlike multiple linear regression that demands several statistical assumptions, adaptive estimation techniques can deal with statistical variations [38].
5. Since this is a personalized approach, the solution from population-based studies to control confounding, such as randomization, restriction, and matching, cannot be used. The only way of removing the effect of any confounder is finding out and adding it to the study as an independent variable through medical knowledge-based and/or brute-force search.
6. As mentioned earlier, time delay plays a critical role in physiological dynamics. For example, $x_1[i-1]$ and $x_1[i-60]$ need to be treated as two separate potential confounding variables although they both belong to x_1. In case $x_1$represents heart rate, the heart rate measured 1 minute ago ($x_1[i-1]$) can be correlated with current glucose level ($y[i]$) but the heart rate measured 60 minutes ago ($x_1[i-60]$) may not be so.

Smart (adaptive) and connected (distributed) multivariable system identification for personalized artificial pancreas

The continuous glucose monitoring sensor data can be unreliable due to sensor deterioration/failure, unexpected communication interruption, significant environmental disturbance, etc., which can negatively influence or even disable the system's learning process. A distributed approach that enables "collective learning" can be considered for improved performance and greater resilience to failure. Privacy and secrecy are other reasons that promote distributed implementation.

The analysis and design of a distributed multi-agent network for collective optimization, adaptation, and learning from real-time streaming data have been studied in diverse fields of science and engineering, including swarm robotics, social and economic sciences, and biological sciences. Nature provides numerous examples of distributed networks, such as fish schooling, bee swarming, and bird flight formation, exhibiting sophisticated and intelligent behavior that emerges from interactions among agents and with the environment. I previously demonstrated this collective intelligence using a population of effector helper T cell, which is a type of immune cells that senses biological attacks such as bacterial infection at an early stage [40]. My simulation and analysis suggested that these cells communicate with each other and make "collective decisions" more reliable than individual decisions prone to error and failure, while individual cells exhibit adaptive behavior.

There are three families of distributed strategies reported previously: (a) incremental strategies [41, 42], (b) consensus strategies [43, 44], and (c) diffusion strategies [45-48]. In this section, I will focus on diffusion strategies which outperforms the other two strategies in the context of adaptive networks [45]. Adaptive networks consist of distributed agents with adaptation and learning abilities. These agents are linked together through a specific topology and they interact with each other locally to solve inference and optimization problems in real time. Each agent shares information with its neighbors to improve its learning process. The continuous sharing and "diffusion" of information through overlapped local interactions enable the agents to respond to drifts in the data and to changes in the network topology in a continuous fashion. These networks are also robust to vertex (agent) and edge failures and easily scalable. Diffusion strategies have been used to implement adaptive sensor networks [45-49] in communication engineering and to model biological adaptation. Biological examples include bird flight formation [50], honeybee swarming behavior [51], and bacterial motility [52]. To the best of my knowledge, diffusion strategies have not yet been applied to the artificial pancreas.

A network with 9 vertices is illustrated in **Figure 7A**. Each vertex represents a single patient. Vertices (patients) connected by edges can share information with each other. Note these connected vertices can exist in diverse places, including hospital electronic health record systems and cloud computing environments. The multi-dimensional distance between two vertices is determined by the differences and similarities between two patients. For example, the distance between two patients with similar age, weight, height, past medical history, family history, social history, etc. will be very short. Since shorter distance is associated with less confounding, diffusion strategies should allow interactions between immediate or close neighbors only for better performance. Personal information can diffuse through overlapped localized interactions and may eventually affect the whole network in a dynamic fashion. For example, although red and blue sharing groups in **Figure 7A** are far apart without direct communication, vertices 4 and 6 mediate information flows so that these two distant groups can

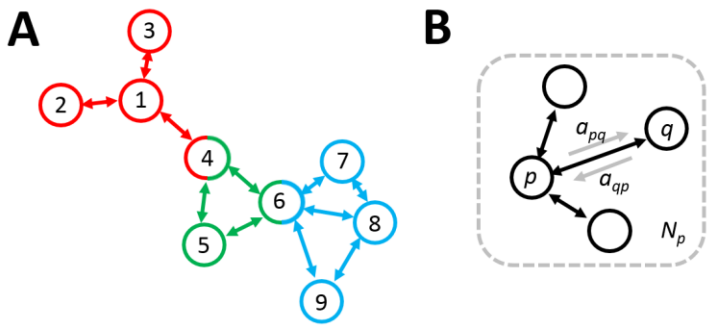

Figure 7. (A) 9-vertex network. (B) *p* vertex neighborhood (N_p).

share information. However, this diffusion mechanism does not necessarily lead to one global consensus due to constantly fluctuating local dynamics.

The distance between any two vertices are not fixed and can dynamically change over time as the patient data evolve. This may result in changes in the network topology. The network topology can also be adaptively modified through re-wiring to achieve better performance. For example, a neighbor vertex that constantly provides unacceptable information can be disconnected and replaced by a more reliable vertex. **Figure 7B** illustrates the neighborhood of vertex *p* (denoted by N_p) that consists of all vertices connected to *p* by edges. It is shown in the figure that a pair of nonnegative weights a_{pq} and a_{qp} are assigned to the edge connecting *p* and its close neighbor *q*. From this figure, a diffusion strategy using LMS can be expressed as [46]:

$$\begin{cases} \boldsymbol{\psi}_p[i+1] = \widehat{\boldsymbol{w}}_p[i] + \mu \boldsymbol{u}_p[i] e[i] \\ \widehat{\boldsymbol{w}}_p[i+1] = \sum_{q \in N_p} a_{qp} \boldsymbol{\psi}_q[i] \quad \left(\sum_{q \in N_p} a_{qp} = 1 \right) \end{cases} \quad [8]$$

The first equation of Eq. 8 involves the adaptation process of vertex *p* and assigns the result to a temporary variable $\boldsymbol{\psi}_p$. The second equation combines these temporary values from the neighbors and the vertex to produce the final $\widehat{\boldsymbol{w}}_p$. The weight a_{qp} can be determined by the distance between the vertices while the sum of all weights should be equal to one. Note the temporary variable $\boldsymbol{\psi}_p$ is used to link these two equations. This coefficient estimation (also called adaptive system identification [53] which is a methodology commonly used in engineering for constructing time-varying mathematical models of dynamic systems from input and output data measured in real time) can be used to control or drive y values in a desired way. Feedback control algorithms such as MPC (Model Predictive Control) can be integrated with proposed smart and connected estimation to achieve robust and/or optimal control [54-56].

Preliminary work #3: Microservice-enabled smart and connected multivariable system identification

This preliminary work uses an Azure Service Fabric-based actor model [15] and I plan to transform this into a nanoservice-based prototype during Phase I. Collaboration with neighbors may save a patient when the learning capability is unexpectedly lost. For example, assume that there are four patient vertices (each vertex representing a type-1 diabetic patient) and patient *p* suddenly loses its learning capability (red arrow in **Figure 8**) while maintaining communication with its neighbors. The model for each patient can be expressed as:

$$y[i] = \beta_0[i] + \alpha_1[i] y[i-1] + \alpha_2[i] y[i-2] + v[i] \quad [9]$$

where y is the glucose level measured every five minutes. Eq. 9 is basically stating that the current glucose level $y[i]$ depends on the level 5 minutes ago ($y[i-1]$) and the level 10 minutes ago ($y[i-2]$). **Figure 8** shows that the patient *p* can still "learn" and make relatively sound predictions ($\hat{y}[i+1]$) using the coefficients estimates from its neighbors (yellow line in the figure) when in cooperative mode. Note the prediction errors are large in the beginning but decrease as the learning proceeds (dark blue line in the figure). In contrast, non-cooperative mode results in poor prediction (light blue line in the figure). This preliminary work is based on the online datasets from a study titled "A Randomized Clinical Trial to Assess the Efficacy of Real-Time Continuous Glucose Monitoring in the Management of Type 1 Diabetes" [57].

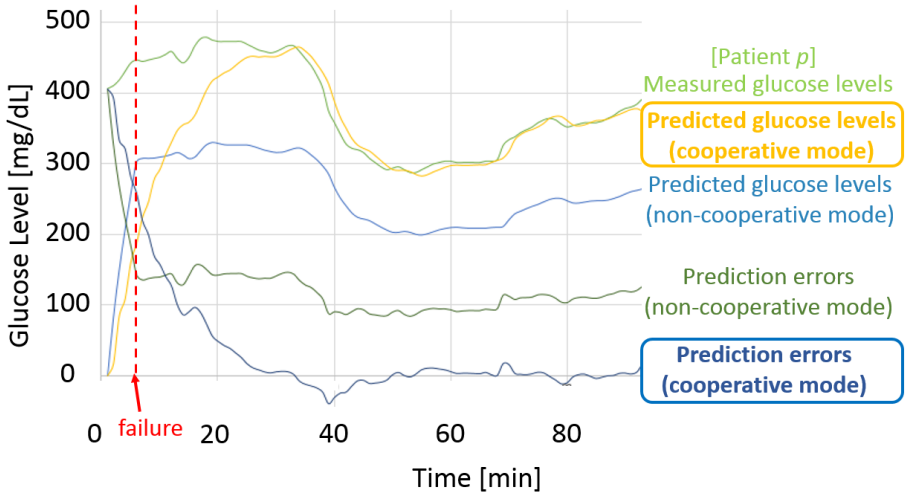

Figure 8. Cooperative vs. non-cooperative mode.

Edge computing for artificial pancreas

Edge computing is an approach of delivering cloud capabilities to the local (edge) devices to reduce the communication burden and to cope with the loss of network connectivity. Several vendors are currently providing edge computing service, including Amazon Web Service Greengrass [17] and Microsoft Azure IoT Edge [18]. Edge computing enables cloud code and services to be securely delivered to local artificial pancreas systems, thus distributing intelligence from the cloud to the edge. Although edge computing is cloud-enabled, edge devices can run even when they are offline or have intermittent connectivity to the cloud. Local artificial pancreas systems supported by Azure IoT Edge can communicate with Azure IoT Hub, a fully managed service that enables reliable and secure bidirectional communications between millions of edge devices and the cloud (**Figure 8**).

Recently, the lengthy and costly preclinical animal studies for the artificial pancreas have been replaced by *in silico* (computer-based) simulations, using the University of Virginia-Padova type 1 diabetes mellitus simulator (T1DMS). T1DMS is a computer model of the human metabolic system based on the glucose-insulin dynamics in human subjects. The T1DMS technology provides realistic computer simulation of clinical trials using a population of 300 subjects with parameters derived from triple tracer metabolic studies that reflect the human metabolism found with type 1 diabetes. It has been validated against actual clinical data and is accepted by the FDA as a substitute for pre-clinical animal trials in the testing of certain control strategies, including feedback control algorithms, for type 1 diabetes.

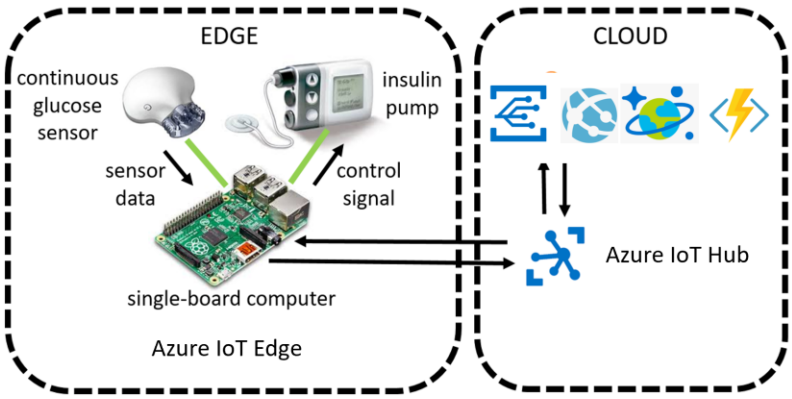

Figure 8. Edge computing for artificial pancreas.

Preliminary work #4: Nanoservice-enabled edge computing for artificial pancreas *in silico* pre-clinical trials

Function nanoservice (Azure Function) holding an optimized system identification or control algorithm can be put into a container such as Docker [58]. Container-enabled edge computing makes it possible for local edge devices to deploy multiple containers holding such algorithms from cloud repositories. These containers can interact with each other within the device while individual container is executing a specific function such as a control algorithm. A proof-of-concept edge computing for artificial pancreas is presented here. A control algorithm (PID) wrapped within a container is pulled from the cloud to a local device and used to interact with the UVA/PADOVA Type 1 Diabetes Simulator (**Figure 9**). Blood glucose levels are significantly lowered with control after meal intake (disturbance).

Project goals and questions

The primary goal of this project is to build a proof of concept nanoservices platform that engineers and DIY hobbyists can use to easily design, build, and test smart and connected IoT applications. Commonly-used

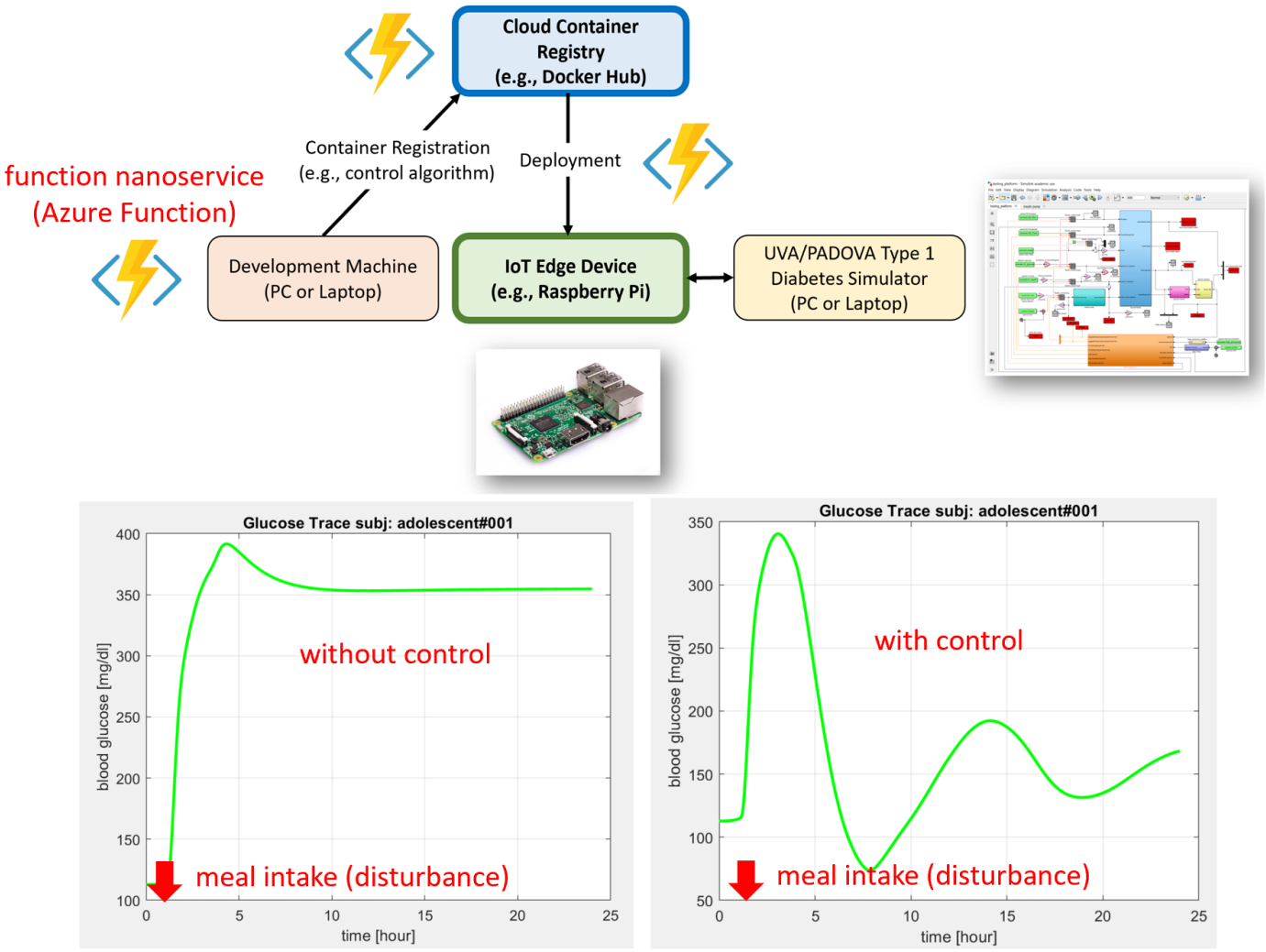

Figure 9. **Nanoservice-enabled edge computing for artificial pancreas** (*In silico* preclinical trial).

engineering algorithms such as LMS and MPC and their smart and connected applications will be implemented as prototypes to investigate any challenges and issues that they might face. More specifically, following goals will be achieved during Phase I:

1. Develop a prototype of nanoservice-enabled smart and connected multivariable system identification system.
2. Develop a prototype of nanoservice-enabled model predictive control (a nanoservice-enabled PID control prototype is demonstrated in this proposal).
3. Develop a prototype of nanoservice-enabled smart and connected adaptive model predictive control (smart and connected multivariable system identification + model predictive control).
4. Develop a prototype chatbot to consume nanoservices.
5. Develop a user-friendly process through which any engineer or DIY hobbyist can make her or his customized algorithm available as a function nanoservice that can also be deployed to edge devices.
6. Develop a prototype of integrating proposed nanoservices with other web services (e.g., email or text message notification service). For instance, if controller performance drops below a certain threshold, it can be notified to the engineer by email or text messaging.

While pursuing these goals, following research questions will also be explored:

- Distance/weighting: How can we define the distance between two vertices (e.g., patient vertices)? What are the factors which can affect weighting?
- Network topology: How can immediate neighbors be determined? Can we develop criteria for determining which and how many neighbors to interact? Will more neighbors always bring better performance? How can adaptive topology be implemented?

- Idle agents and link failures: How can idle agents and link failures be managed?
- Asynchronous communication: Asynchronous communication is preferred within an uncertain environment like the internet. However, asynchronous communication can cause irregular communication delay. Will this irregular delay degrade performance? What will be the tolerable range of delay?
- Sampling: The sampling interval for measuring each variable can be different and irregular (the time elapsed between two consecutive measurements may vary). How can this issue be addressed?
- Performance: What are the factors that can affect performance across the network?

Related project (prior support from NSF)
In August 2017, I received a 2-year grant titled “Distributed and Adaptive Personalized Medicine” from the NSF Smart and Connected Health program.

References

[1] *MATLAB,* https://www.mathworks.com/.
[2] *LabVIEW,* http://www.ni.com/en-us/shop/labview.html.
[3] *Arduino IDE,* https://www.arduino.cc/en/Main/Software.
[4] B. T. Murari, K. Wnuk and J. Börstler. Impact of internet of things on software business model and software industry (thesis), Blekinge Institute of Technology, 2016.
[5] *AWS IoT,* https://aws.amazon.com/iot/.
[6] *Azure IoT Hub,* https://azure.microsoft.com/en-us/services/iot-hub/.
[7] *Google IoT Core,* https://cloud.google.com/iot-core/.
[8] *Cloud Education Lags at Universities: 4 Professors' Perspectives,* https://clutch.co/cloud/resources/cloud-computing-education-2017.
[9] *Alexa Voice Service,* https://developer.amazon.com/public/solutions/alexa/alexa-voice-service.
[10] *What are microservices?*, https://microservices.io/.
[11] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin and L. Safina. Microservices: Yesterday, today, and tomorrow. Present and Ulterior Software Engineering. Springer, Cham, 2016.
[12] *GE Digital: What is Edge Computing?,* https://www.ge.com/digital/blog/what-edge-computing.
[13] *The Open Artificial Pancreas System Project,* https://openaps.org/.
[14] *Kubernetes*, https://kubernetes.io/.
[15] *Azure Service Fabric*, https://azure.microsoft.com/en-us/services/service-fabric/.
[16] *5G IS WEEKS AWAY — AND TUESDAY MARKS ITS FIRST REAL TEST,* https://www.theverge.com/2018/11/30/18119818/verizon-5g-network-demo-att-qualcomm-summit-snapdragon-hawaii.
[17] *AWS IoT Greengrass,* https://aws.amazon.com/greengrass/.
[18] *Microsoft Azure IoT Edge,* https://azure.microsoft.com/en-us/services/iot-edge/.
[19] *AWS: What is serverless?,* https://aws.amazon.com/serverless/.
[20] *neo4j: What is Graph Database?,* http://www.refworks.com/refworks2/default.aspx?r=references|MainLayout::init.
[21] *Introduction to Azure Cosmos DB: Gremlin API,* https://docs.microsoft.com/en-us/azure/cosmos-db/graph-introduction.
[22] *AWS Neptune,* https://aws.amazon.com/neptune/.
[23] *Azure Functions,* https://azure.microsoft.com/en-us/services/functions/.
[24] *AWS Lambda,* https://aws.amazon.com/lambda/.
[25] *Google Cloud Function,* https://cloud.google.com/functions/.
[26] *IBM Cloud Functions,* https://console.bluemix.net/openwhisk/.
[27] *Azure Event Grid*, https://azure.microsoft.com/en-us/services/event-grid/https://azure.microsoft.com/en-us/services/event-grid/.
[28] *AWS Simple Notification Service,* https://aws.amazon.com/sns/.
[29] *Google Cloud Pub/Sub,* https://cloud.google.com/pubsub/.
[30] *Apache TinkerPop,* http://tinkerpop.apache.org/.
[31] *Azure Bot Service,* https://azure.microsoft.com/en-us/services/bot-service/.
[32] *Azure Language Understanding,* https://azure.microsoft.com/en-us/services/cognitive-services/language-understanding-intelligent-service/.

[33] D. G. Gardner, D. M. Shoback, M. Anderson, D. C. Aron, M. L. Badell, D. D. Bikle, G. D. Braunstein, T. B. Carroll, M. I. Cedars, O. H. Clark, F. A. Conte, D. S. Cooper, J. W. Findling, P. A. FitzGerald, M. S. German, S. E. Gitelman, S. L. Greenspan, M. M. Grumbach, C. Grunfeld, J. C. Jaume, B. R. Javorsky, A. M. Kanaya, P. W. Ladenson, G. Lal, G. Lee, M. J. Malloy, U. Masharani, R. A. Nissenson, N. M. Resnick, A. G. Robinson, D. E. Sellmeyer, D. Styne, R. N. Taylor, J. B. Tyrrell and W. F. Young. *Greenspan's Basic and Clinical Endocrinology* (9th edition), 2011.
[34] S. Del Favero, D. Bruttomesso and C. Cobelli. Artificial pancreas: A review of fundamentals and inpatient and outpatient studies. *Technological Advances in the Treatment of Type 1 Diabetes.* pp. 166-189. 2015.
[35] *FDA approves first automated insulin delivery device for type 1 diabetes,* https://www.fda.gov/newsevents/newsroom/pressannouncements/ucm522974.htm.
[36] *NIH Announces Major Artificial Pancreas Clinical Trials,* https://www.endocrinologyadvisor.com/diabetes-technology/nih-artificial-pancreas-clinical-trials-to-begin/article/637655/.
[37] K. Turksoy, E. S. Bayrak, L. Quinn, E. Littlejohn and A. Cinar. Multivariable adaptive closed-loop control of an artificial pancreas without meal and activity announcement. *Diabetes Technology & Therapeutics 15(5),* pp. 386. 2013.
[38] A. H. Sayed, *Adaptive Filters.* Hoboken, New Jersey: Wiley-IEEE Press, 2008.
[39] Shin YJ, Sayed AH and Shen X, "Adaptive models for gene networks." *PLoS One,* vol. 7, pp. e31657, 2012.
[40] Y. Shin and B. Mahrou, "Modeling collective & intelligent decision making of multi-cellular populations," *36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Chicago, IL, 2014, pp. 334-337.
[41] C. G. Lopes and A. H. Sayed, "Incremental Adaptive Strategies Over Distributed Networks," *IEEE Transactions on Signal Processing,* vol. 55, pp. 4064-4077, 2007.
[42] D. P. Bertsekas. A new class of incremental gradient methods for least squares problems. *SIAM Journal on Optimization, 7(4),* pp. 913-926. 1997.
[43] S. Kar and J. M. F. Moura. Sensor networks with random links: Topology design for distributed consensus. *IEEE Transactions on Signal Processing, 56(7),* pp. 3315-3326. 2008.
[44] R. Olfati-Saber and R. M. Murray. Consensus problems in networks of agents with switching topology and time- delays. *IEEE Transactions on Automatic Control, 49(9),* pp. 1520-1533. 2004.
[45] A. H. Jianshu Chen and A. H. Sayed. Diffusion adaptation strategies for distributed optimization and learning over networks. *IEEE Transactions on Signal Processing, 60(8),* pp. 4289-4305. 2012.
[46] Cattivelli F.S. and Sayed A.H., "Diffusion LMS strategies for distributed estimation," *IEEE Transactions on Signal Processing,* vol. 58, pp. 1035-1048, 2010.
[47] F. S. Cattivelli, C. G. Lopes and A. H. Sayed, "Diffusion Recursive Least-Squares for Distributed Estimation Over Adaptive Networks," *IEEE Transactions on Signal Processing,* vol. 56, pp. 1865-1877, 2008.
[48] Lopes C.G. and Sayed A.H., "Diffusion least-mean squares over adaptive networks: Formulation and performance analysis," *IEEE Transactions on Signal Processing,* vol. 56, pp. 3122-3136, 2008.
[49] F. S. Cattivelli and A. H. Sayed, "Diffusion Strategies for Distributed Kalman Filtering and Smoothing," *IEEE Transactions on Automatic Control,* vol. 55, pp. 2069-2084, 2010.
[50] Cattivelli F.S. and Sayed A.H., "Modeling bird flight formations using diffusion adaptation," *IEEE Transactions on Signal Processing,* vol. 59, pp. 2038-2051, 2011.
[51] Jinchao Li, Sheng-Yuan Tu and A. H. Sayed, "Honeybee swarming behavior using diffusion adaptation," in *Digital Signal Processing Workshop and IEEE Signal Processing Education Workshop (DSP/SPE)*, pp. 249-254, 2011
[52] Chen J., Zhao X., Sayed A.H., "Bacterial motility via diffusion adaptation," *Asilomar Conference on Signals, Systems and Computers,* pp. 1930-1934, 2010.
[53] L. Ljung, *System Identification: Theory for the User.* Upper Saddle River, New Jersey: Prentice Hall, 1999.
[54] S. Skogestad and I. Postlethwaite, *Multivariable feedback control; analysis and design*, Wiley-Interscience, 2006.
[55] R. F. Stengel, *Optimal Control and Estimation.* Mineola, NY: Dover, 1994.
[56] G. F. Franklin and J. D. Powell, *Digital Control of Dynamic Systems.* Reading, Mass.: Addison-Wesley Pub. Co., 1980.
[57] Mauras N, Beck R, Xing D, et al., "A randomized clinical trial to assess the efficacy and safety of real-time continuous glucose monitoring in the management of type 1 diabetes in young children aged 4 to <10 years," *Diabetes Care*, 35(2):204-10, 2012.
[58] *Docker,* https://www.docker.com/.