GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

IMPLEMENTATION OF MICROGRID DISTRIBUTED CONTROL ARCHITECTURES ON AN INDUSTRIAL-GRADE CONTROL HARDWARE PLATFORM

Autor: Ginés Martínez Rivera

Director: Alejandro Domínguez García

University of Illinois at Urbana-Champaign

Champaign, IL.

Mayo 2022

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

IMPLEMENTATION OF MICROGRID DISTRIBUTED CONTROL ARCHITECTURES ON AN INDUSTRIAL-GRADE CONTROL HARDWARE PLATFORM

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2021/22 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Ginés Martínez Rivera Fecha: 20/05/2022

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Alejandro Domínguez García Fecha: 21/05/2022

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

IMPLEMENTATION OF MICROGRID DISTRIBUTED CONTROL ARCHITECTURES ON AN INDUSTRIAL-GRADE CONTROL HARDWARE PLATFORM

Autor: Ginés Martínez Rivera

Director: Alejandro Domínguez García

University of Illinois at Urbana-Champaign

Champaign, IL.

Mayo 2022

Abstract

Large power systems, e.g., bulk power transmission networks, base their generation control functions on a three-layered hierarchical control structure comprising a primary, secondary, and tertiary control layer. Conventionally, the secondary and tertiary control layers are based on a centralized control architecture in which a centrally located computer performs all the relevant computational tasks that are required to execute the pertinent control functions [3]. Its main drawback relies on the fact that this central computer needs to receive a massive amount of information from all the generating buses to execute each particular iteration of the control algorithm. This is where distributed control architecture comes into play.

The distributed architecture for Islanded AC Microgrids, which we will be working on, replicates the functionality of the secondary and tertiary control layers, e.g., secondary frequency regulation and optimal dispatch. In this project, we will synthesize numerous distributed algorithms on industrial-grade control hardware devices. These algorithms will facilitate the implementation of secondary frequency control, secondary voltage control, and economic dispatch via a distributed control architecture.

The hardware that will be utilized to emulate the microgrid's generators is the HIL's 4th Generation Typhoon HIL404, and C-language is the programming language that will be used to synthesize the distributed algorithms onto this hardware. Each HIL device will communicate with an Arduino Due microcontroller via CAN Bus communication protocol, which in turn will be able to wirelessly exchange information with each other through embedded MaxStream XB24-DMCIT-250 rev B XBee modules. Finally, we will test and validate it in a hardware-in-the-loop testbed under a variety of scenarios.

Subject Keywords: microgrid; distributed generation control architecture; distributed control algorithms; optimal dispatch; Distributed Generation Resources (DGR); CAN Bus; CAN Bus – XBee.

Acknowledgments

Special mention should be given to Research Engineer Olaolu Ajala, whose patience and support throughout the development of this thesis deserve my sincere appreciation. I would also like to express my gratitude to Professor Alejandro Dominguez-Garcia for having chosen me to work on this project. I deeply appreciate his interest, guidance, and willingness to help over the full year. It has been an incredible experience to perform this research project with them, and I am honestly thankful for this opportunity.

My sincere thanks to Universidad Pontificia de Comillas ICAI for having made it possible for me to be here this year as an exchange student, as well as to University of Illinois at Urbana-Champaign and the Electrical and Computer Engineering Department for their teaching and the fantastic treatment they have given to us all.

Finally, I must express my deepest gratitude to my family and closest friends, who have been with me in the good times, but have also supported me and my bad moods in the tough ones. They deserve 90% of the credit for the work collected on the following pages.

IMPLEMENTACIÓN DE ARQUITECTURAS DE CONTROL DISTRIBUIDO DE MICROGRIDS EN UNA PLATAFORMA DE HARDWARE DE CONTROL DE GRADO INDUSTRIAL

Autor: Martínez Rivera, Ginés.

Director: Domínguez García, Alejandro.

Entidad Colaboradora: University of Illinois at Urbana-Champaign

RESUMEN DEL PROYECTO

Se trabajará en una arquitectura de control de generación distribuido para microgrids, que replica la funcionalidad del segundo y tercer nivel del control frecuencia-potencia que impera en los grandes sistemas eléctricos -regulaciones secundaria y terciaria-. Se desarrollarán algoritmos de control distribuidos (programados en lenguaje C), que se implementarán en varias unidades de hardware de control de grado industrial (HIL's 4th Generation Typhoon HIL404). Cada dispositivo HIL se comunicará con un microcontrolador Arduino Due a través del protocolo de comunicación CAN Bus. A su vez, estos podrán intercambiar información entre sí de forma inalámbrica por medio de módulos XBee MaxStream XB24-DMCIT-250 rev B integrados. Finalmente, los algoritmos y los mencionados protocolos de comunicación serán probados y validados mediante pruebas de hardware-in-the-loop ante múltiples escenarios.

Palabras clave: microgrid; arquitectura de control de generación distribuida; algoritmos de control distribuido; Recursos de Generación Distribuida (RGD); CAN Bus; CAN Bus – XBee.

1. Introducción

El naciente campo de los microgrids atrae cada vez más atención y estudios que buscan explotar su potencial. Se prevé que los actuales grandes sistemas eléctricos centralizados evolucionen hacia redes más descentralizadas [3]. El trabajo que aquí se presenta pretende aprovechar las características inherentes de los microgrids para optimizar su arquitectura de control frecuencia-potencia.

Un microgrid puede definirse genéricamente como una red de cargas y generadores interconectados, que es físicamente más pequeño, maneja potencias y capacidades inferiores a las de los grandes sistemas de energía eléctrica, y puede funcionar aislado de la red. A pesar de estas diferencias, el funcionamiento normal tanto de los microgrids como de los grandes sistemas eléctricos puede reducirse a tres requisitos:

(i) la generación debe satisfacer la demanda total de la red, (ii) la frecuencia debe mantenerse próxima a su valor nominal (50 Hz en Europa, 60 Hz en Estados Unidos), y (iii) los costes de generación deben minimizarse [1]. Variaciones de carga o fallos en generadores y líneas de transmisión pueden provocar un desajuste en el sistema.

Para hacer frente a estas situaciones, cualquier tipo de sistema eléctrico de corriente alterna implementa una arquitectura de control de generación organizada en tres niveles: primario, secundario y terciario. La regulación primaria actúa localmente de forma instantánea y se encarga de equilibrar la potencia generada y la demandada, al tiempo que garantiza que la frecuencia no se desvíe en exceso de su valor nominal. La regulación secundaria o Automatic Generation Control (AGC), restablece la frecuencia nominal y mantiene un intercambio de energía adecuado entre las diferentes áreas del sistema. Por último, la regulación terciaria se encarga de ajustar los niveles de oferta de los distintos generadores de la red de forma que se minimice el coste total de generación [2], [4].

Convencionalmente, para los grandes sistemas de energía, únicamente la regulación primaria depende de información local. La regulación secundaria y terciaria, en cambio, se fundamentan en una arquitectura centralizada [4]. En otras palabras, un ordenador central se encarga de recopilar información de todos los nodos de generación y de coordinar las funciones de control. Sin embargo, debe notarse que este enfoque centralizado implica una gran complejidad computacional, ya que el ordenador central debe recibir y gestionar una enorme cantidad de información de cada nudo generador en cada iteración.

2. Definición del Proyecto

El enfoque centralizado del control frecuencia-potencia está ya sobradamente establecido en la actual operación de los grandes sistemas de energía eléctrica, pero este no es el caso en el emergente campo de los microgrids [1]. Además, sus características estructurales hacen que estos sean especialmente adecuados para la implementación de una arquitectura de control distribuido. De este modo, los controladores ubicados en cada Recurso de Generación Distribuido (RGD)[1] podrían realizar colectivamente la misma función que un ordenador central mediante el simple intercambio de información local con sus controladores vecinos.

[1] El término *Recursos de Generación Distribuidos (RGD)* hará referencia tanto a generadores síncronos como a aquellos conectados a la red a través de inversores (generación renovable principalmente) [1].

A lo largo del proyecto se demostrará cómo la arquitectura de control distribuido para microgrids de CA en isla puede replicar la funcionalidad de la regulaciones secundaria y terciaria de los grandes sistemas eléctricos. Esta estructura descentralizada puede aportar indudables ventajas sobre el enfoque centralizado tradicional, ya que no requiere conocimiento del tipo y características de cada generador, ni la existencia de una compleja red de comunicación entre cada RGD y un procesador central responsable del manejo de toda esta información. Una arquitectura de control distribuido también aumentaría la adaptabilidad del sistema para añadir o eliminar unidades de generación sin afectar a su normal funcionamiento [1].

3. Descripción General del Sistema

La arquitectura de control de generación distribuido se implementará en un microgrid de laboratorio compuesto por cuatro generadores síncronos interconectados con varias cargas resistivas. Cada RGD (HIL's 4th Generation Typhoon HIL404) está equipado con un microcontrolador Arduino Due, capaz de intercambiar información (bi o unidireccionalmente) con sus controladores vecinos a través de módulos MaxStream XB24-DMCIT-250 rev B integrados. La Figura 1 muestra un esquema representativo del sistema.

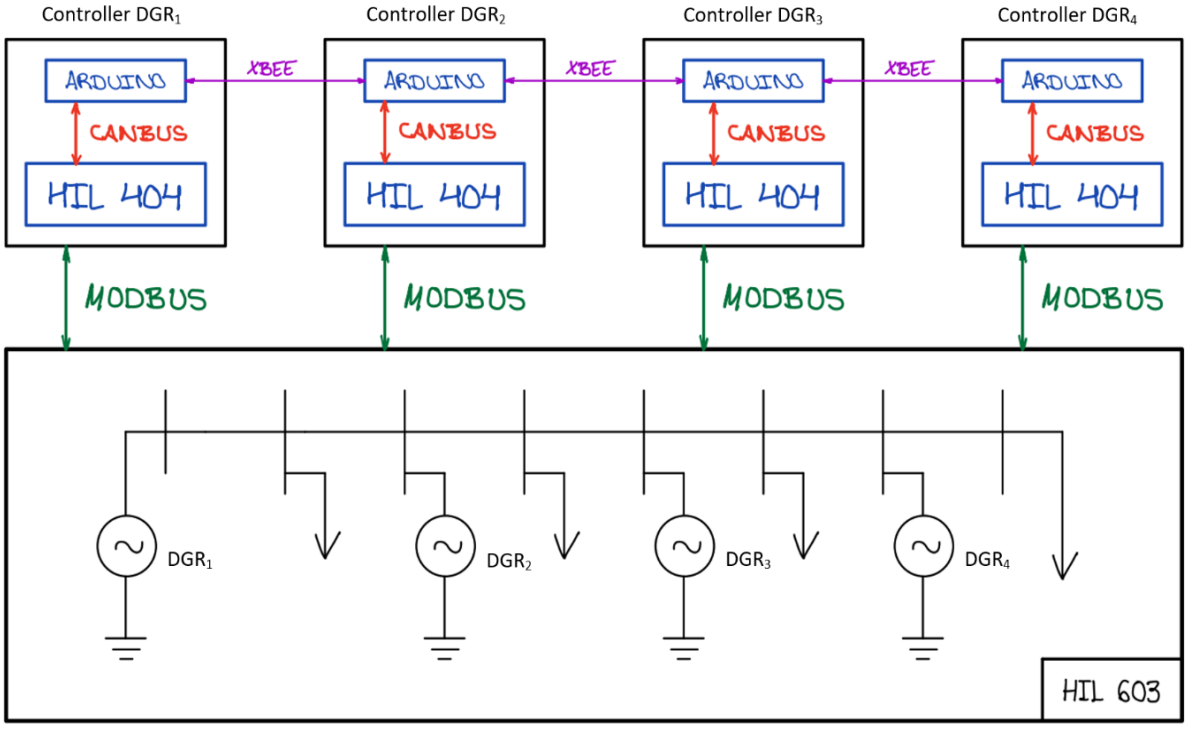

Figura 1. Esquema general del proyecto

La primera y principal tarea requerirá la síntesis de diferentes algoritmos distribuidos en cuatro dispositivos de hardware de control de grado industrial (HIL's 4th Generation Typhoon HIL404). Estos algoritmos permitirán a los generadores llevar a cabo tanto la regulación de frecuencia como, principalmente, la regulación terciaria.

Una vez implementado el código en estos dispositivos, el siguiente paso será permitir que cada uno de los HIL404 (en la práctica, los RGD) interactúe con sus respectivos microcontroladores Arduino a través del protocolo de comunicación CAN Bus. A su vez, estos controladores locales podrán enviar y recibir información de forma inalámbrica hacia y desde su vecindario a través de módulos XBee integrados.

Por último, aunque no será abordado en este proyecto, este sistema de cuatro nodos se sintetizará, a través del protocolo de comunicación ModBus, en una simulación en tiempo real del microgrid a nivel de laboratorio (dispositivo HIL 603) para probarlo y validarlo frente a diferentes escenarios.

4. Resultados

La primera etapa de la implementación del control de generación distribuido, representado en la Figura 1, consiste en el desarrollo y programación de los algoritmos distribuidos. En primer lugar, se aborda el denominado Ratio-Consensus, un algoritmo lineal iterativo que servirá de punto de partida para ejecutar la regulaciones secundaria y terciaria de manera distribuida. Sin embargo, Ratio-Consensus resulta insuficiente a la hora de su aplicación práctica, ya que no tiene en cuenta la pérdida de paquetes de información al viajar entre nodos. Por lo tanto, se hace necesario desarrollar una extensión de este algoritmo: el así llamado Robust Ratio-Consensus.

Finalmente, utilizamos estos algoritmos preliminares para programar y modelar el algoritmo Robust Distributed Primal-Dual, que se probará capaz de resolver de forma distribuida el problema del reparto óptimo de generación en un microgrid de CA en funcionamiento en isla. Dejemos que $f_i(p_i) = a_i \cdot p_i^2$, $i = 1, 2, 3, 4$, denote las funciones de coste de los cuatro RGD del microgrid representado en la Figura 1 [5]. A continuación, en Figura 2 se incluye la convergencia del algoritmo ante diferentes valores de los parámetros a_i.

La gráfica superior de la Figura 2 muestra el caso en el que $a_i = 0.1,\ i = 1, 2, 3, 4$, es decir, los cuatro generadores tienen idénticos costes de generación. El resultado es que las potencias de salida convergen sinusoidalmente a 0,4 pu, coincidiendo la demanda total previamente fijada en 1,6 pu. En la parte central de la Figura 2, las funciones de coste de los DGR se han modificado de forma que $a_1 = a_2 = 0.1,\ a_3 =$

$a_4 = 0.4$. Aunque las curvas se solapan, es fácil ver que la potencia suministrada por los generadores 1 y 2 (*gamma9* y *gamma10*) es mayor que la suministrada por 3 y 4 (*gamma11* y *gamma12*). Por último, la gráfica inferior de la Figura 2 considera el escenario en que todos los DGR tienen diferente rentabilidad: $a_1 = 0.1,\ a_2 = 0.2,\ a_3 = 0.3,\ a_4 = 0.4$. Al igual que antes, la potencia generada por cada uno está inversamente relacionada con el coste de su generación. Se puede comprobar fácilmente que, en estado estacionario, la suma de las cuatro potencias satisface la demanda total de 1,6 pu.

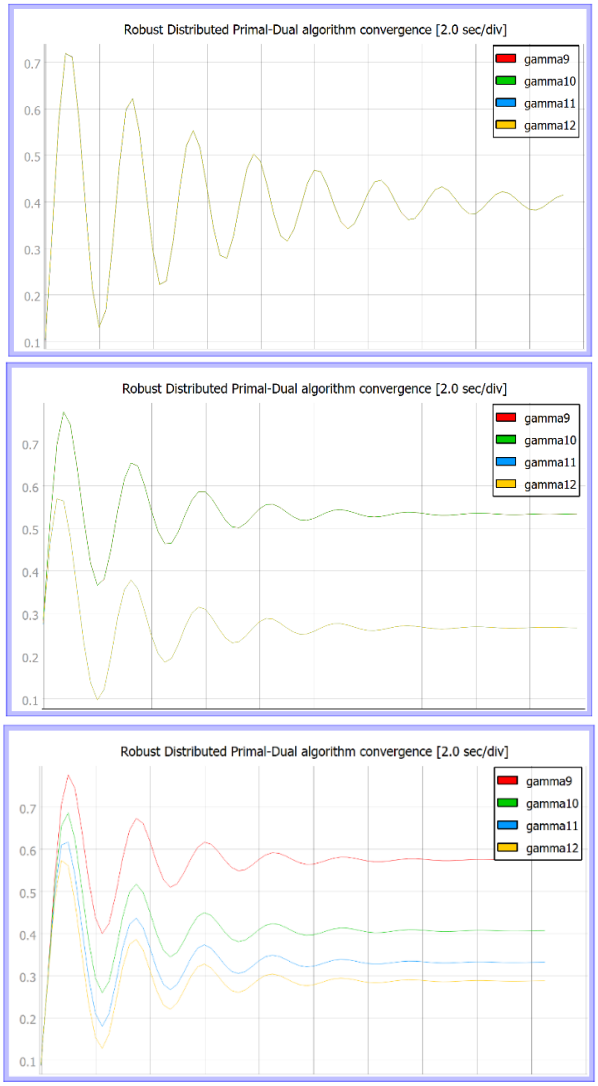

Figura 2. Convergencia del algoritmo Robust Distributed Primal-Dual

Con estas simulaciones se pretende ilustrar que el algoritmo Robust Distributed Primal-Dual puede ser utilizado para implementar con éxito la regulación terciaria de forma distribuida. Para ello, primero se deben desarrollar y testar los protocolos de comunicación que permitirán el intercambio de información entre los nodos de la red.

En primer lugar, una vez sintetizados los algoritmos de control distribuido en los dispositivos Typhoon HIL404, se aborda el intercambio bidireccional de información entre éstos -los DGR- y los controladores locales -placas Arduino con módulos XBee integrados- a través del protocolo Controller Area Network (CAN o CAN Bus). A continuación, se incluyen los resultados de la prueba de comunicación CAN Bus, si bien el presente resumen omitirá una descripción más detallada de este protocolo.

En lugar de ejecutar el algoritmo completo de regulación terciaria (Robust Distributed Primal-Dual), se enviarán cuatro valores constantes (5, 10, 15 y 1) desde el generador HIL a su correspondiente controlador local Arduino-XBee. A su vez, el Arduino leerá y procesará esta información, y también transmitirá otras cuatro variables desde el microcontrolador al HIL404. Estas se inicializan a 1, 2, 3 y 4, y se incrementarán en 0.1 en cada iteración. Estos valores serán recibidos y monitorizados por el dispositivo HIL. Los resultados se muestran en la Figura 3.

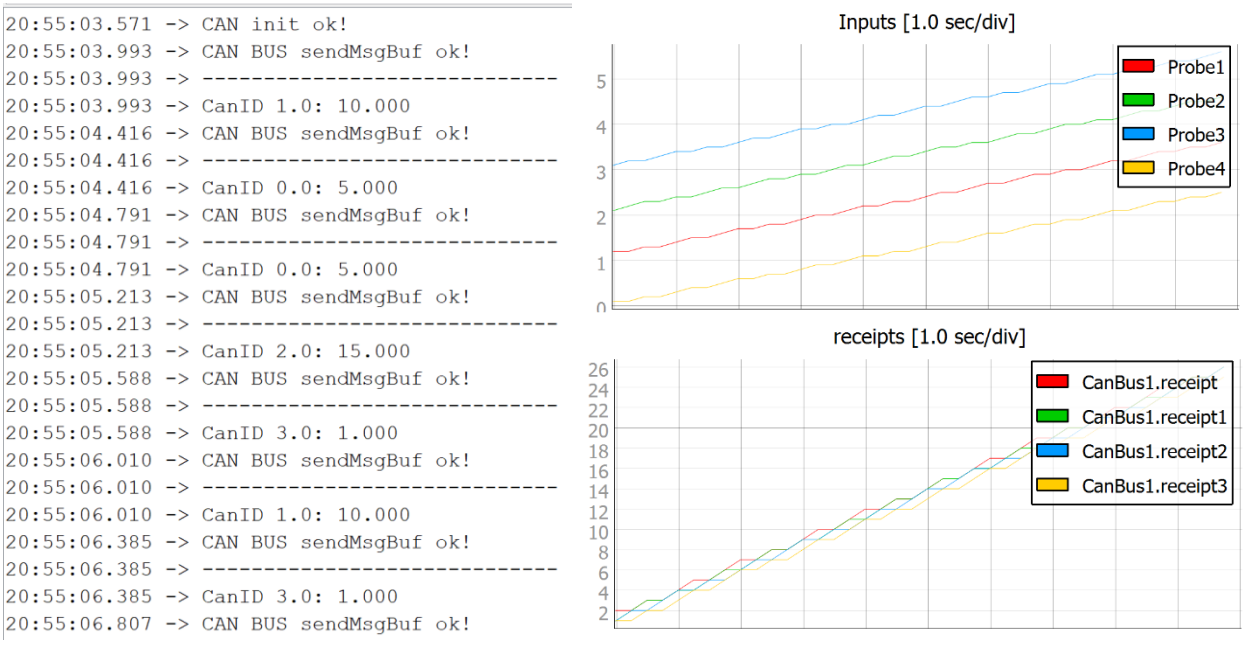

Figura 3. Resultado de la prueba de comunicación CAN Bus

En la parte izquierda de la Figura 3, el Serial Port Monitor del microcontrolador Arduino recoge los datos recibidos desde el HIL404 a lo largo de un breve fragmento de tiempo. Se puede comprobar fácilmente que los valores (con sus respectivos identificadores de mensaje -CanID- de 0 a 3) coinciden con las constantes 5, 10, 15 y 1 mencionadas anteriormente. En la parte superior derecha, vemos la evolución de las variables enviadas desde el Arduino al dispositivo HIL (aumentando en 0,1 cada vez que se envían). Por último, la gráfica inferior derecha representa simplemente un contador del número de veces que se recibe cada señal.

Una vez demostrada la efectividad de la comunicación CAN Bus entre los generadores y sus respectivos controladores locales, se aborda la siguiente y última capa de comunicación: el intercambio inalámbrico de información entre los controladores locales situados en cada RGD. Nos referiremos a esta interconexión como comunicación CAN Bus - XBee y, como antes, se incluyen aquí únicamente los resultados de la prueba que se ha llevado a cabo.

Al igual que en el test anterior, no se utilizará en este caso el algoritmo Robust Distributed Primal-Dual completo, sino que nos limitaremos a intercambiar algunos valores constantes entre dos nodos. Pasaremos al primer HIL404 las mismas entradas constantes que antes (5, 10, 15 y 1), mientras que el segundo nodo emitirá los valores 20, 25, 30 y 2 (pese a no ser relevante en este momento, el valor 1 enviado desde el primer nodo y el valor 2 enviado desde el segundo se corresponden con el identificador de cada bus, cuyo conocimiento es requerido por cada controlador local para ejecutar el algoritmo Robust Distributed Primal-Dual). Por lo tanto, esperamos ver que cada microcontrolador Arduino recibe a través de CAN Bus los valores emitidos por su propio HIL404 por un lado, y los datos XBee procedentes de los controladores vecinos por otro. Estos últimos, a su vez, deberían ser transmitidos de nuevo al dispositivo HIL de su propio nodo a través de CAN Bus.

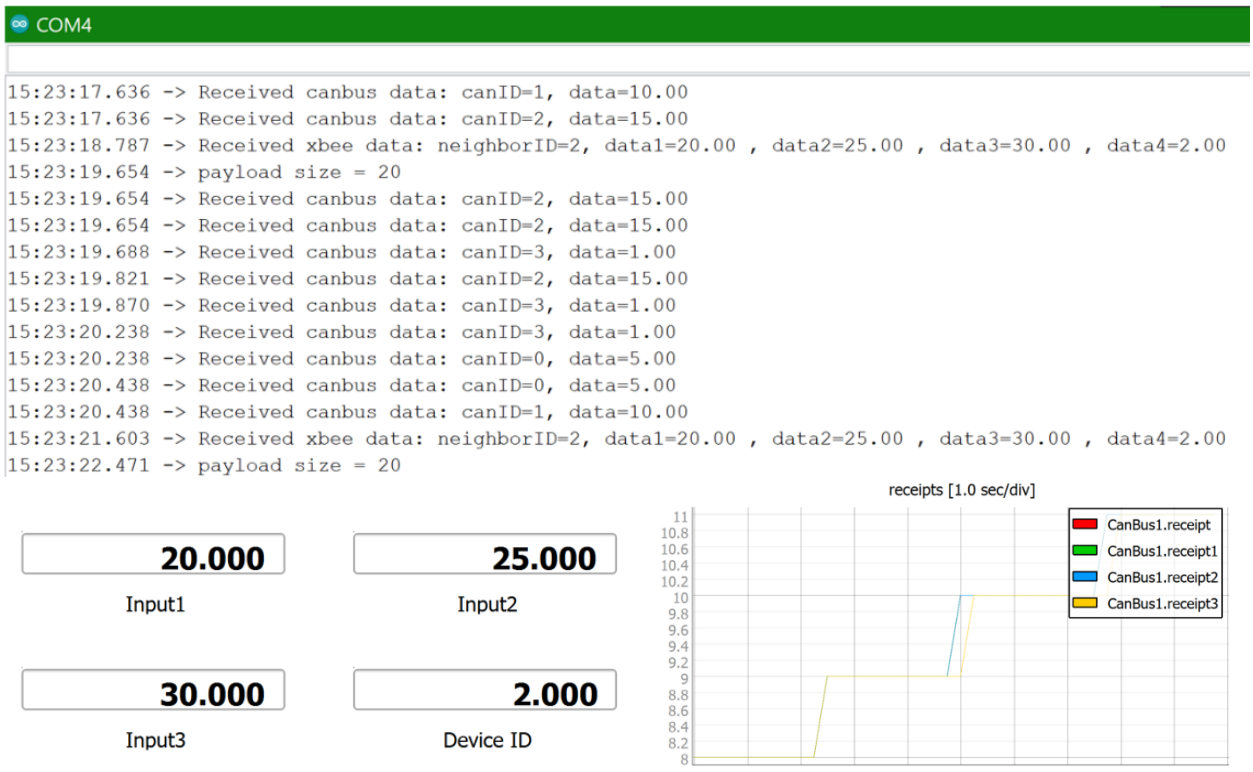

Figura 4. Resultados de la prueba de comunicación CAN Bus – XBee correspondientes al nodo 1

La Figura 4 trata de recoger estos resultados para el nodo 1. Se ha incluido la cabecera del Serial Port Monitor de Arduino con el número de puerto (COM4) para facilitar la distinción entre nodos.

El controlador local recibe vía CAN Bus las constantes transmitidas por el HIL404 de su mismo nodo (1), pero también recibe vía XBee los valores emitidos por el nodo 2 (neighborID=2, data=20, 25, 30, 2). Finalmente, esta información es enviada a su vez a su propio RGD, de nuevo a través del protocolo de comunicación CAN Bus. La Figura 4 muestra el Serial Port Monitor del microcontrolador Arduino (arriba), cuatro displays digitales con las cuatro señales XBee finalmente recibidas por el HIL404 -desde el Arduino- procedentes del nodo 2 (abajo a la izquierda), y la evolución del contador de paquetes de información recibidos (abajo a la derecha). Los mismos resultados se presentan en la Figura 5 para el nodo 2.

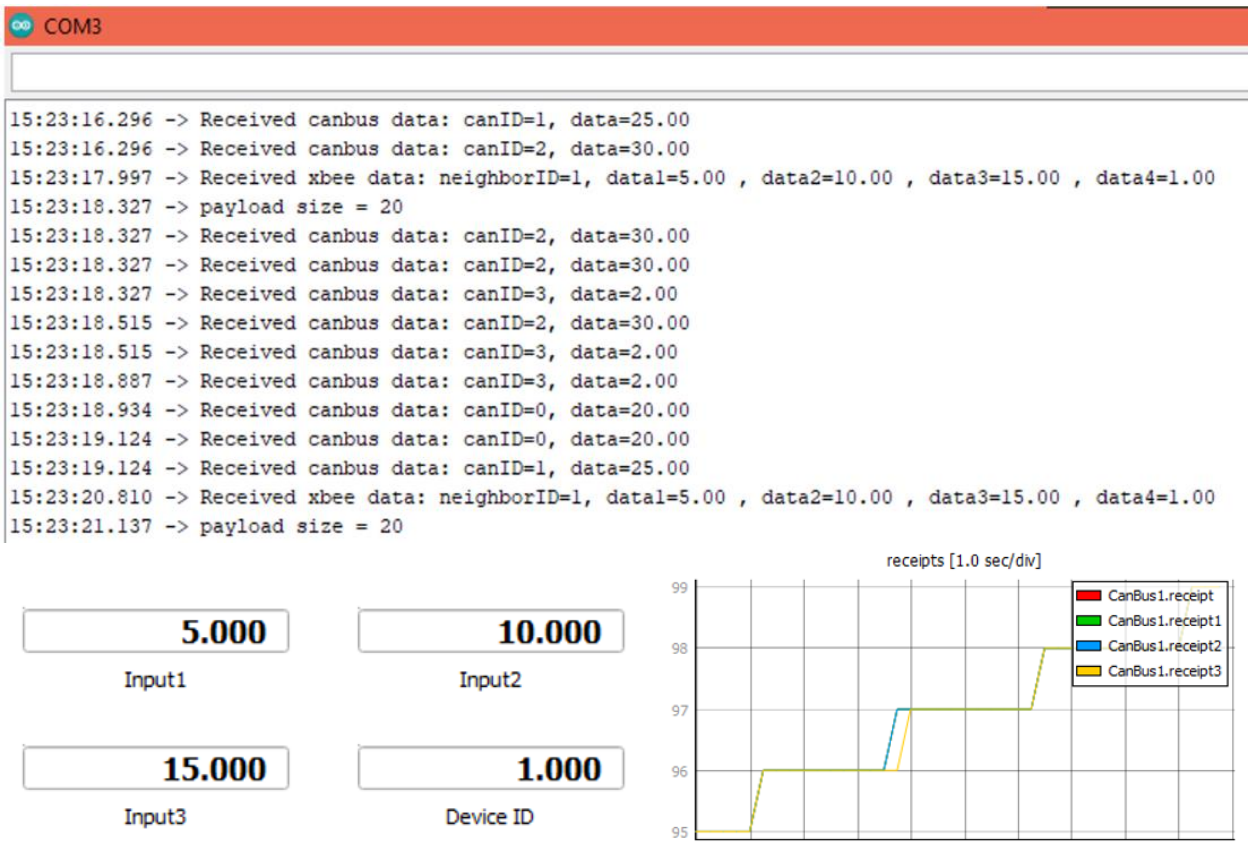

Figura 5. Resultados de la prueba de comunicación CAN Bus – XBee correspondientes al nodo 2

Una vez desarrollados los algoritmos de control distribuido y demostrada la capacidad de estos protocolos de comunicación para el intercambio inalámbrico de información entre los nodos, se tienen todos los elementos para proceder a la implementación la regulación terciaria de manera distribuida.

Desgraciadamente, debido a limitaciones de tiempo y otras dificultades, la implementación del algoritmo Robust Distributed Primal-Dual en un banco de pruebas de laboratorio no pudo llevarse a cabo en la práctica. Se experimentaron múltiples problemas de conectividad al intentar cargar y ejecutar los modelos en los cuatro dispositivos HIL, así como al utilizar una configuración de arranque independiente para activar los modelos automáticamente sin conexión con un ordenador. Además, posiblemente debido a la falta de una sincronización de reloj adecuada y de protocolos de inicialización como los incluidos en [1], o tal vez debido a detalles menores en el código que han podido resultar cruciales en la práctica, a menudo se ha incurrido en problemas de sobrecarga de datos, la aparición de valores infinitos y, en el mejor de los casos, la no convergencia de las potencias de salida de los generadores.

5. Conclusión

El proyecto aquí propuesto ha intentado demostrar que una arquitectura de control de generación distribuido, basada en el intercambio de información local entre nodos vecinos y en sencillas computaciones, puede realizar las mismas funciones que una estructura centralizada. Además, un enfoque distribuido aumentaría la adaptabilidad del sistema para añadir o eliminar unidades de generación sin afectar al resto de la red, y evitaría la necesidad de un procesador central responsable de almacenar y gestionar enormes cantidades de información y coordinar la respuesta de la red.

Aunque no ha sido posible completar con éxito una simulación completa de regulación terciaria a nivel de laboratorio, el trabajo descrito en este documento ha sido capaz de programar y modelar varios algoritmos de control distribuido y ha demostrado -mediante simulaciones por ordenador- la capacidad del algoritmo Robust Distributed Primal-Dual para implementar la regulación terciaria de forma distribuida. Además, se ha desarrollado y probado el rendimiento de los protocolos de comunicación CAN Bus y CAN Bus - XBee en el intercambio de información local entre los nodos de la red. Todo ello se suma al trabajo previo sobre arquitecturas de control de generación distribuido para microgrids de CA en isla - un pequeño fragmento del cual puede encontrarse entre las referencias utilizadas en la redacción de este proyecto- y abre puertas a nuevos estudios que continúen lo aquí propuesto.

6. Referencias

[1] S. T. Cady, A. D. Domínguez-García and C. N. Hadjicostis, “A Distributed Generation Control Architecture for Islanded AC Microgrids,” *IEEE Transactions on Control Systems Technology*, vol. 23, no. 5, pp. 1717-1735, Septiembre 2015.

[2] A. Gómez-Expósito, A. J. Conejo and C. Cañizares, *Electric Energy Systems: Analysis and Operation (Electric Power Engineering Series)*, 1ª ed., CRC Press, 2009, pp. 355-400.

[3] S. Pullins, "Why microgrids are becoming an important part of the energy infrastructure," *The Electricity Journal*, vol. 32, no. 5, pp. 17-21, Junio 2019.

[4] A. J. Wood and B. F. Wollenberg, *Power Generation, Operation, and Control*, New York, NY, USA: Wiley, 1996.

[5] M. Zholbaryssov, C. N. Hadjicostis and A. D. Domínguez-García, "Fast Coordination of Distributed Energy Resources over Time-Varying Communication Networks," 2019. Disponible online en: https://doi.org/10.48550/arXiv.1907.07600. Actualizado por última vez el 4 de mayo de 2020 a las 06:37:38 UTC.

IMPLEMENTATION OF MICROGRID DISTRIBUTED CONTROL ARCHITECTURES ON AN INDUSTRIAL-GRADE CONTROL HARDWARE PLATFORM

Author: Martínez Rivera, Ginés.

Supervisor: Dominguez-Garcia, Alejandro.

Collaborating Entity: University of Illinois at Urbana-Champaign

ABSTRACT

The distributed architecture for Islanded AC Microgrids which we will be working on, replicates and intends to optimize the functionality of their generation control architecture in a distributed fashion. In this project, we will synthesize numerous distributed algorithms (programmed in C-language) and we will implement them on several industrial-grade control hardware units (HIL's 4th Generation Typhoon HIL404). Each HIL device will communicate with an Arduino Due microcontroller via CAN Bus communication protocol, which in turn will be able to wirelessly exchange information with each other through embedded MaxStream XB24-DMCIT-250 rev B XBee modules. Finally, we will test and validate the algorithms and the communication network in a hardware-in-the-loop testbed under a variety of scenarios.

Keywords: microgrid; distributed generation control architecture; distributed control algorithms; optimal dispatch; Distributed Generation Resources (DGR); CAN Bus; CAN Bus – XBee.

1. Introduction

The nascent field of microgrids is attracting increasing attention and research that seeks to harness its potential. It is envisioned that current large and centralized power systems will evolve towards more decentralized networks [3]. The work presented herein is intended to take advantage of microgrids' inherit characteristics to optimize their generation control architecture.

A microgrid can be generically defined as a network of interconnected loads and generators, which is physically smaller, has lower power ratings and capacities than large power systems, and can operate in isolation from the utility grid. Despite these differences, the normal operation of both microgrids and large power systems, e.g., bulk power transmission networks, can be reduced to three requirements: (i) supply must match demand, (ii) frequency must be regulated to its nominal value (50 Hz in Europe, 60 Hz

in the United States), and (iii) costs of generation must be minimized [1]. Load variations or faults in generators and transmission lines can cause a mismatch in the system.

To deal with these situations, any type of ac power system implements a three-layered (primary, secondary and tertiary) generation control architecture. Primary regulation acts instantaneously and is responsible for balancing supply and demand, while, at the same time, it ensures that frequency does not deviate in excess from its nominal value. Secondary control layer, also called frequency regulation, restores nominal frequency and maintains appropriate power interchange between control areas. Finally, tertiary regulation -optimal dispatch- is responsible for adjusting the supply levels of the different generating units so that the total generation cost of the system is minimized [2], [4].

Conventionally, for large power systems, only the first layer -droop control- of the three previously mentioned relies on local information. Secondary and tertiary control layers, i.e., frequency regulation and optimal dispatch, are based on a centralized control architecture [4]. In other words, a centrally located computer is responsible for gathering information from all the generator buses and coordinating the control functions. However, this centralized architecture involves a great computational complexity since the central computer needs to receive and manage a huge amount of information from every gen-bus at every iteration.

2. Project Definition

The centralized control strategy is already well established in the current operation of large electrical power systems, but it is not in the emerging field of microgrids [1]. Besides, its structural characteristics make microgrids particularly suitable for a distributed control architecture. In this way, the controllers located at each distributed generation resource (DGR)[2] could collectively perform the same function as a central computer by simply exchanging local information with their neighboring controllers.

Throughout the project, it will be demonstrated how the distributed control architecture for islanded ac microgrids can replicate the functionality of the frequency regulation and optimal dispatch -secondary and tertiary control layers- of large electrical systems. This implementation has clear advantages over traditional centralized generation controls, since it does not require either full knowledge of the type and characteristics of each generator or a complex communication network between each DGR and a central

[2] *Distributed Generation Resource* (*DGR*) will refer to both synchronous generators and inverter-interfaced power supplies [1].

processor. A distributed control structure would also increase the adaptability of the system to add or remove generating units without affecting its normal operation [1].

3. General Description of the System

The distributed generation control architecture will be implemented on a laboratory-grade microgrid comprised of four synchronous generators interconnected with several resistive loads. Each DGR (HIL's 4th Generation Typhoon HIL404) is equipped with an Arduino Due microcontroller that is capable of exchanging information (bi or unidirectionally) with its neighboring controllers via embedded MaxStream XB24-DMCIT-250 rev B XBee modules. A representative schematic of the system is shown in Figure 6.

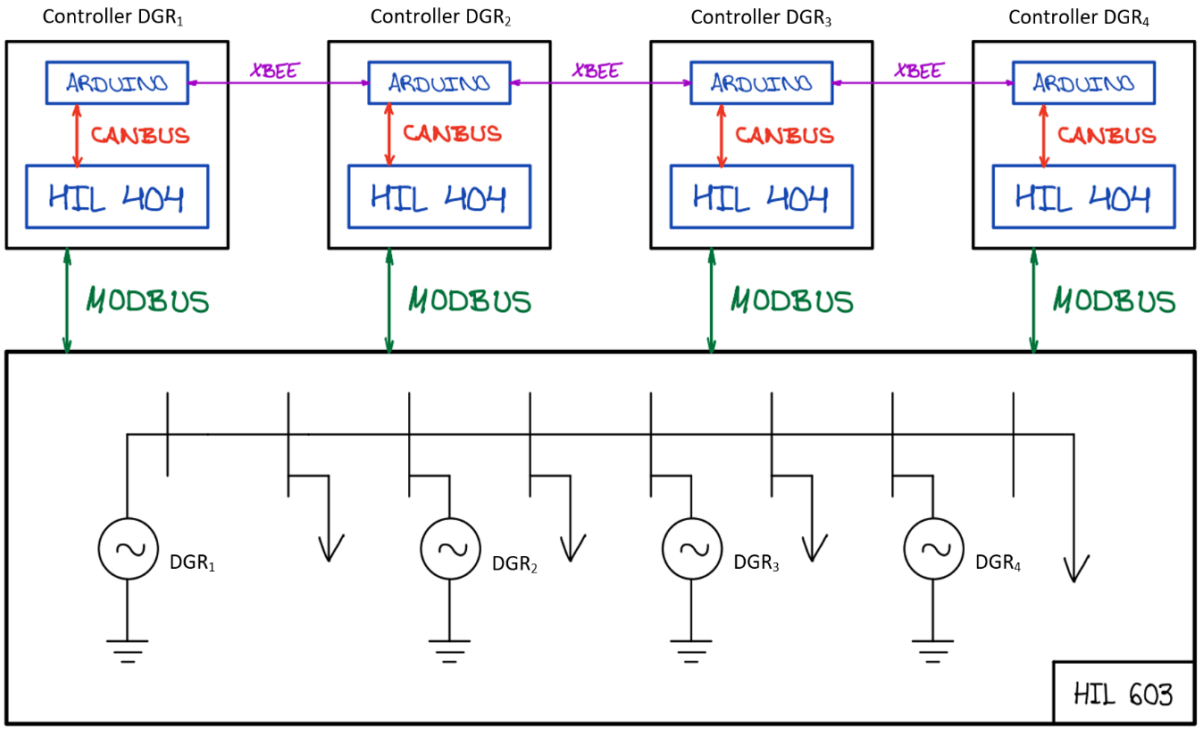

Figure 6. General diagram of the project

The first and main task will require the synthesis of distributed algorithms onto four industrial-grade control hardware devices (HIL's 4th Generation Typhoon HIL404), each one of them representing a DGR. These distributed algorithms will enable the generators to perform both frequency regulation and, mainly, optimal dispatch.

Once the code is implemented on the hardware control units, the next step will be to allow each of the HIL devices (in practice, the DGRs) to interact with their respective Arduino microcontrollers via CAN

Bus communication protocol. These local controllers will be able to wirelessly send and receive information to and from their neighborhood through embedded XBee modules.

Finally, although this will not be addressed in this project, this four-node system will be integrated via ModBus communication protocol into a laboratory-grade real-time simulation of the microgrid (HIL 603 device) to test and validate it under a variety of scenarios.

4. Results

The first stage of the distributed generation control implementation, depicted in Figure 6, consists of developing and programming the distributed algorithms. First, we address the so-called Ratio-Consensus, a linear iterative algorithm which will serve as the starting point to execute frequency regulation and optimal dispatch -secondary and tertiary generation control layers- on a distributed basis. However, Ratio-Consensus happens to be insufficient when it comes to its practical implementation, since it does not account for the loss of information packages when traveling among nodes. Therefore, a robust extension of this algorithm -Robust Ratio-Consensus- must be developed.

Finally, we use these preliminary distributed algorithms to program and model Robust Distributed Primal-Dual algorithm, which will be proved capable of solving the optimal dispatch problem in an AC microgrid in a distributed fashion. Let $f_i(p_i) = a_i \cdot p_i^2,\ i = 1, 2, 3, 4$, denote the cost functions of the four DGRs of the microgrid in Figure 6 [5]. Then, Figure 7 includes the convergence of the algorithm to different values of the a_i parameters.

Figure 7 (top) shows the case when $a_i = 0.1,\ i = 1, 2, 3, 4$, i.e., the four generators have identical supply costs. The result is that the power outputs sinusoidally converge to 0.4 pu, matching the total demand, previously set to 1.6 pu. In Figure 7 (middle), the DGRs' cost functions have been modified so that $a_1 = a_2 = 0.1,\ a_3 = a_4 = 0.4$. Although the curves overlap, it is easy to see that the generated power by generators 1 and 2 (*gamma9* and *gamma10*) is greater that the power supplied by 3 and 4 (*gamma11* and *gamma12*). Lastly, Figure 7 (bottom) considers the situation in which all the DGRs have different cost-efficiency: $a_1 = 0.1, a_2 = 0.2, a_3 = 0.3, a_4 = 0.4$. As before, the power produced by each of them is inversely correlated to the cost of its power output. One can easily check that the steady-state power supplies add up to meet the total demand of 1.6 pu.

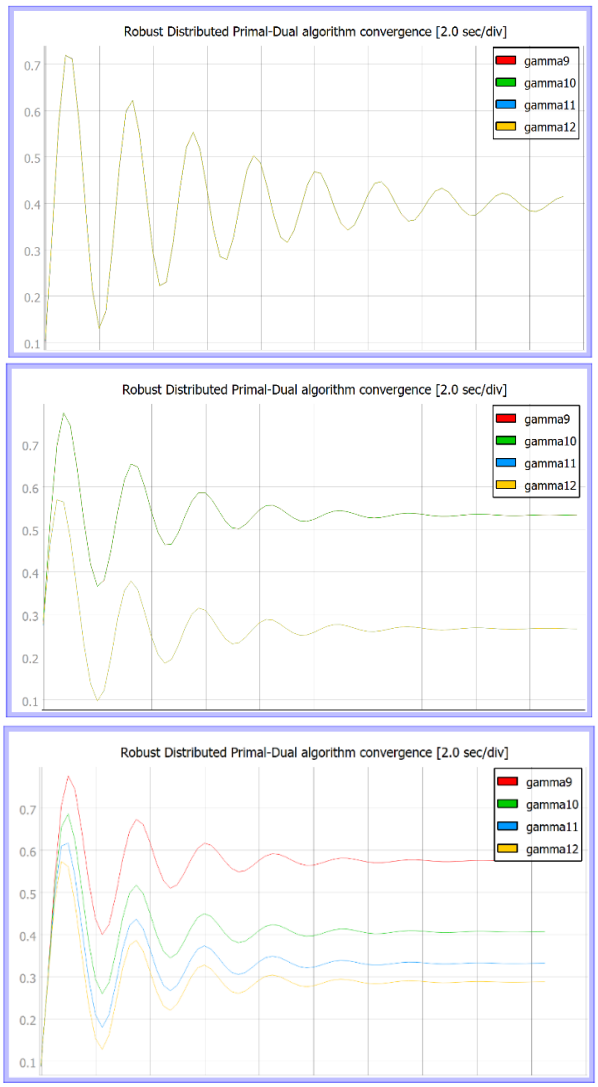

Figure 7. Robust Distributed Primal-Dual algorithm successful convergence

With these computer simulations we intend to illustrate that Robust Distributed Primal-Dual algorithm can be used indeed to implement optimal dispatch in a distributed fashion. To this end, we first need to describe and test the communication protocols that will allow for the exchange of information among nodes.

First, once the distributed control algorithms have been synthesized onto the Typhoon HIL404 devices, the bidirectional exchange of information between these -the DGRs- and the local controllers -Arduino boards with embedded XBee modules- has been addressed via the Controller Area Network (CAN or CAN Bus) protocol. Further details will be omitted in this summary, but we include below the results of the CAN Bus communication test.

Instead of running the full optimal dispatch algorithm, four constant values (5, 10, 15 and 1) will be sent from the HIL DGR to its Arduino-XBee local controller. In turn, the Arduino will read and process this information, and will also send four other variables from the microcontroller to the HIL404. These are initialized to the values 1, 2, 3, and 4, and will be increased by 0.1 in each iteration. They will be received and monitored by the HIL device. The results are depicted in Figure 8.

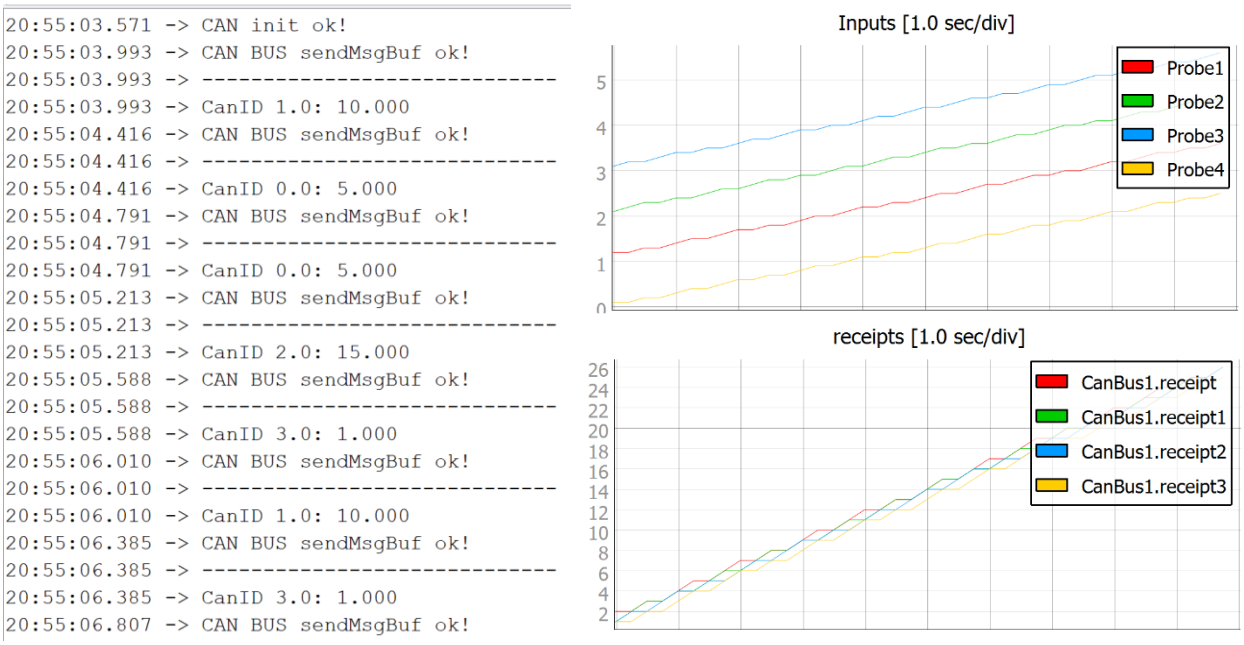

Figure 8. CAN Bus communication test results

On the left-hand side, the Serial Port Monitor of the Arduino microcontroller gathers the data received from the HIL404 over a fragment of time. One can easily check that the values (with their respective message identifiers -CanID- from 0 to 3) match the aforementioned constants 5, 10, 15, and 1. At the top right, we see the evolution of the variables transmitted from the Arduino to the HIL device (increasing by 0.1 each time they are sent). Finally, the graph at the bottom right is simply a counter of the number of times each signal is received.

Once CAN Bus communication between the DGRs and their respective local controllers have been demonstrated successful, we address the following and last communication layer, i.e., the wireless exchange of information among the local controllers located at each DGR. This will be referred to as CAN Bus – XBee communication and, as before, we limit ourselves to include here only the results of the test that has been conducted.

As in the previous test, we will not be using the Robust Distributed Primal-Dual algorithm but exchanging some constant values between two nodes, since the only objective here is to test the performance of the communication protocol. We are passing to the first HIL404 the same inputs as before (5, 10, 15 and 1), whereas the second node will broadcast the values 20, 25, 30 and 2 (not relevant here, but the value 1 sent from the first node and the value 2 sent from the second, correspond to the identifier of each bus, whose knowledge is required by each local controller to execute Robust Distributed Primal-Dual algorithm). Therefore, we expect to see every Arduino microcontroller receiving CAN Bus data from its own HIL404 on one side, and XBee data coming from the neighboring controllers on the other, which in turn should be able to communicate back to its HIL device via CAN Bus.

Figure 9 tries to gather these results for node 1. The header of the Arduino Serial Port Monitor with the port number (COM4) has been included for an easier distinction between nodes.

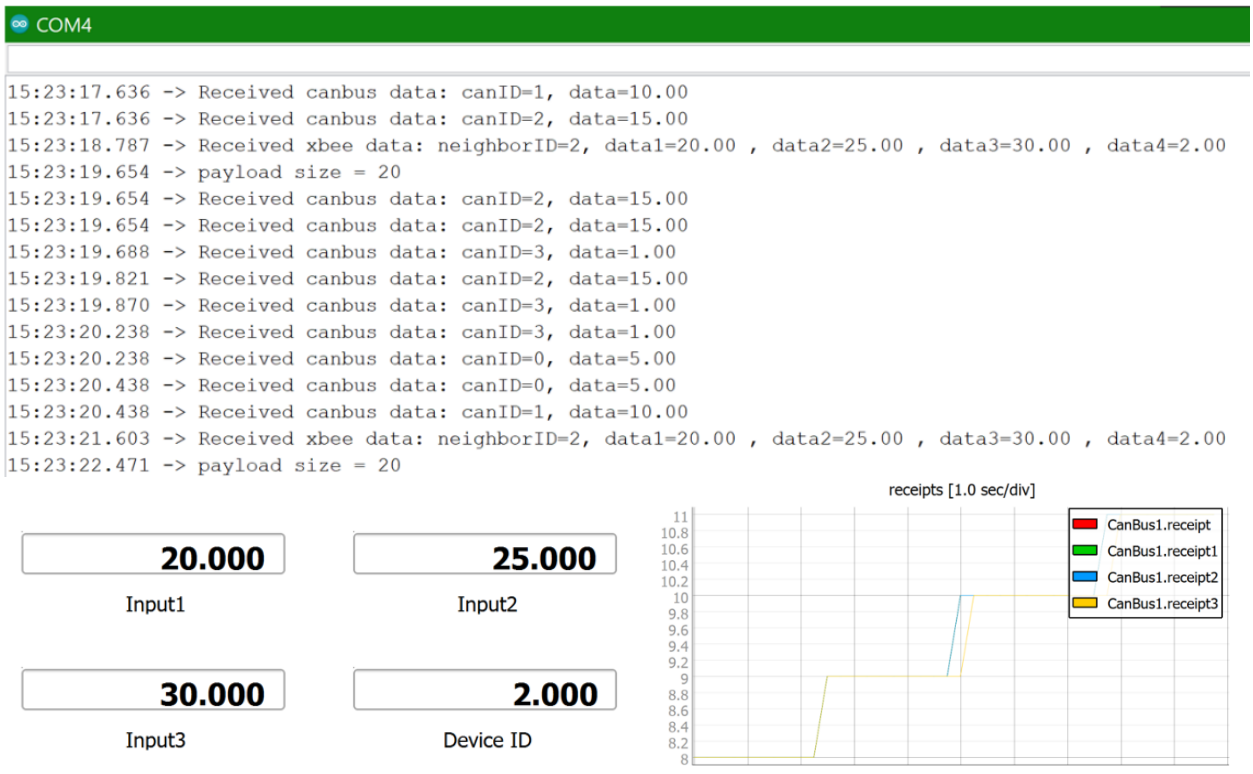

Figure 9. Node 1 CAN Bus – XBee communication test results

Clearly, the local controller receives via CAN Bus the constant inputs passed to the HIL404 of its same node (1), but it also receives XBee data taking the values broadcasted by node 2 (*neighborID=2, data=20, 25, 30, 2*). Then, this information is sent to its own HIL device. Figure 9 shows the Serial Port Monitor of the Arduino microcontroller (top), four digital displays with the four XBee signals finally received by the

HIL404 from the Arduino coming from node 2 (bottom left) and the evolution of the counter of received information packets (bottom right). The same results are depicted in Figure 10 for node 2.

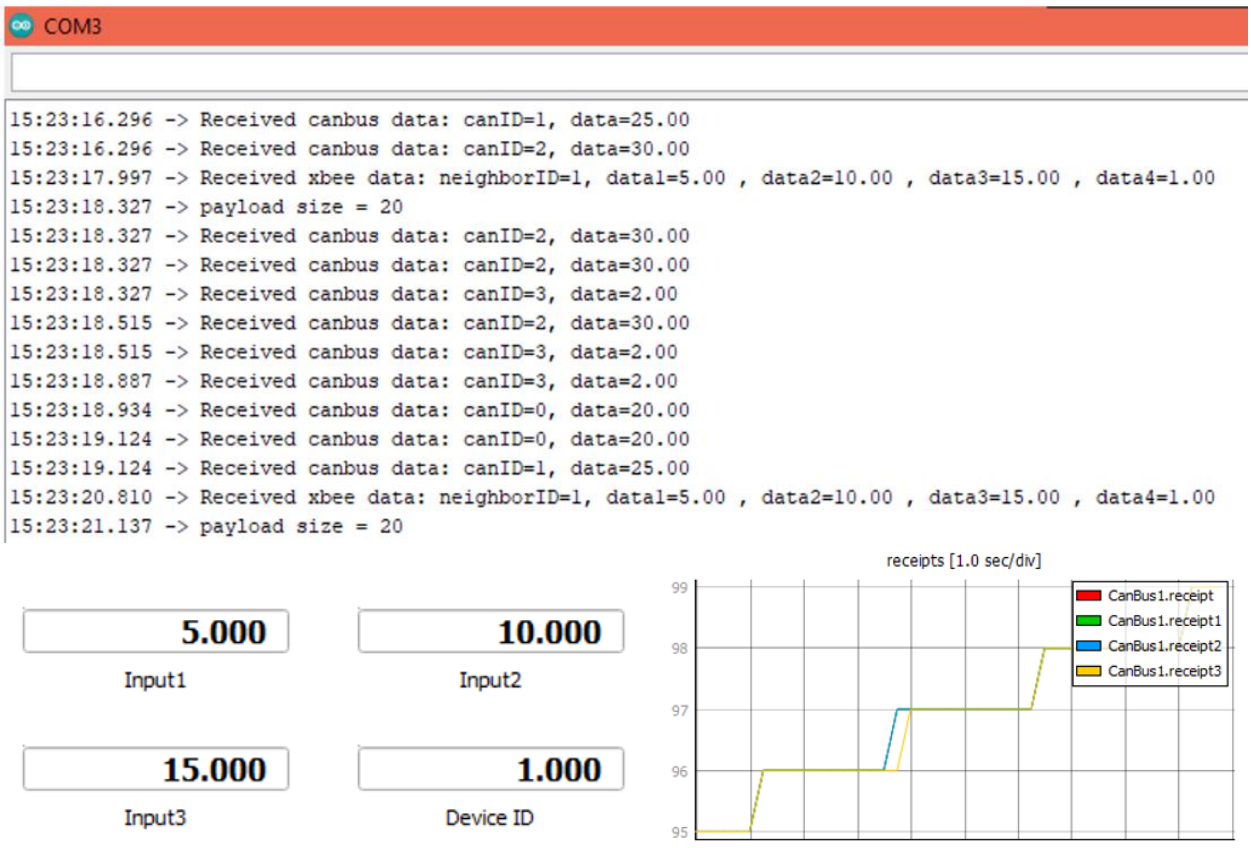

Figure 10. Node 2 CAN Bus – XBee communication test results

Now that the distributed control algorithms have already been developed, and these communication protocols have been proved capable of wirelessly exchanging information among nodes, we have all the elements to implement optimal dispatch in a distributed fashion.

Unfortunately, due to time constraints and other difficulties, the implementation of the Robust Distributed Primal-Dual algorithm in a laboratory testbed could not be accomplished in practice. We experienced multiple connectivity issues when trying to upload and run the models on the four HIL devices; and we struggled to use the HIL standalone boot configuration to activate the models automatically with no connection to a computer. Moreover, possibly due to the lack of a proper clock synchronization and initialization protocols as included in [1], or maybe because of minor details in the code that happened to become crucial in practice, we often ran into data overrun issues, the appearance of NaN values, and, in the best-case scenario, the non-convergence of the generators' power outputs.

5. Conclusion

The project here proposed has attempted to prove that a distributed generation control architecture, based on local information exchanged with the neighboring nodes and simple computations, can perform the same functions as its centralized counterpart. Moreover, a distributed approach would increase the adaptability of the system to add or remove generating units without affecting the rest of the network, while avoiding the need for resource-consuming central processor.

Even though we have not been able to complete a successful optimal dispatch simulation in a laboratory testbed, the work described in this document has programs and models several distributed control algorithms and proved -with software simulations- the capability of Robust Distributed Primal Dual algorithm to distributively implement optimal dispatch. Moreover, we develop and test the performance of CAN Bus and CAN Bus – XBee communication protocols in the exchange of local information between the nodes participating in the distributed algorithms. All this adds to the existing groundwork on distributed generation control architectures for islanded AC microgrids -a piece of which can be found among the references used in the writing of this thesis- and allows for further developments based on what has been proposed herein.

6. References

[1] S. T. Cady, A. D. Domínguez-García and C. N. Hadjicostis, "A Distributed Generation Control Architecture for Islanded AC Microgrids," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 5, pp. 1717-1735, September 2015.

[2] A. Gómez-Expósito, A. J. Conejo and C. Cañizares, *Electric Energy Systems: Analysis and Operation (Electric Power Engineering Series)*, 1st ed., CRC Press, 2009, pp. 355-400.

[3] S. Pullins, "Why microgrids are becoming an important part of the energy infrastructure," *The Electricity Journal*, vol. 32, no. 5, pp. 17-21, June 2019.

[4] A. J. Wood and B. F. Wollenberg, *Power Generation, Operation, and Control*, New York, NY, USA: Wiley, 1996.

[5] M. Zholbaryssov, C. N. Hadjicostis and A. D. Domínguez-García, "Fast Coordination of Distributed Energy Resources over Time-Varying Communication Networks," 2019. Available online at: https://doi.org/10.48550/arXiv.1907.07600. Last updated 4 May 2020 06:37:38 UTC.

Contents

1. Introduction

1.1. Microgrids. Power Systems' Operation.

The nascent field of microgrids is attracting increasing attention and research that seeks to harness its potential. It is envisioned that current large and centralized power systems will evolve towards more decentralized networks [10]. The work presented herein is intended to take advantage of microgrids' inherit characteristics to optimize their generation control architecture.

A microgrid can be generically defined as a network of interconnected loads and generators, which is physically smaller, has lower power ratings and capacities than large power systems, and can operate in isolation from the utility grid. Despite these differences, the normal operation of both microgrids and large power systems, e.g., bulk power transmission networks, can be reduced to three requirements: (i) supply must match demand, (ii) frequency must be regulated to its nominal value (50 Hz in Europe, 60 Hz in the United States), and (iii) costs of generation must be minimized [4]. Load variations or faults in generators and transmission lines can cause a mismatch in the system.

When generation overcomes demand, the excess of supply is transformed into an excess of kinetic energy that, in turn, involves an increase in the rotating speed of the generators and the frequency of the system. When demand is greater than supply, the inverse process leads to a frequency drop. On the one hand, sub-frequencies can cause refrigeration difficulties and even the alternators to resonate. Under Frequency Load Shedding (UFLS) mechanisms are meant to prevent frequency drops. Over-frequencies, on the other hand, increase losses due to hysteresis, Foucault currents, and frictions [8].

To deal with these situations, any type of ac power system implements a three-layered (primary, secondary and tertiary) generation control architecture. Primary regulation acts instantaneously and is responsible for balancing supply and demand, while, at the same time, it ensures that frequency does not deviate in excess from its nominal value. Secondary control layer, also called frequency regulation, restores nominal frequency and maintains appropriate power interchange between control areas. Finally, tertiary regulation -optimal dispatch- is responsible for adjusting the supply levels of the different generating units so that the total generation cost of the system is minimized [8], [16].

1.2. Introduction to Distributed Generation Control

Conventionally, for large power systems, only the first layer -droop control- of the three previously mentioned relies on local information. Secondary and tertiary control layers, i.e., frequency regulation and optimal dispatch, are based on a centralized control architecture [16]. In other words, a centrally located computer is responsible for gathering information from all the generator buses and coordinating the control functions. However, this centralized architecture involves a great computational complexity since the central computer needs to receive and manage a huge amount of information from every gen-bus at every iteration.

As mentioned, this control strategy is already well established in the current operation of large electrical power systems, but it is not in the emerging field of microgrids [3]. Besides, its structural characteristics make microgrids particularly suitable for a distributed control architecture. In this way, the controllers located at each distributed generation resource (DGR)[3] could collectively perform the same function as a central computer by simply exchanging local information with their neighboring controllers.

Throughout this thesis, it will be demonstrated how the distributed control architecture for islanded ac microgrids can replicate the functionality of the frequency regulation and optimal dispatch -secondary and tertiary control layers- of large electrical systems. This implementation has clear advantages over traditional centralized generation controls: it does not require either full knowledge of the type and characteristics of each generator or a complex communication network between each DGR and a central processor. A distributed control structure would also increase the adaptability of the system to add or remove generating units without affecting its normal operation [3].

1.3. Project Overview

The distributed generation control architecture will be implemented on a laboratory-grade microgrid comprised of four synchronous generators interconnected with several resistive loads. Each DGR is equipped with an Arduino microcontroller that is capable of exchanging information (bi or unidirectionally) with its neighboring controllers via wireless XBee transceivers. A representative schematic of the project is shown in Figure 11.

[3] In the remainder, the term *Distributed Generation Resource* (*DGR*) will collectively refer to both synchronous generators and inverter-interfaced power supplies [3].

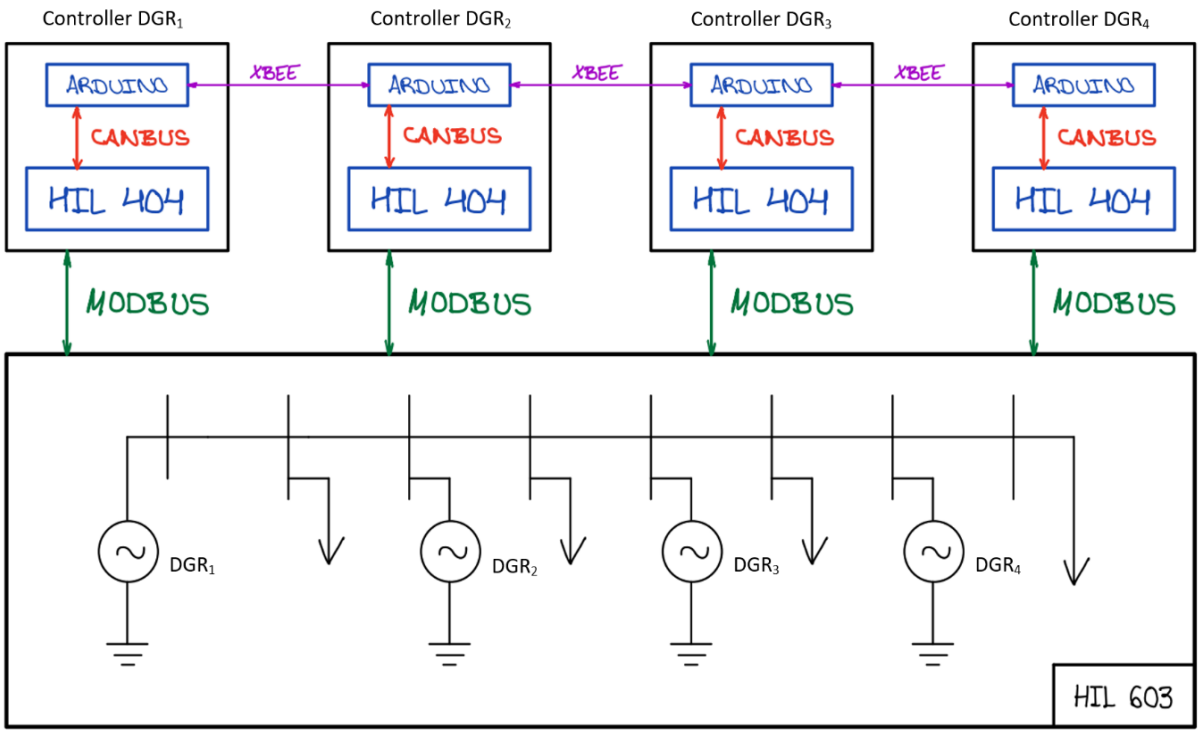

Figure 12. General diagram of the project.

The first and main task will require the synthesis of distributed algorithms onto four industrial-grade control hardware devices (HIL's 4th Generation Typhoon HIL404), each one of them representing a DGR. These distributed algorithms will enable the generators to perform both frequency regulation and, mainly, optimal dispatch.

Once the code is implemented on the hardware control units, the next step will be to allow each of the HIL devices (in practice, the DGRs) to interact with their respective Arduino microcontrollers via CAN Bus communication protocol. These local controllers will be able to wirelessly send and receive information to and from their neighborhood through embedded XBee modules.

Finally, although this will not be addressed in this thesis, this four-node system will be integrated via ModBus communication protocol into a laboratory-grade real-time simulation of the microgrid (HIL 603 device) to test and validate it under a variety of scenarios.

2. Literature Review: Distributed Generation Control Architecture for Islanded AC Microgrids

2.1. Preliminaries

This section introduces a theoretical framework for distributively implementing a generation control architecture in ac microgrids. First, a brief overview is included of the control functions of this architecture, which are essentially parallel to those applicable to large power systems. Then, it will follow a dynamical analysis of the generators' behavior, together with the description of the cyber communication network interconnecting the local controllers of the DGRs. Finally, we will formulate an iterative algorithm that constitutes a starting point for implementing frequency regulation and optimal dispatch on a distributed basis.

2.1.1. Control Objectives

As it has been previously stated, the generation control objectives for islanded AC microgrids are equivalent to those of bulk power systems [3]. The only, yet crucial, difference is that, in this case, these objectives will be achieved through a distributed computation, while traditional electrical systems commonly rely on centralized architectures. This distributed computation will be based on a simple iterative algorithm described in the next chapter. For the time being, a brief discussion is provided of the control functions of each of the three layers that typically characterize the generation control. The reader is referred to [8] and [16] for further information.

1) *Droop Control*

Also referred to as *primary generation control*, it acts instantaneously (less than one minute) to balance supply and demand after a load variation or any type of fault occurs. It is based on a proportional control; thus, it does not completely restore the frequency to its nominal value. Moreover, and more importantly concerning this project, droop control is executed on a decentralized basis, since it only relies on local measurements and actions taken by each DGR [3]. This means that there is no need for a distributed alternative to primary regulation and, therefore, it will not be addressed in the remainder of this thesis.

2) *Frequency Regulation*

The secondary generation *control* or *automatic generation control* (AGC) relies on an integral control that adjusts the generators' set-points to restore the frequency to its nominal value (typically 50/60 Hz). It is slower than primary control, and it allows to recover the primary reserve of the generating units. At the same time, it is responsible for eliminating the error in the interchange of power between control areas. Nevertheless, since no control areas can be distinguished within a microgrid, this functionality will not be considered either in the following developments. Finally, its implementation is centralized: a central computer gathers data and measurements from every generating unit and coordinates the system's response.

3) *Optimal Dispatch*

The *tertiary control* reallocates the power supply among the generating units in the most economically efficient way. In other words, assuming that the cost derived from each generator is dependent on its power output, the objective is to minimize the total generation cost.

As well as the frequency regulation, this third layer of the control architecture is implemented via a centrally located computer [3]. This central controller requires information of, e.g., cost functions, maximum and minimum power outputs of each DGR, total supply and demand of the system, the magnitude and angle of the phasor associated with the voltage at each bus, and active and reactive power injections at every generator and load bus. This is just a sample of the operational complexity that this centralized control architecture involves, not to mention the amount of data that needs to flow through the cyber layer of the network at every iteration.

2.1.2. DGRs' Dynamics and Communication Network

Firstly, some notions about the dynamic behavior of the generating units of an islanded ac microgrid must be introduced in order to implement the aforementioned distributed frequency regulation and optimal dispatch. Then, we describe the communication network that will allow the exchange of information between the local controllers located at each generator. Figure 12 (extracted from [3]) below shows a graphical depiction of the physical (DGRs) and cyber layer (controllers) of a microgrid.

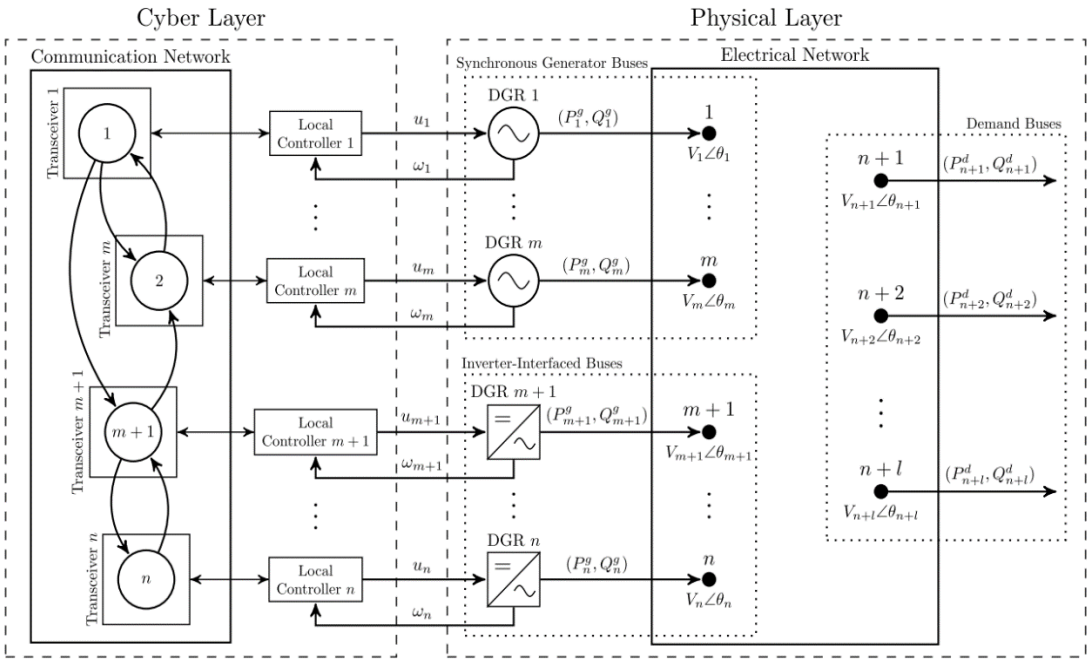

Figure 13. Block diagram describing the physical and cyber layer of a microgrid's distributed control architecture [3].

A) Physical Layer Model

The analysis of the DGRs' dynamics corresponds to the physical layer of the microgrid [3]. This section will briefly recall the dynamical models of both synchronous generators and inverter-interfaced power supplies. Although these models will not be addressed or used directly over the project, it has been deemed appropriate to include them here in order to provide an overview of how generators are capable of adjusting their power supply via their speed governors.

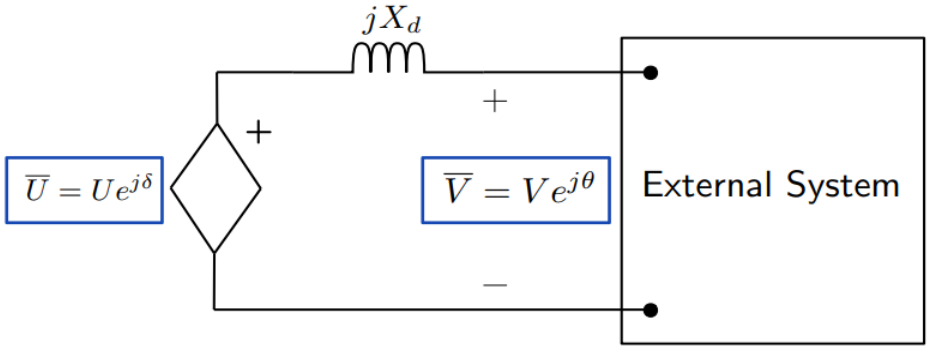

Figure 14. Per-phase equivalent circuit model of a synchronous generator [4].

Synchronous generators can be described as constant voltage sources behind a reactance as represented in Figure 13 (from [4]), where X_d accounts for the stator winding reactance [4]. For a synchronous generator that is injecting active power into the microgrid at bus i, let $\delta_i(t)$ denote the phase angle of the voltage source (behind the reactance) as measured with respect to a reference that rotates at the system nominal electrical frequency ω_0. Let also $\omega_i(t)$ denote the rotor electrical angular speed. Finally, $P_i^m(t)$ denotes the mechanical power applied to the generator, and P_i^r (t) its generation set-point. Then:

$$\frac{d\delta_i}{dt} = \omega_i - \omega_0$$

$$M_i \frac{d\omega_i}{dt} = -D_i(\omega_i - \omega_0) + P_i^m - V_i \sum_{k=1}^{n} V_k [G_{ik}\cos(\theta_i - \theta_k) + B_{ik}\sin(\theta_i - \theta_k)]$$

$$\tau_i \frac{dP_i^m}{dt} = -P_i^m - \frac{1}{R_i \omega_0}(\omega_i - \omega_0) + P_i^r$$

where n is the number of buses of the system, D_i [s/rad] is the so-called generator's damping coefficient, M_i [s^2/rad] is a scaled inertia constant, τ_i is the time constant of the generator, R_i [pu] is the speed-droop characteristic slope, and G_{ik} and B_{ik} are the real and imaginary part of the system's admittance matrix respectively [3]-[4].

Inverter-interfaced power supplies can be similarly described by the equivalent model shown in Figure 13 for synchronous generators [4], i.e., a constant voltage source behind a reactance. Therefore, for an inverter-interfaced power supply connected to bus i, let $\delta_i(t)$ denote the phase angle of the voltage source (behind the reactance) as measured with respect to a reference that rotates at the system nominal electrical frequency ω_0, and P_i^r (t) its generation set-point. Then:

$$\frac{d\delta_i}{dt} = \omega_i - \omega_0 = \frac{1}{H_i}[P_i^r - V_i \sum_{k=1}^{n} V_k [G_{ik}\cos(\theta_i - \theta_k) + B_{ik}\sin(\theta_i - \theta_k)]$$

where H_i [s/rad] is the speed-droop characteristic slope of the generating unit [3]-[4].

It is worth mentioning that no reference has been made to the reactive power injected by the generators nor to the voltage magnitudes at each bus, since they are not necessary for the development of frequency regulation and optimal dispatch algorithms that is addressed herein.

B) Cyber Layer Model

For the successful operation of the distributed generation control architecture, it is required that the DGRs can send and receive information from and to one another. This interconnection is achieved through the local controllers located at each generating unit, which conform the cyber layer of the microgrid. These are capable of both communicating with the respective generator and exchanging data with their neighboring controllers. In addition, the controllers can compute simple primitive algorithms that will allow for implementing frequency regulation and optimal dispatch [3]. These methods will be addressed in the following section but, for the moment, it will be briefly described the communication network through which the DGR's local controllers are interconnected.

Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ denote a directed graph, where $\mathcal{V} := \{1, 2, \dots, m\}$ is the set of generator buses, and $\mathcal{E} \subseteq \mathcal{V}\, x\, \mathcal{V}$ is the edge set with $(i, j) \in \mathcal{E}$ if and only if the controller located at node j can receive information from the controller at node i. By using a directed graph, we allow for cases in which bus i can send information to bus j ($(i, j) \in \mathcal{E}$), but not the other way around ($(j, i) \notin \mathcal{E}$).

We introduce next the concept of in-neighborhood of node i, i.e., the set of nodes from which the controller located at this bus can receive information: $\mathcal{N}_i^- := \{j \in \mathcal{V} : (j, i) \in \mathcal{E}\}$. Equivalently, the set of nodes to which bus i can send information can be defined as its out-neighborhood: $\mathcal{N}_i^+ := \{j \in \mathcal{V} : (i, j) \in \mathcal{E}\}$. Finally, the cardinality of the out-neighborhood will be denoted as $D_i^+ := |\mathcal{N}_i^+|$, and referred to in the remainder as the out degree of node i [15].

The correct coordination of the physical and cyber layers described above and schematized in Figure 12 will allow for distributively implementing frequency regulation and optimal dispatch. In this case, the distributed implementation of the secondary and tertiary generation control functions will rely on a primitive iterative model: the so-called Ratio-Consensus algorithm.

2.1.3. Ratio-Consensus Algorithm

To compute ratio-consensus, each DGR's local controller maintains two values denoted as internal states. At every iteration, each controller updates its internal states as a linear combination of the previous states of all nodes in its in-neighborhood. In particular, at every time instant $k \geq 0$:

$$y_i[k+1] = \sum_{j \in \mathcal{N}_i^-} \frac{1}{D_j^+} y_j[k] \tag{2.1a}$$

$$z_i[k+1] = \sum_{j \in \mathcal{N}_i^-} \frac{1}{D_j^+} z_j[k], \tag{2.1b}$$

where D_j^+ is the out-degree of DGR j that belongs to the in-neighborhood of i ($j \in \mathcal{N}_i^-$) [3]. At every iteration, the local controller i also computes

$$\gamma_i[k] = \frac{y_i[k]}{z_i[k]} \tag{2.2}$$

assuming that $z[k] \neq 0, \ \forall\, k \geq 0$ [3].

It can be proved [6] that, for $y_i[k]$ and $z_i[k]$ being the result of (2.1) for some $y_i[0]$ and $z_i[0] > 0$, then:

$$\lim_{k \to \infty} \gamma_i[k] = \frac{\sum_{j=1}^{n} y_j[0]}{\sum_{j=1}^{n} z_j[0]} \ \ \forall i. \tag{2.3}$$

In other words, (2.2) converges to some constant value that is the same for every bus i, which is equal to the ratio of the summation of the initial internal states of every node. The number of iterations that are required for (2.2) to converge is dependent on the structure of communication network and on the out-degrees of the local controllers [3].

The consequence of this result is that, only through the exchange of information with their own neighborhood, the local controllers can obtain knowledge of the full network (the value of the ratio $\frac{\sum_{j=1}^{n} y_j[0]}{\sum_{j=1}^{n} z_j[0]}$). This conclusion suggests that a distributed architecture can indeed perform the same functions as a centralized approach. In the remainder, it will be proved that ratio-consensus can be employed as a primitive algorithm to distributively implement frequency regulation and optimal dispatch.

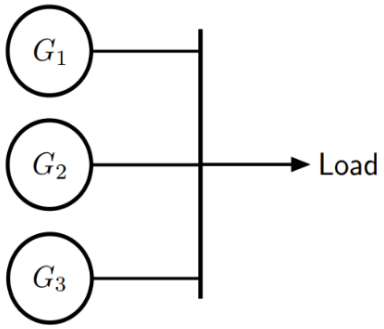

Figure 14. Power system with three generators and one load for Ratio-Consensus application example [4].

2.1.4. Ratio-Consensus – Motivating Application Example

Let us say that a 5 MW increase in demand ($\Delta P_D = 5\ MW$) must be collectively covered by three synchronous generators (DGR 1, DGR 2 and DGR 3) located nearby. A graphical depiction can be seen in Figure 14 (from [4]). Each DGR can generate between its respective maximum and minimum power outputs. In the case of DGR 1, we have that $P_1 \in [-3, 3]$; DGR 2 can supply $P_2 \in [-5, 5]$ and DGR 3 provides $P_3 \in [-2, 2]$. Let us assume further that the three generators were offline ($P_1 = P_2 = P_3 = 0\ MW$) at the moment of the load increase. Imagine a scenario in which the DGRs are participating in Ratio-Consensus algorithm, and their initial states are initialized as follows:

$$y_i[0] = \frac{\Delta P_D}{n}, \qquad z_i[0] = P_i^{max} - P_i^{min}, \qquad i = 1,2,3 = n$$

After several iterations, (2.2) will converge to:

$$\lim_{k\to\infty} \gamma_i[k] = \lim_{k\to\infty} \frac{y_i[k]}{z_i[k]} = \frac{\sum_{j=1}^{n} y_j[0]}{\sum_{j=1}^{n} z_j[0]} = \frac{\Delta P_D}{\sum_{j=1}^{n}(P_j^{max} - P_j^{min})} = \frac{1}{4} = \gamma, \qquad i = 1,2,3$$

Therefore, each generator could determine the power that it must supply in steady state by using this information. One possible approach to do this is the following:

$$\Delta P_1 = (P_1^{max} - P_1^{min}) \cdot \gamma = \frac{3}{2}\ MW$$

$$\Delta P_2 = (P_2^{max} - P_2^{min}) \cdot \gamma = \frac{5}{2}\ MW$$

$$\Delta P_3 = (P_3^{max} - P_3^{min}) \cdot \gamma = 1\ MW$$

from where one can easily check that:

$$\Delta P_1 + \Delta P_2 + \Delta P_3 = \frac{3}{2} + \frac{5}{2} + 1 = 5\ MW = \Delta P_D$$

By executing Ratio-Consensus, the DGR's local controllers have been able to obtain information of the full network (γ) to compute the fraction of ΔP_D they must cover. This example proves that Ratio-Consensus can be utilized to compute frequency regulation and, as it will be shown later, as a primitive algorithm for distributively implementing optimal dispatch.

2.1.5. Ratio-Consensus Practical Implementation – Robust Ratio-Consensus

In regards to practical implementation, the ratio-consensus algorithm happens to be insufficient, since it does not account for the loss of information packets when sent from one bus to another [3]. If the communication link between nodes i and j is unavailable at a time instant $k \geq 0$, the computation of (2.1) will be inaccurate and may imply the non-convergence of the algorithm. Therefore, a modified version of the ratio-consensus must be implemented in practice.

In the Robust Ratio-Consensus algorithm (see [7] for a complete description), the DGR's local controllers broadcast a cumulative sum of their internal states up to the current iteration k. In this way, in case of loss of information, the controllers can recover the lost packets at the following successful iteration.

To execute this version of the algorithm, each node i computes the same two internal states as before, i.e., $y_i[k]$ and $z_i[k]$ for every $k \geq 0$. Besides, each bus also maintains two additional values, $\mu_i[k]$ and $\sigma_i[k]$, which are accumulated sums of the internal states from $k = 0$. These are the broadcasted values to the out-neighbors of bus i to account for unreliable communication. Mathematically:

$$\mu_i[k+1] = \mu_i[k] + \frac{1}{D_i^+} y_i[k] = \sum_{t=0}^{k} \frac{1}{D_i^+} y_i[t] \tag{2.4a}$$

$$\sigma_i[k+1] = \sigma_i[k] + \frac{1}{D_i^+} z_i[k] = \sum_{t=0}^{k} \frac{1}{D_i^+} z_i[t] \tag{2.4b}$$

with $\mu_i[0] = 0$ and $\sigma_i[0] = 0$. Then, the internal states are updated as follows:

$$y_i[k+1] = \frac{1}{D_i^+} y_i[k] + \sum_{i \neq j;\ j \in \mathcal{N}_i^-} (v_{ij}[k+1] - v_{ij}[k]) \tag{2.4c}$$

$$z_i[k+1] = \frac{1}{D_i^+} z_i[k] + \sum_{i \neq j;\ j \in \mathcal{N}_i^-} (\tau_{ij}[k+1] - \tau_{ij}[k]) \tag{2.4d}$$

where $v_{ij}[k+1]$ and $\tau_{ij}[k+1]$ are auxiliar variables whose values depend on the successful transmission of information from bus j to bus i at iteration k, i.e.,

$$\begin{aligned} v_{ij}[k+1] &= \mu_j[k+1]\,, \qquad if\ (i,j) \in \mathcal{E}[k] \\ v_{ij}[k+1] &= v_{ij}[k]\,, \qquad if\ (i,j) \notin \mathcal{E}[k] \end{aligned} \tag{2.4e}$$

$$\tau_{ij}[k+1] = \sigma_j[k+1]\,, \qquad if\ (i,j) \in \mathcal{E}[k] \tag{2.4f}$$
$$\tau_{ij}[k+1] = \tau_{ij}[k]\,, \qquad if\ (i,j) \notin \mathcal{E}[k]$$

where the use of $\mathcal{E}[k] \subseteq \mathcal{E}$ denotes the corrected edge set at time instant $k \geq 0$, that accounts for the possible failure of communication links [3]-[7]. Finally, as it is shown in [7], the convergence of (2.2) as in (2.3), i.e., to the ratio of the summation of the initial internal states of every bus, identically applies to the robust version of ratio-consensus.

Robust Ratio-Consensus provides a solid basis for distributively implementing frequency regulation and optimal dispatch. However, as it has been demonstrated in the example in Section 2.1.4, it can be directly used to compute, e.g., the amount of power that a group of DGRs must supply to satisfy a certain increase in demand (and therefore, to perform frequency regulation). In the remainder, attention will be focused on applying Robust Ratio-Consensus algorithm to the third layer of the generation control structure for islanded ac microgrids: the optimal coordination of the network's distributed generation resources.

2.2. Optimal Coordination of Distributed Generation Resources

In this chapter, we address the distributed implementation of optimal dispatch, taking as starting point the algorithms developed in the previous section. It will be shown that this third layer of the generation control architecture can be approached as a convex optimization problem, in which we seek to minimize a sum of convex functions, i.e., the generators' cost functions, respecting a linear equality constraint (supply must match demand) and multiple linear inequality constraints due to the DGR's capacity limits [17]. To solve this optimization problem, a preliminary distributed primal-dual algorithm will be presented in first place. Then, we will tailor robust ratio-consensus to refine this algorithm and obtain a complete set of equations that can successfully synthesize optimal dispatch in a distributed fashion.

2.2.1. Convex Optimization Problem

As previously stated, the third layer of the generation control architecture is responsible for minimizing the total generation cost of the system while meeting the full electric power demand and respecting the capacity limits of the generating units. In the remainder, we assume that the individual generation cost of each DGR is a function of its power output. We assume further that these cost functions are quadratic -thus, twice differentiable, and strongly convex- of the form:

$$f_i(p_i) = a_i \cdot p_i^2 + b_i \cdot p_i + c_i \tag{2.5}$$

where $f_i(\cdot)$ denotes the generation cost associated to DGR i, $i = [1, 2, \dots, n]$; p_i is the power output; and a_i, b_i and c_i are constant parameters [5].

Therefore, optimal dispatch can be approached through a convex optimization problem, in which we intend to minimize a convex function -the sum of the DGRs' individual cost functions- subject to a linear equality constraint related to the total power supply being equal to the total demand (assuming the system to be lossless), and linear inequality constraints that account for the DGRs' capacity limits. Mathematically:

$$\underset{p \in R^n}{\text{Minimize}} \sum_{i=1}^{k} f_i(p_i) = \sum_{i=1}^{k} (a_i \cdot p_i^2 + b_i \cdot p_i + c_i) \tag{2.6a}$$

$$subject\ to:\ 1^T p = 1^T l \tag{2.6b}$$

$$p_{min} \leq p \leq p_{max} \tag{2.6c}$$

where $p = [p_1, p_2, \ldots, p_n]^T$; $l = [l_1, l, \ldots, l_n]^T$ denotes the power demanded at every bus $i = [1, 2, \ldots, n]$; $1^T \in R^n$ is the all-ones vector; and $p_{min} = [p_{min\_1}, p_{min\_2}, \ldots, p_{min\_n}]^T$ and $p_{max} = [p_{max\_1}, p_{max\_2}, \ldots, p_{max\_n}]^T$ denote the lower and upper generation limits of each DGR [5], [17]-[18].

2.2.2. Preliminary Distributed Primal-Dual Algorithm

By introducing the so-called Lagrange multiplier λ, associated with the power balance constraint, i.e., $1^T p = 1^T l$, we have the following primal-dual algorithm [2] to help solve (2.6):

$$p_i[k+1] = [\, p_i[k] - s f_i(p_i[k]) + s\xi\bar{\lambda}[\mathrm{k}] \,]_{p_{min}}^{p_{max}} \tag{2.7a}$$

$$\bar{\lambda}[\mathrm{k}+1] = \bar{\lambda}[\mathrm{k}] - s1^T(p[k] - l) \tag{2.7b}$$

where s is constant step-size, ξ is costant parameter, $p[k] = [p_1[k], p_2[k], \ldots, p_n[k]]^T$, and $[\cdot]_{p_{min}}^{p_{max}}$ denotes the projection of p onto the interval defined by its upper and lower limits: $p = p_{max}$ if $p > p_{max}$, $p = p_{min}$ if $p > p_{min}$, and $p = p$ if $p_{min} < p < p_{max}$. Finally, $\bar{\lambda}[\mathrm{k}]$ denotes an estimate of the Lagrange multiplier at time $k \geq 0$.

To develop a distributed version of (2.7), the controller at bus i must maintain a local estimate of $\bar{\lambda}[\mathrm{k}]$, i.e., $\lambda_i[k]$. In order to obtain $\lambda_i[k]$, each DGR i will also need an estimation of the total power imbalance $1^T(p[k] - l)$, which will be denoted as $y_i[k]$. This will be calculated by computing the average of bus i's power imbalance and the estimates of its neighbors:

$$y_i[k+1] = (1 - \sum_j a_{ij}[k]) y_i[k] + \sum_j a_{ij}[k]\, y_j[k] + \hat{n}(p_i[k+1] - p_i[k]) \tag{2.8}$$

where $\hat{n}$ is an estimate of the number of nodes (n) that each local controller has, and $a_{ij}[k] > \eta > 0$ if $(\mathrm{i}, \mathrm{j}) \in \mathcal{E}$, and $a_{ij}[k] = 0$ otherwise. η is chosen so that $1 - \sum_j a_{ij}[k] \geq \eta$ [18].

Now, by introducing the local estimate of the Lagrange multiplier $\lambda_i[k]$ and using (2.8), we can express (2.7) in a distributive fashion as:

$$p_i[k+1] = [\, p_i[k] - s f_i(p_i[k]) + s\xi\lambda_i[k] \,]_{p_{min}}^{p_{max}} \tag{2.9a}$$

$$\lambda_i[k+1] = (1 - \sum_j a_{ij}[k]) \lambda_i[k] + \sum_j a_{ij}[k]\, \lambda_j[k] - s y_i[k] \tag{2.9b}$$

$$y_i[k+1] = (1 - \sum_j a_{ij}[k]) y_i[k] + \sum_j a_{ij}[k]\, y_j[k] + \hat{n}(p_i[k+1] - p_i[k]) \tag{2.9c}$$

where $\lambda_i[0] = 0;\ y_i[0] = \hat{n}(p_i[0] - l_i)$, and $p_i[0]$ and l_i are given by the system [18].

Finally, by recalling the ratio-consensus algorithm described in Section 2.1.3 and utilizing it to estimate the total power imbalance $y_i[k]$ and the Lagrange multiplier $\lambda_i[k]$, (2.9) can be rewritten for directed communication networks as:

$$p_i[k+1] = [\, p_i[k] - sf_i(p_i[k]) + s\xi x_i[k] \,]_{p_{min}}^{p_{max}} \tag{2.10a}$$

$$\lambda_i[k+1] = \sum_{j \in \mathcal{N}_i^-[k]} \frac{\lambda_j[k] - sy_j[k]}{D_j^+[k]} \tag{2.10b}$$

$$v_i[k+1] = \sum_{j \in \mathcal{N}_i^-[k]} \frac{v_j[k]}{D_j^+[k]} \tag{2.10c}$$

$$x_i[k+1] = \frac{\lambda_i[k+1]}{v_i[k+1]} \tag{2.10d}$$

$$y_i[k+1] = \sum_{j \in \mathcal{N}_i^-[k]} \frac{y_j[k]}{D_j^+[k]} + \hat{n}(p_i[k+1] - p_i[k]) \tag{2.10e}$$

where $\lambda_i[0] = 0;\ v_i[0] = 1;\ x_i[0] = 0;\ y_i[0] = \hat{n}(p_i[0] - l_i)$; and $p_i[0]$ and l_i are given by the system [18]. It is required to distinguish between the instantaneous out-degree of node i at time k ($D_i^+[k]$) and its nominal out-degree (d_i^+). This differentiation lies on the possible loss of information packages when sent from bus i, what reduces its number of "real" neighbors at that iteration. In the remainder, it will be assumed that only the nominal out-degrees are known by the local controllers [18], which means that a robust version of the Distributed Primal-Dual algorithm presented in (2.10) must be developed.

2.2.3. Robust Distributed Primal-Dual Algorithm

To account for the lack of knowledge of the instantaneous out-degrees, due to unreliable communication links, a variation of the Robust Ratio-Consensus must be implemented (the so-called Running-Sum Ratio-Consensus, the reader is referred to [9] for a more detailed depiction of the algorithm). Recall the expressions in (2.4a – 2.4d), as well as the convergence of (2.2) as in (2.3), but now the auxiliar variables $v_{ij}[k]$ and $\tau_{ij}[k]$, dependent on the successful sending of information from local controller j to i, will be updated in a slightly different manner [18]:

$$v_{ij}[k+1] = (1-\gamma)v_{ij}[k] + \gamma \sum_{t=0}^{k} \frac{1}{d_j^+} y_j[t], \qquad if\ j \in \mathcal{N}_i^-[k], j \neq i \tag{2.11a}$$

$$v_{ij}[k+1] = v_{ij}[k] + \frac{1}{d_j^+} y_j[k], \qquad if\ j = i$$

$$v_{ij}[k+1] = v_{ij}[k], \qquad otherwise$$

$$\tau_{ij}[k+1] = (1-\gamma)\tau_{ij}[k] + \gamma \sum_{t=0}^{k} \frac{1}{d_j^+} z_j[t], \qquad if\ j \in \mathcal{N}_i^-[\mathrm{k}], j \neq i \tag{2.11b}$$

$$\tau_{ij}[k+1] = \tau_{ij}[k] + \frac{1}{d_j^+} z_j[k], \qquad if\ j = i$$

$$\tau_{ij}[k+1] = \tau_{ij}[k], \qquad otherwise$$

where $0 < \gamma < 1$ is constant parameter.

By using the Robust Ratio-Consensus in (2.4a – 2.4d) with $v_{ij}[k]$ and $\tau_{ij}[k]$ updated as in (2.11), we can formulate a robust extension of the Distributed Primal-Dual algorithm depicted in (2.10) [18]:

$$p_i[k+1] = [\, p_i[k] - sf_i(p_i[k]) + s\xi x_i[k] \,]_{p_{min}}^{p_{max}} \tag{2.12a}$$

$$\lambda_i[k+1] = \sum_{j \in \mathcal{N}_i^-[\mathrm{k}]} (\lambda_{ij}[k+1] - \lambda_{ij}[k] - sy_{ij}[k+1] + sy_{ij}[k]) \tag{2.12b}$$

$$v_i[k+1] = \sum_{j \in \mathcal{N}_i^-[\mathrm{k}]} (v_{ij}[k+1] - v_{ij}[k]) \tag{2.12c}$$

$$x_i[k+1] = \frac{\lambda_i[k+1]}{v_i[k+1]} \tag{2.12d}$$

$$y_i[k+1] = \sum_{j \in \mathcal{N}_i^-[\mathrm{k}]} (y_{ij}[k+1] - y_{ij}[k]) + \hat{n}(p_i[k+1] - \, p_i[k]) \tag{2.12e}$$

with $\lambda_{ij}[k]$, $v_{ij}[k]$ and $y_{ij}[k]$ updated by tailoring the expressions in (2.11):

$$\lambda_{ij}[k+1] = (1-\gamma)\lambda_{ij}[k] + \gamma \sum_{t=0}^{k} \frac{1}{d_j^+} \lambda_j[t], \qquad if\ j \in \mathcal{N}_i^-[k], j \neq i \tag{2.12f}$$

$$\lambda_{ij}[k+1] = \lambda_{ij}[k] + \frac{1}{d_j^+} \lambda_j[k], \qquad if\ j = i$$

$$\lambda_{ij}[k+1] = \lambda_{ij}[k], \qquad otherwise$$

$$v_{ij}[k+1] = (1-\gamma)v_{ij}[k] + \gamma \sum_{t=0}^{k} \frac{1}{d_j^+} v_j[t], \qquad if\ j \in \mathcal{N}_i^-[k], j \neq i \tag{2.12g}$$

$$v_{ij}[k+1] = v_{ij}[k] + \frac{1}{d_j^+} v_j[k], \qquad if\ j = i$$

$$v_{ij}[k+1] = v_{ij}[k], \qquad otherwise$$

$$y_{ij}[k+1] = (1-\gamma)y_{ij}[k] + \gamma \sum_{t=0}^{k} \frac{1}{d_j^+} y_j[t], \qquad if\ j \in \mathcal{N}_i^-[\mathrm{k}], j \neq i \tag{2.12h}$$

$$y_{ij}[k+1] = y_{ij}[k] + \frac{1}{d_j^+} y_j[k], \qquad if\ j = i$$

$$y_{ij}[k+1] = y_{ij}[k], \qquad otherwise$$

where $0 < \gamma < 1$ and the iterative process is initialized with $\lambda_i[0] = 0, v_i[0] = 1, x_i[0] = 0, y_i[0] = \hat{n}(p_i[0] - l_i)$; and $p_i[0]$ and l_i are given by the system. The choice of the parameters s and ξ will not be developed in this paper. Based on the results in [18], reasonable values are $0 < s << 1$ and $0 < \xi < 1$.

Deeper theoretical analysis of the convergence of this algorithm will not be addressed herein; the reader is referred to [18] for further information. However, in this project, Robust Distributed Primal-Dual algorithm will be programmed and simulated using the *Typhoon HIL Control Center* software in first place, and then synthesized onto several industrial-grade control hardware devices (HIL's 4th Generation Typhoon HIL404) to showcase its performance in the implementation of optimal dispatch.

3. Description of Research Results

3.1. Distributed Control Algorithms

Throughout this section, we address the first stage of the distributed generation control implementation, i.e., the development and programming of the distributed algorithms. First, a description is included of the cyber layer of the microgrid where the algorithms are to be applied. Then, we show the modeling, programming and numerical simulations of the three algorithms described above using the *Typhoon HIL* toolchain.

3.1.1. Communication Graph and Software Utilized to Implement the Distributed Algorithms

The developed algorithms will be synthesized onto a four-DGRs, laboratory-grade testbed. Each of these generators will be equipped with an Arduino microcontroller, with which it will be able to interact. In turn, these local controllers will send and receive information, to and from their out- and in-neighborhood respectively, through the four-node directed communication network depicted in Figure 15 (from [3]).

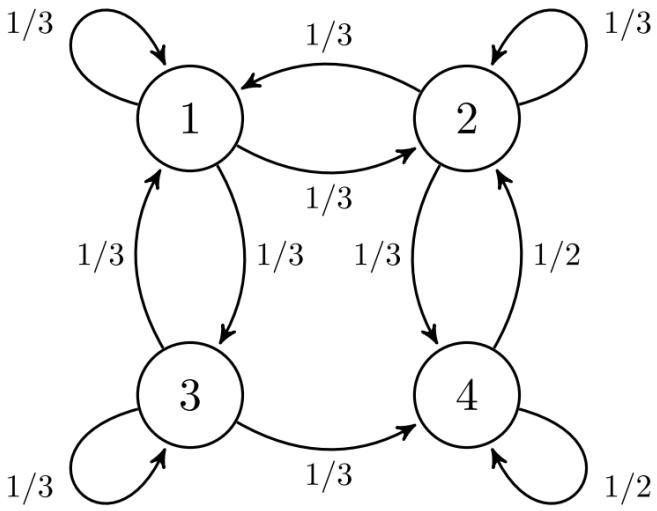

Figure 15. Four-node communication network [3].

This schematic can be understood as a graphical representation of the cyber-layer of the microgrid, which results from tailoring the left-hand side of Figure 2 (see Section 2.1.2) to the system proposed herein. In view of Figure 15, we can derive:

$$\mathcal{N}_1^- = \mathcal{N}_1^+ = \{1,2,3\};\ D_1^+ := |\mathcal{N}_1^+| = 3 \tag{3.1a}$$

$$\mathcal{N}_2^- = \mathcal{N}_2^+ = \{1,2,4\};\ D_2^+ := |\mathcal{N}_2^+| = 3 \tag{3.1b}$$

$$\mathcal{N}_3^- = \{1,3\};\ \ \mathcal{N}_3^+ = \{1,3,4\};\ D_3^+ := |\mathcal{N}_3^+| = 3 \tag{3.1c}$$

$$\mathcal{N}_4^- = \{2,3,4\};\ \ \mathcal{N}_4^+ = \{2,4\};\ D_4^+ := |\mathcal{N}_4^+| = 2 \tag{3.1d}$$

With this information we can now address the implementation of the distributed generation control algorithms described earlier, i.e., Ratio-Consensus, Robust Ratio-Consensus and Robust Distributed Primal-Dual (optimal dispatch).

The initial synthesis and numerical simulations of the algorithms, as well as the modeling of the network described above, has been performed by using the Model-Based System Engineering (MBSE) toolchain *Typhoon HIL Control Center*, designed for high performance and long-term numerical stability [13]. In particular, the drawing of the models and the programming and compiling of the algorithms have been conducted in the *Schematic Editor* tool of this software, whereas *HIL SCADA* has allowed the testing, visualization of the results and convergence analysis.

3.1.2. Ratio-Consensus Algorithm

The preliminary Ratio-Consensus lays the groundwork, both theoretically and in terms of programming, for developing the rest of the work proposed herein, so it is the first to be addressed.

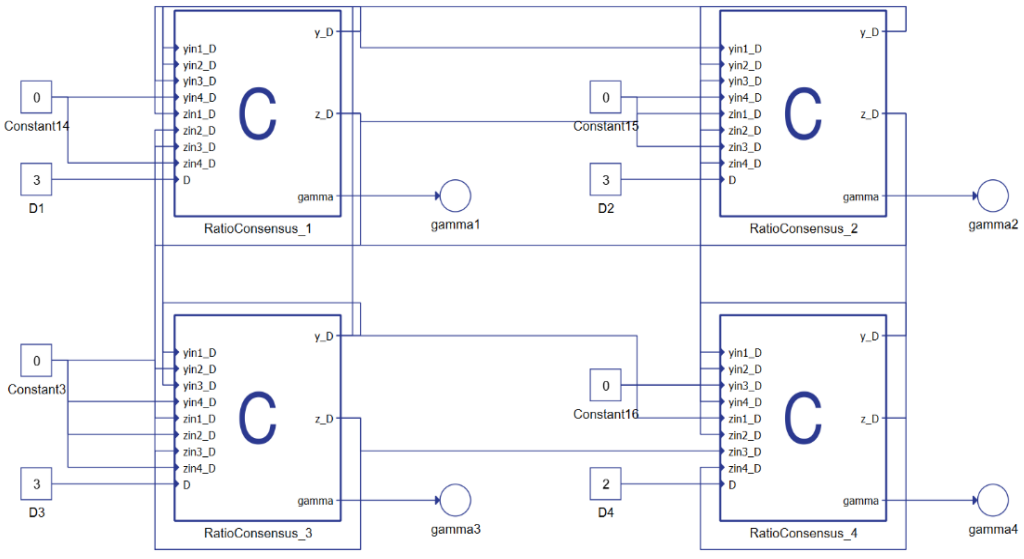

Figure 16. Ratio-Consensus algorithm – Typhoon HIL Schematic Editor.

A) Model

Figure 16 shows the model used to implement the basic version of Ratio-Consensus algorithm in the *Typhoon HIL Schematic Editor* software, where each C-function block plays the role of the local controller located at a DGR (where the code is synthesized) and the lines interconnecting them represent the communication network depicted in Figure 15 and allow the sending and receiving information between nodes. The reader is referred to Appendix A for the complete code of the algorithm.

It is worth noting that, at every iteration $\mathrm{k} \geq 0$, each node j broadcasts the values $\frac{y_j[\mathrm{k}]}{D_\mathrm{j}^+}$ and $\frac{\mathrm{z}_j[\mathrm{k}]}{D_\mathrm{j}^+}$ (denoted by y_D and z_D respectively in the model) to its out-neighborhood. This prevents the local controllers from requiring previous knowledge of the out-degrees of their neighbors (D_j^+), so that they can update their states by simply adding up the values they receive ($yin1\_D = \frac{y_1[k]}{D_1^+}, \ldots, yin4\_D = \frac{y_4[k]}{D_4^+}$; $zin1\_D = \frac{z_1[k]}{D_1^+}, \ldots, zin4\_D = \frac{z_4[k]}{D_4^+}$). Note also that each node's own out-degree is passed as an input to the corresponding C-function, since it is assumed to be known by the local controllers.

Moreover, if there is not a communication link between nodes i and j, then the input i of node j will be zero (see node 1, input $yin4\_D$). In this regard, the communication network defining the in- and out-neighborhood of every bus as described in (3.1) has been manually fixed by directly interconnecting the corresponding inputs and outputs. However, in a more realistic setup, this could be dynamically determined in base of, e.g., the strength of the signals and, therefore, the proximity of two nodes [3].

Finally, at every iteration each local controller computes $\gamma[k]$ as in (2.2), which has been referred to as *gamma* in the model. This output is connected to a probe to visualize its evolution.

B) Simulation

The algorithm is initialized with the following values for the internal states [3]:

$$y_1[0] = 0.35;\ y_2[0] = 0.5;\ y_3[0] = -0.15;\ y_4[0] = -0.1 \tag{3.2a}$$

$$z_1[0] = 0.15;\ z_2[0] = 0.15;\ z_3[0] = 0.25;\ z_4[0] = 0.15 \tag{3.2b}$$

Inspection of (2.3) reveals that $\gamma_i[k]$ ($gamma$ in the model) of every node must converge to [6]:

$$\lim_{k\to\infty} \gamma_i[k] = \frac{\sum_{j=1}^{n} y_j[0]}{\sum_{j=1}^{n} z_j[0]} = \frac{0.6}{0.7} = 0.8571, \qquad i = 1,2,3,4 \tag{3.3}$$

By compiling the model and plotting the information collected by the probes in *HIL SCADA* (see Figure 17 below), we can observe that the output signals of the four local controllers ($\gamma_i[k], i = 1,2,3,4$) converge to the expected value calculated in (3.3) after about eight iterations (eight divisions, 1.0 sec/div, 1 sec. execution rate).

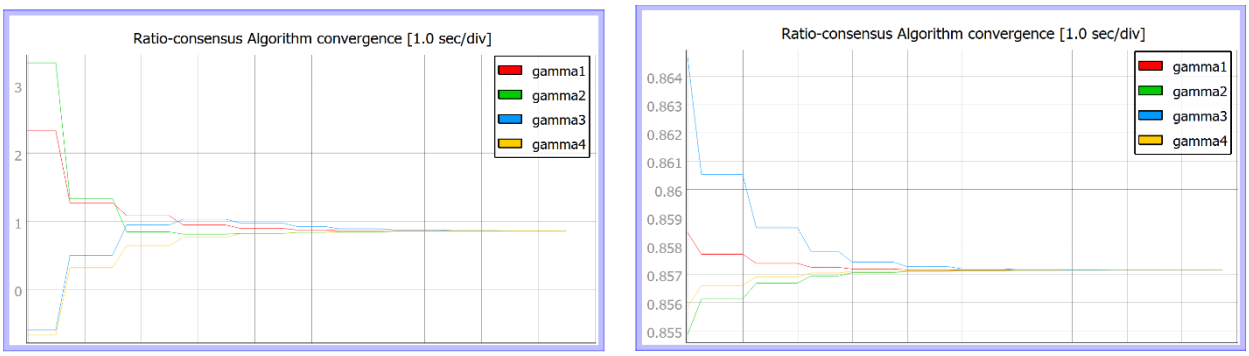

Figure 17. Ratio-Consensus algorithm successful convergence – HIL SCADA

3.1.3. Robust Ratio-Consensus Algorithm

As explained above, Ratio-Consensus is insufficient when it comes to its practical implementation. To mitigate the effects of lost communication packages when sent from one bus to another, a robust version of this algorithm must be developed.

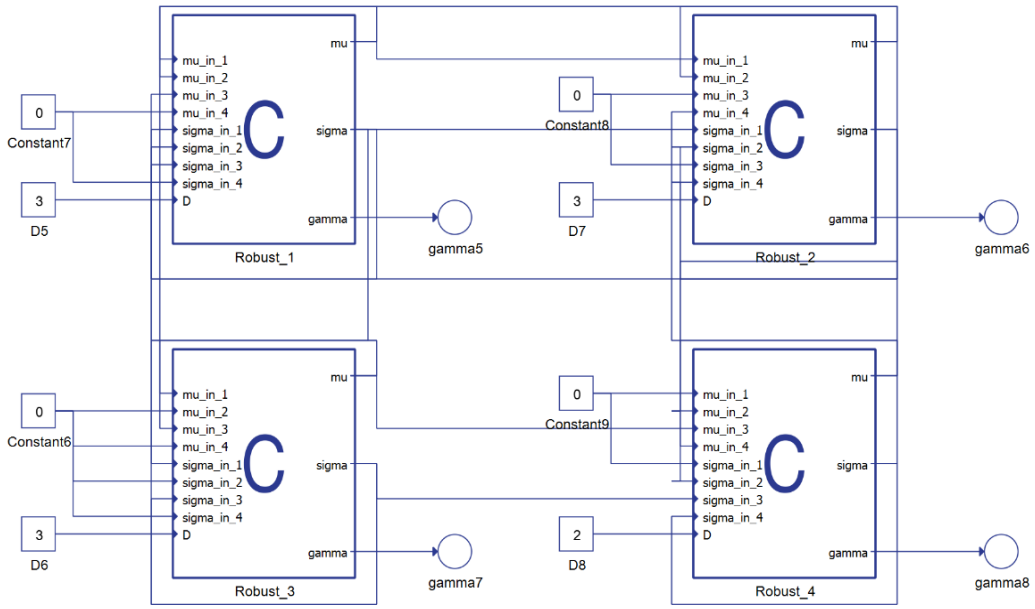

Figure 18. Robust Ratio-Consensus algorithm – Typhoon HIL Schematic Editor.

A) Model

The model developed to synthesize Robust Ratio-Consensus is depicted in Figure 18. Note that the values broadcasted by the local controllers are now the running sums $\mu_i[k] = \sum_{t=0}^{k} \frac{1}{D_i^+} y_i[t]$ and $\sigma_i[k] = \sum_{t=0}^{k} \frac{1}{D_i^+} z_i[t]$ (mu and $sigma$) defined in (2.4a – 2.4b), which are then used by each node in its out-neighborhood to update its internal states $y_i[\mathrm{k}]$ and $z_\mathrm{i}[k]$ as in (2.4c – 2.4d).

Inspection of (2.4c – 2.4f) indicates that the controllers are required to maintain, for the next iteration, the most recent successfully received values $v_{ij}[k-1]$ and $\tau_{ij}[k-1]$ from each node in its in-neighborhood [3]. This has been achieved by creating two arrays $(v_{ij\_new}[i], \tau_{ij\_new}[i],\ i = 1,2,3,4)$ containing all the μ_i and σ_i received from every bus ($mu\_in\_i, sigma\_in\_i,\ i = 1,2,3,4$, obviously some elements will be zero); and other two dummy arrays $(v_{ij}[i], \tau_{ij}[i],\ i = 1,2,3,4)$ saving the previous successfully received values. Clearly, at the end of every iteration, $v_{ij}[i]$ and $\tau_{ij}[i]$ are set equal to $v_{ij\_new}[i]$ and $\tau_{ij\_new}[i]$, so that the new received values can be stored in these latter two.

Note also that the summations in (2.4c – 2.4d) involve that each local controller must know its node number (from 1 to 4), i.e., at which node it is located. This have been set as a parameter in the initialization of each C-function in the model. Finally, the local controllers compute the ratio of their internal states ($gamma$) normally as in (2.2), value that we are monitoring here with a probe as in the Ratio-Consensus model. The reader is referred to Appendix A for the complete code of the algorithm.

B) Simulation

The internal states of each node are initialized to the same values included in (3.2) for the basic Ratio-Consensus [3]. But now, in addition, we need to set the running sums $\mu_i[0]$ and $\sigma_i[0]$ to be zero at the beginning of the iterative process.

Again, $\gamma_i[k] = \frac{y_i[k]}{z_i[k]}$, $\mathrm{i} = 1,2,3,4$, must converge [6] to $\frac{\sum_{j=1}^{n} y_j[0]}{\sum_{j=1}^{n} z_j[0]} = \frac{0.6}{0.7} = 0.8571$ as in (3.3).

Figure 19 illustrates the evolution of $\gamma_i[k]$ for every node. We see a very similar behavior to the observed for Ratio-Consensus algorithm, with a slightly faster convergence after six iterations (six divisions, 1.0 sec/div, 1 sec. execution rate) to the value indicated above.

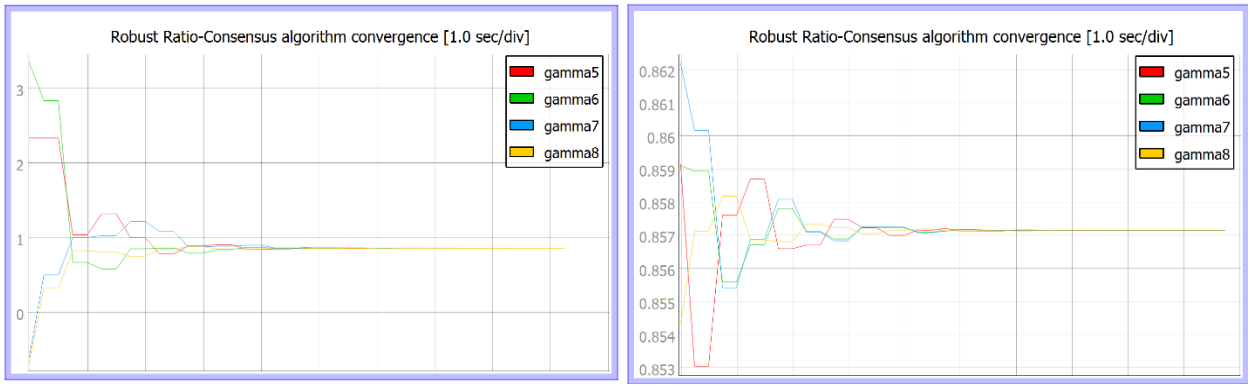

Figure 19. Ratio-Consensus algorithm successful convergence – HIL SCADA

3.1.4. Robust Distributed Primal-Dual Algorithm

By utilizing the Robust Ratio-Consensus as in (2.4a – 2.4d) and (2.11), we develop the algorithm described in (2.12). Robust Distributed Primal-Dual algorithm will be proved capable of solving the optimal dispatch problem in an AC microgrid in a distributed fashion.

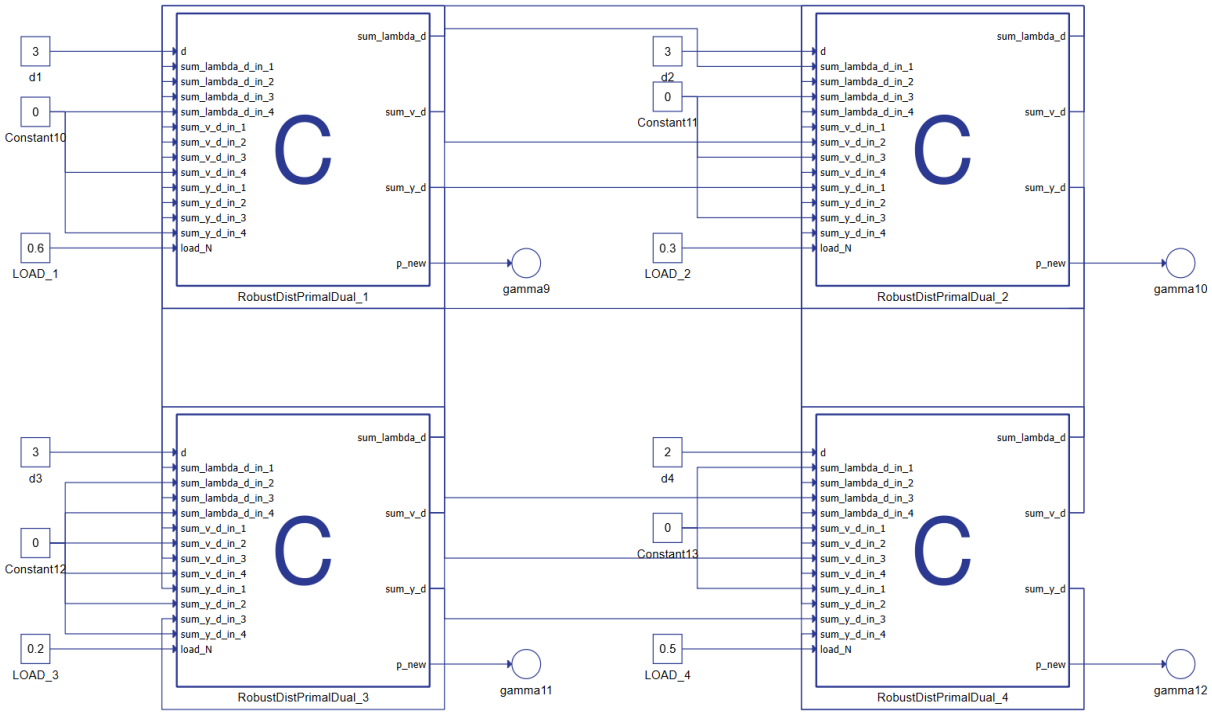

Figure 20. Robust Ratio-Consensus algorithm – Typhoon HIL Schematic Editor.

A) Model

Figure 20 shows the model utilized to implement the algorithm. As with the robust version of Ratio-Consensus, we let each local controller j broadcast the accumulated sums $\sum_{t=0}^{k} \frac{\lambda_j[t]}{d_j^+}$, $\sum_{t=0}^{k} \frac{v_j[t]}{d_j^+}$ and $\sum_{t=0}^{k} \frac{y_j[t]}{d_j^+}$ $(sum\_lambda\_d,\ sum\_v\_d,\ sum\_y\_d)$, which will be used by receiving node i to update the variables $\lambda_{ij}[k]$, $v_{ij}[k]$ and $y_{ij}[k]$.

Again, one can check that each agent is required to store the latest $\lambda_{ij}[k]$, $v_{ij}[k]$ and $y_{ij}[k]$, in order to update these values (i.e., to calculate $\lambda_{ij}[k+1]$, $v_{ij}[k+1]$ and $y_{ij}[k+1]$) and to compute the summations in (2.12b – 2.12c) and (2.12e). Therefore, as explained in the previous section, three arrays $\lambda_{ij\_new}[i], v_{ij\_new}[i], y_{ij\_new}[i],\ i = 1,2,3,4$, have been created to store the most recent calculated values, together with three dummy arrays to save the ones of the previous iteration ($\lambda_{ij}[i], v_{ij}[i], y_{ij}[i]$, $i = 1,2,3,4$). Besides, inspection of (2.12e) reveals that also the most recent value of the DGRs' output power must be stored, so two different variables p_{new} (the actual output, monitored by a probe) and p have been used in each node. At the end of each iteration, $\lambda_{ij}[i] = \lambda_{ij\_new}[i]$, $v_{ij}[i] = v_{ij\_new}[i]$, $y_{ij}[i] = y_{ij\_new}[i]$ and $p = p_{new}$.

This time, in addition to its node number and out-degree, each local controller requires knowledge of its DGR's maximum and minimum power outputs and cost function (parameters a, b and c), the load demand at that bus, and the estimation of the number of nodes. In this case, the load has been set as an input (to allow for demand variations), whereas the other values (*a priori* constant for each node), as well as the parameters $s, \xi,$ and γ, have been included in the initialization of each C-function block. See Appendix A for a complete depiction of the algorithm.

B) Simulation

The iterative process is initialized with:

$$p_i[0] = 0, \quad \lambda_i[0] = 0, \quad v_i[0] = 1, \quad x_i[0] = 0, \quad y_i[0] = \hat{n} \cdot (p_i - load_i), \qquad (3.4)$$
$$\lambda_{ij}[0] = v_{ij}[0] = y_{ij}[0] = 0,\ \ i = 1,2,3,4$$

where the estimation of the number of nodes has been set to the real number of nodes ($\hat{n} = 4$), and the load is passed as an input at every bus in the system as shown in Figure 20.

The parameter values have been chosen as in the numerical simulations performed in [18]:

$$s = 0.02, \qquad \xi = 0.02, \qquad \gamma = 0.9 \tag{3.5}$$

The load demands at every bus have been picked randomly, to make a total of 1.6 pu (per unit magnitude) as can be checked in Figure 20. The generation capacity limits of the DGRs have been also arbitrarily selected as:

$$p_{min,i} = -1\ pu, \qquad p_{max,i} = 1\ pu, \qquad i = 1, 2, 3, 4 \tag{3.6}$$

Finally, the DGR's cost functions have been modified to perform different simulations. As in [18], we choose $b_i = c_i = 0,\ i = 1, 2, 3, 4$, so that:

$$f_i(p_i) = a_i \cdot p_i^2, \qquad i = 1, 2, 3, 4 \tag{3.7}$$

where a_i will vary to test the performance of the algorithm in different scenarios.

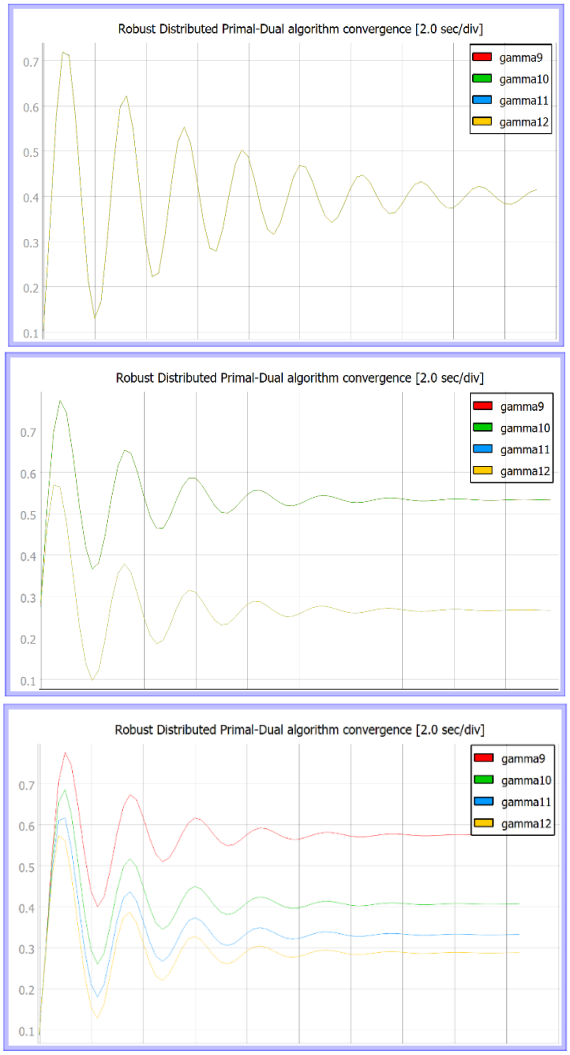

Figure 21. Robust Distributed Primal-Dual algorithm successful convergence – HIL SCADA.

Figure 21 (top) shows the case when $a_i = 0.1,\ i = 1, 2, 3, 4$, i.e., the four generators have identical cost functions. The result is that the power outputs sinusoidally converge to 0.4 pu, matching the total demand of 1.6 pu. Then, we alter the DGRs' cost functions so that $a_1 = a_2 = 0.1,\ a_3 = a_4 = 0.4$ -Figure 21 (middle)-. Although the curves overlap, it is easy to see that the generated power by generators 1 and 2 (*gamma9* and *gamma10*) is greater that the power supplied by 3 and 4 (*gamma11* and *gamma12*).

Finally, Figure 21 (bottom) considers the situation in which all the DGRs have different cost-efficiency: $a_1 = 0.1, a_2 = 0.2, a_3 = 0.3, a_4 = 0.4$. As before, the power produced by each of them is inversely correlated to the cost of its power output. One can easily check that the steady-state power supplies add up to meet the total demand of 1.6 pu. It is worth mentioning that, for this algorithm to converge (after about 14 seconds), the execution rate had to be decreased to 0.005 sec., which is consistent with its increased computational complexity.

With these computer simulations we intend to illustrate that Robust Distributed Primal-Dual algorithm, largely based on those developed earlier, can be used to implement optimal dispatch in a distributed fashion. As explained in Section 1.3, these algorithms are meant to be synthesized onto four industrial-grade control hardware devices (HIL's 4th Generation Typhoon HIL404) in order to perform real-time simulations of the distributed generation control architecture on a laboratory-grade four-DGRs microgrid (not addressed in this paper). In the remainder, we describe the communication protocols that allow not only the HIL404 devices (in practice, the DGRs) to exchange information with their respective local controllers, but also these to interact with one another.

3.2. Laboratory Testbed

This section will briefly describe the hardware components that have been used as the physical and cyber layers of the microgrid testbed. Then, we develop the communication protocols that will enable the wireless exchange of information among nodes and, therefore, that will allow for the implementation of the distributed control algorithms depicted above.

3.2.1. Physical and Cyber Layer Hardware

A) Physical Layer

The proposed distributed generation control architecture will be implemented on a laboratory-grade microgrid with four generation nodes. The physical layer of this microgrid will be comprised of four HIL's 4th Generation Typhoon HIL404. Customized to the HIL Control Center software and tailored for multiple interface possibilities including USB 3.0, Ethernet and mainly CAN [13], these devices constitute a very suitable option for the work presented herein.

Designed for automotive drives, Typhoon HIL404 features the most detailed inverter and electric motors models, broad compatibility, and several connectivity options. It includes an up-to-four-cores processor, 16-bit Analog I/O resolution on all its 16 channels with ±10V voltage range, and 32 Digital I/O channels with 3.5 ns GDS oversampling resolution. Time-steps down to 200ns, real-time emulations and communication with other devices, and parallel connection of up to four HIL404 (particularly useful for their integration in a microgrid) complete the basic specifications of these devices [13].

B) Cyber Layer

The cyber communication network of the microgrid has been built around the open-source electronic prototyping platform Arduino. Each DGR is equipped with an Arduino Due microcontroller board, that has been connected to a CAN-BUS Shield V2.0 to allow for CAN Bus communication. In turn, a MaxStream XB24-DMCIT-250 rev B XBee module embedded in the Arduino shield will enable the wireless exchange of information between nodes.

The Arduino Due board is based on the Atmel SAM3X8E ARM Cortex-M3 CPU, a 32-bit microcontroller with an 84 MHz clock, 54 digital I/O pins with a maximum voltage of 3.3 V, 12 analog inputs, and four Universal Asynchronous Receiver/Transmitter serial ports (UARTs) [1]. The CAN-BUS Shield acts as an interface between the Arduino microcontrollers and the XBees. It features MCP2515 CAN Bus controller

with Serial Peripheral Interface speed up to 10 MHz and MCP2551 CAN transceiver to provide CAN Bus connectivity to the board [11]. Finally, the Series 1 – chip antenna XBee modules serve as embedded RF transceivers that allow the local controllers to wirelessly interact with each other. They operate at 2.4 GHz frequency, and provide 3.3 V of rated voltage, 300 ft range, and 250 kbps of data rate [12].

In the following sections, we describe the communication protocols developed to allow for the exchange of information between nodes through the cyber layer of the microgrid. Once this exchange of information is proved successful, we will be in a position to distributively implement optimal dispatch to the system here proposed.

3.2.2. CAN Bus Communication

Once the distributed control algorithms have been synthesized onto the Typhoon HIL404 devices, the bidirectional exchange of information between these -the DGRs- and the local controllers -Arduino boards with embedded XBee modules- has been addressed via the Controller Area Network (CAN or CAN Bus) protocol.

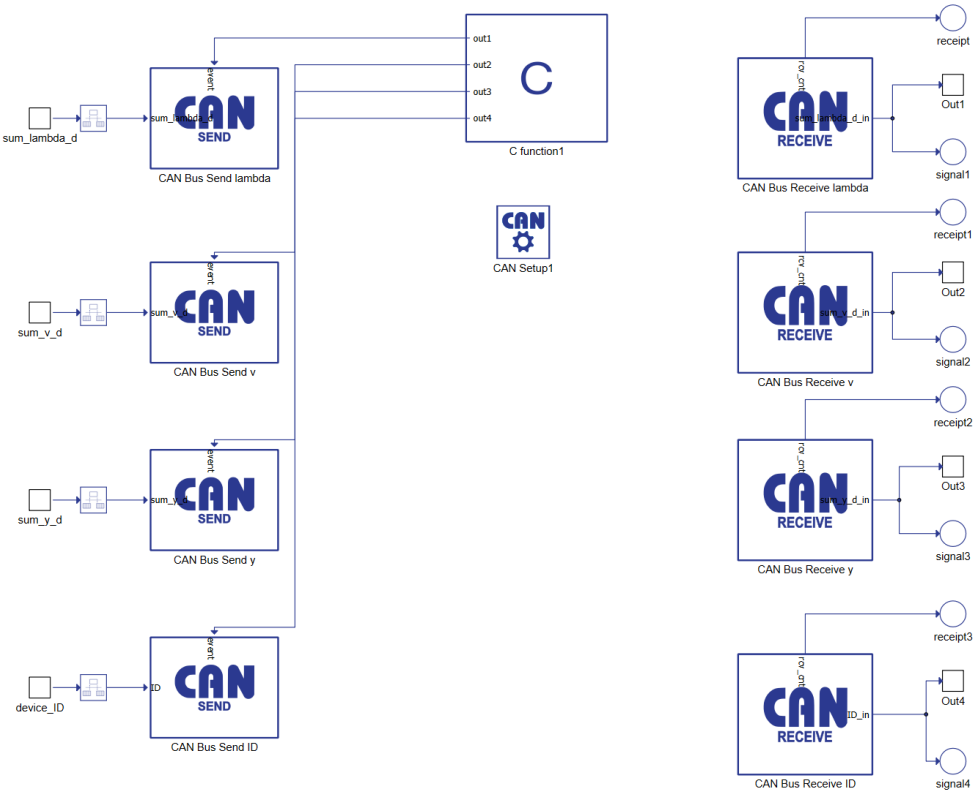

Figure 22. CAN Bus Communication – Typhoon HIL Schematic Editor.

In order to establish this communication, it was necessary to develop an Arduino script to set the SPI CS pin, fix the baud rate of the CAN Bus system, regulate the incoming and outcoming information (e.g., binary to float conversion and *vice versa*) and display the received values. This code can be found in Appendix B. In addition, a Typhoon HIL model has been created to implement this communication protocol. Figure 22 illustrates the designed CAN Bus communication model in the Typhoon HIL Schematic Editor toolchain.

Exactly one CAN Setup component must exist in the model [13], where the HIL device ID (0 by default), baud rates and execution rates are specified. Note also that we are using four CAN Bus Send and four CAN Bus Receive components. This makes sense, since -recalling Robust Distributed Primal-Dual algorithm- each DGR has to broadcast three different values, i.e., $\sum_{t=0}^{k} \frac{\lambda_j[t]}{d_j^+}$, $\sum_{t=0}^{k} \frac{v_j[t]}{d_j^+}$, $\sum_{t=0}^{k} \frac{y_j[t]}{d_j^+}$ ($sum\_lambda\_d, sum\_v\_d, sum\_y\_d$), plus an identifier of the node at which it is located. Similarly, four signals are to be received by the corresponding CAN Bus Receive: the three broadcasted values (received as $sum\_lambda\_d\_in$, $sum\_v\_d\_in$, $sum\_y\_d\_in$), and the identifier of the device they are coming from. Figure 23 shows dialog window of a CAN Bus Send a Receive component.

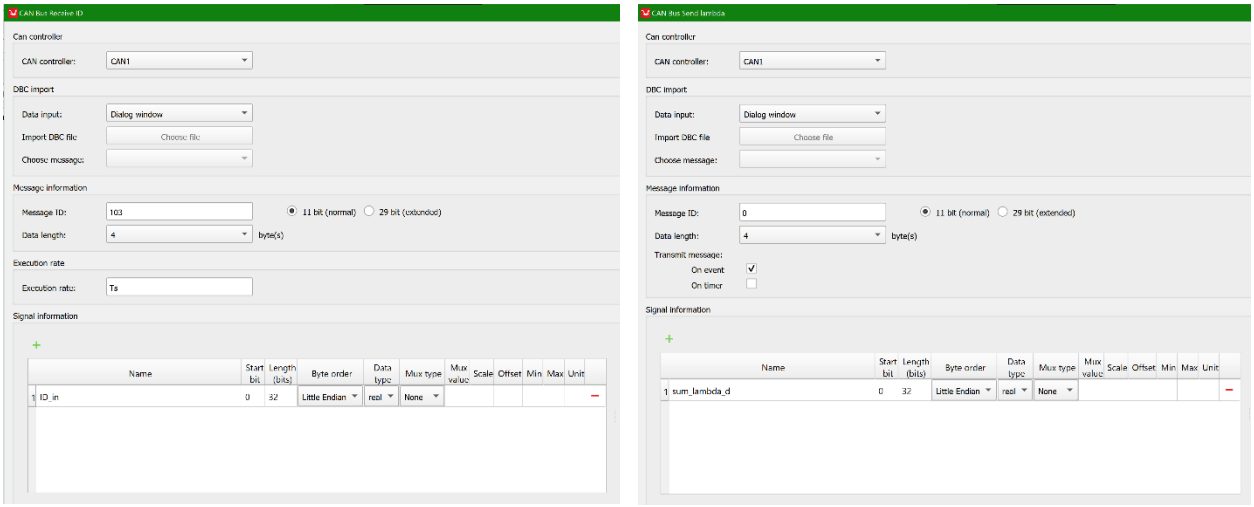

Figure 23. CAN Bus Send and CAN Bus Receive dialog window – Typhoon HIL Schematic Editor.

Each HIL404 device has two separate CAN controllers (CAN1 and CAN2) that can be used to send and receive information. However, the amount of information managed has only required the use of CAN1 in this case. It is worth noting that different message IDs (any number in the range $0 – 2^N-1$, where N is either 11 or 29 depending on the chosen ID type) must be used for each CAN Bus component. Although the IDs could be repeated between a Receive and a Send block, they have been chosen different as a preventive measure (0 – 3 for the sent messages and 100 – 103 for the received) [13]. Since the minimum length for

signed float variables is 32 bits and there is no need for working with very large quantities or with many decimal places, the message data length has been chosen to be 4 bytes. The execution rate for the CAN Bus Receive components has been set to a constant parameter (Ts) defined in the model initialization scrips as 0.1 seconds. Testing revealed that multiple signals being sent at the same time led to loss of information. Therefore, we set the message transmission rate of the CAN Bus Send blocks to happen *On event*, triggered by a C-function that aims to avoid this situation. Finally, multiple signals could be sent (received) in a single message, i.e., by a single CAN Bus Send (Receive) component up to 64 bits (see the signal information window at the bottom of Figure 23). Nevertheless, we needed each of them to have different IDs for the local controller to be able to recognize them separately and assign them to the appropriate variable. Thus, it was decided to send and receive each signal within a different message, i.e., a different component.

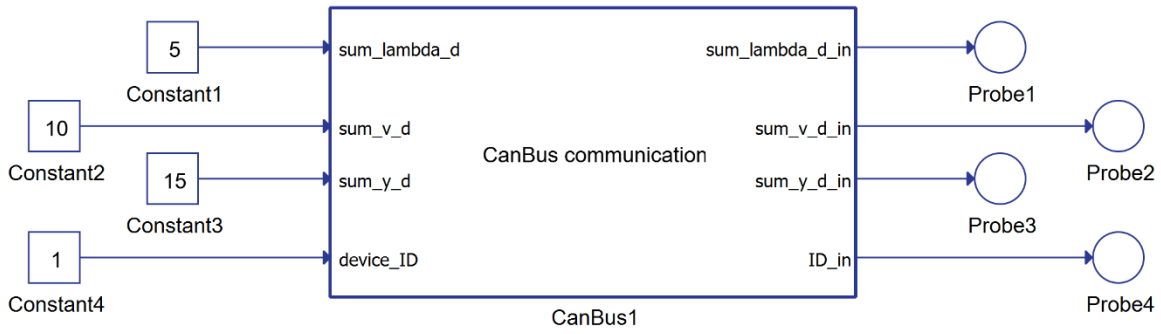

Figure 25. Model for CAN Bus communication test – Typhoon HIL Schematic Editor.

The schematic in Figure 22 has been included into a single subsystem which, in turn, has been defined as Typhoon Library to simplify its manipulation, edition and integration in other models requiring CAN Bus communication protocol. This library block is illustrated in Figure 24 under the name of *CanBus communication*, together with the model that will be used to test this communication protocol.

Note that, instead of running the full optimal dispatch algorithm, four constant inputs (values 5, 10, 15 and 1) have been connected to the CAN Bus subsystem. These will be sent from the HIL DGR to the Arduino-XBee local controller. In turn, the Arduino will read and process this information, and will also send four other variables from the microcontroller to the HIL404. These variables are initialized to values 1, 2, 3 and 4, and will increase by 0.1 in each iteration. They will be received by the HIL device and monitored using the probes connected to the outputs of the Typhoon library. The results of this test are depicted in Figure 25.

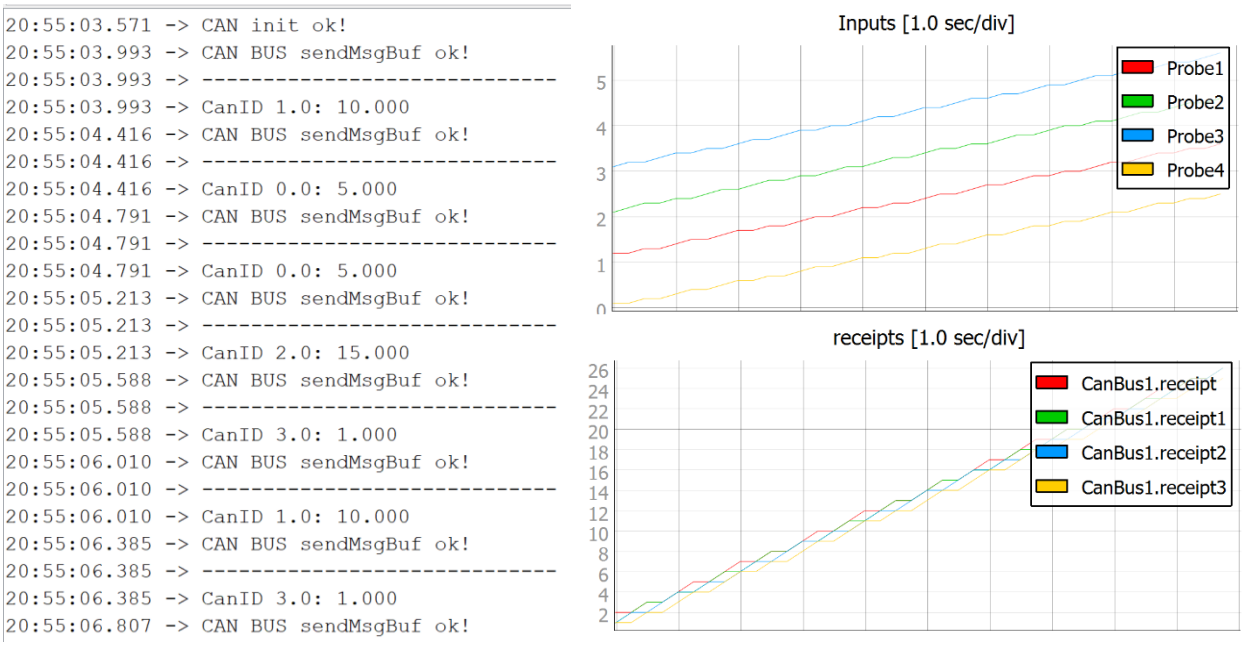

Figure 26. CAN Bus communication test results – Arduino Serial Port Monitor (left), HIL SCADA (right).

On the left-hand side, the Serial Port Monitor of the Arduino microcontroller gathers the data received from the HIL404 over a fragment of time. One can easily check that the values (with their respective message IDs from 0 to 3) match the constants connected to the inputs of the *CanBus communication* library. At the top right, we see the evolution of the variables transmitted from the Arduino to the HIL device (increasing by 0.1 each time they are sent). Finally, the graph at the bottom right is simply a counter of the number of received signals.

Once CAN Bus protocol between the DGRs and their respective local controllers have been proven successful, we address the following and last communication layer, i.e., the wireless exchange of information among the nodes in the cyber network of the microgrid.

3.2.3. CAN Bus – XBee Communication

We include the Arduino code developed for the CAN Bus communication into a new Arduino library named as CANBUS_XBEE. Although this is not the work of the author of this paper, this library includes functions that allow the local controllers to receive information via CAN Bus and to send it through the XBee RF modules (*canbus2xbee*), and vice versa (*xbee2canbus*).

Then, as for CAN Bus, we create a new Arduino script for the CAN Bus – XBee communication. Here we include CANBUS_XBEE and a slightly modified version of the XBee library -part of the XBee-Arduino

Application Programming Interface (API)-. As before, we use this code to set pins, fix baud rates, assign objects to libraries, initialize serial ports and specify one for XBee, and, crucially, to declare the neighboring agents for each local controller (therefore, this will vary for each node in the cyber network). The reader is referred to Appendix B for the complete code.

Regarding the Typhoon models utilized to implement CAN Bus – XBee communication, we use the same schematic described in the previous section (see Figure 22) and included in the *CanBus communication* library. Since at least two nodes are required to test this communication layer, two models -implemented in two different HIL404 devices- like the one illustrated in Figure 24 have been used. Figure 26 shows them together.

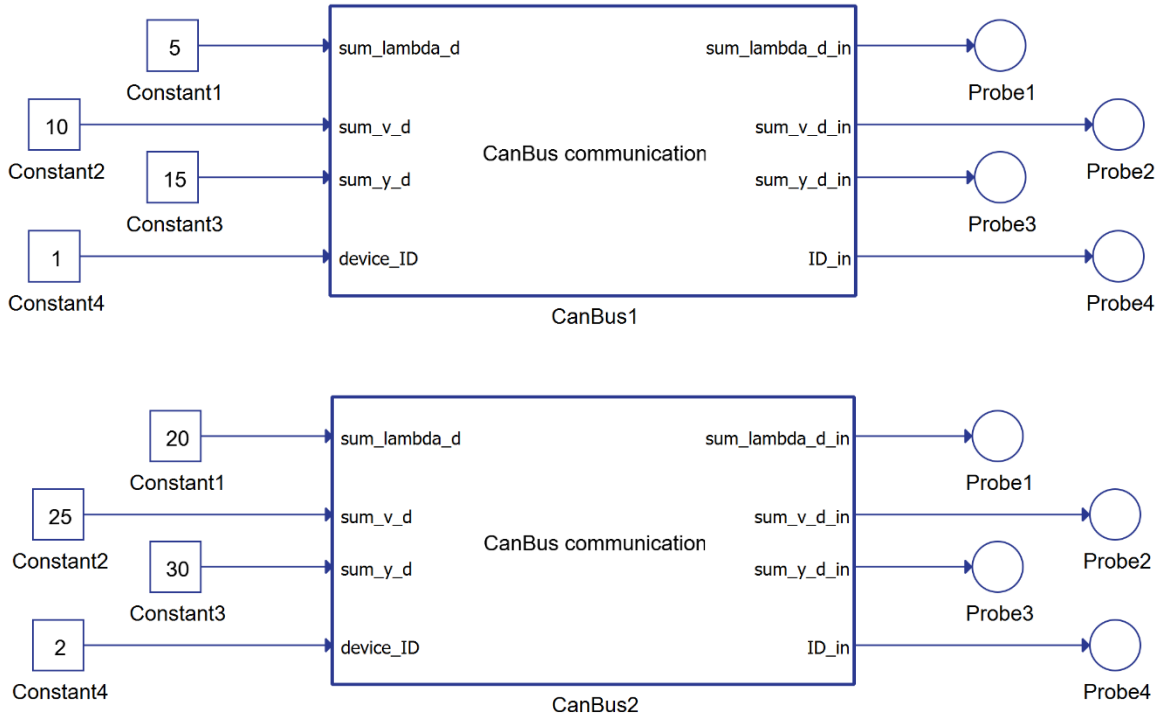

Figure 27. Models for CAN Bus – XBee communication test – Typhoon HIL Schematic Editor.

As in the previous test, we will not be using the Robust Distributed Primal-Dual algorithm but exchanging some constant values between both nodes, since the only objective here is to test the performance of the communication protocol. We are passing to the first HIL404 the same inputs as before (5, 10, 15 and 1), whereas the second DGR will broadcast the values 20, 25, 30 and 2 (recall that each model is uploaded to a different device). Therefore, we expect to see every Arduino microcontroller receiving CAN Bus data from its own HIL404 on one side, and XBee data coming from the neighboring controllers on the other, which in turn should be able to communicate back to its HIL device via CAN Bus.

Figure 27 tries to gather these results for node 1. The header of the Arduino Serial Port Monitor with the port number (COM4) has been included for an easier distinction between nodes.

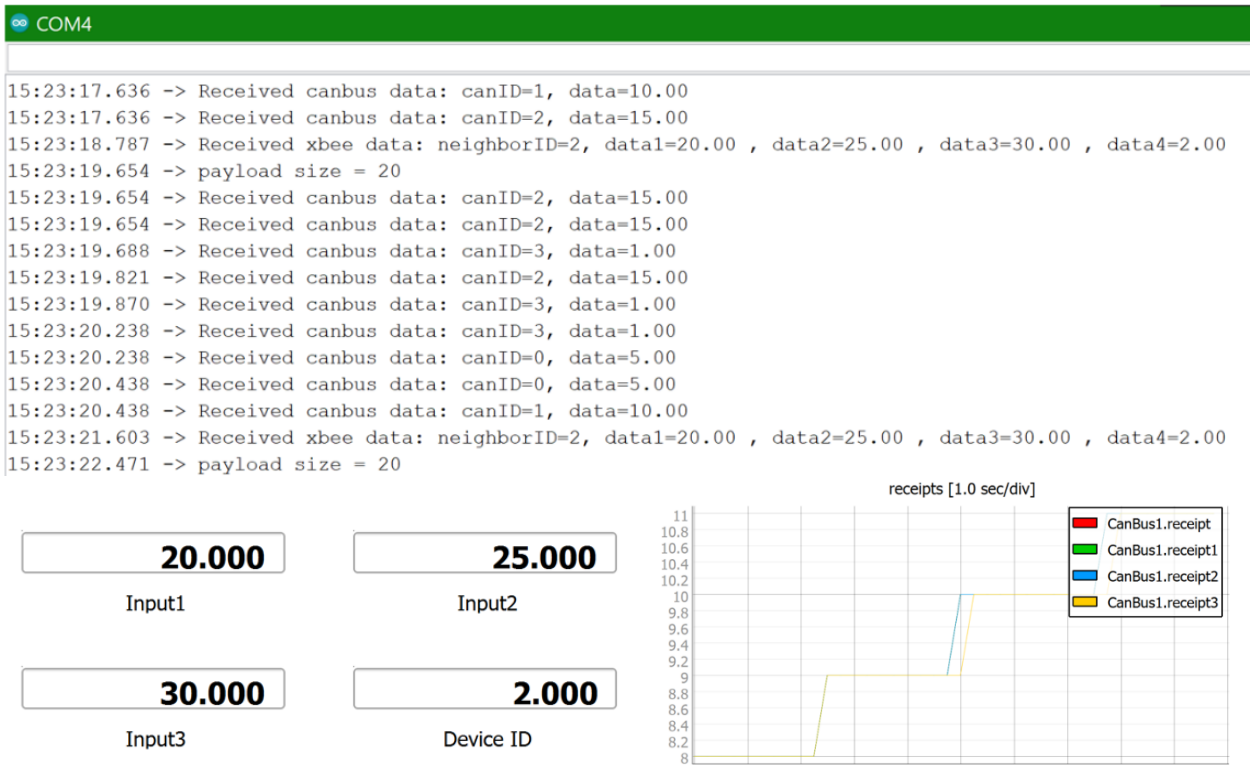

Figure 28. Node 1 CAN Bus – XBee communication test results – Arduino Serial Port Monitor (top), HIL SCADA (bottom).

Clearly, the local controller receives via CAN Bus the constant inputs passed to the HIL404 of its same node, but it also receives XBee data taking the values broadcasted by node 2 (*neighborID=2, data=20, 25, 30, 2*). Then, this information is sent to its own HIL device and, in this case, displayed in HIL SCADA. Figure 27 (top) shows the Serial Port Monitor of the Arduino microcontroller, four digital displays with the four XBee signals finally received by the HIL404 from the Arduino (bottom left) and the evolution of the counter of received information packets (bottom right). The same results are depicted in Figure 28 for node 2.

It is worth mentioning that other tests with more devices showed that, in fact, only the data from the specified neighboring agents (see Arduino code for CAN Bus – XBee communication in Appendix B) was received by the local controllers, while the information coming from the rest was blocked. In these cases, the Arduino detected that something was arriving from those nodes, but the information was not read nor displayed.

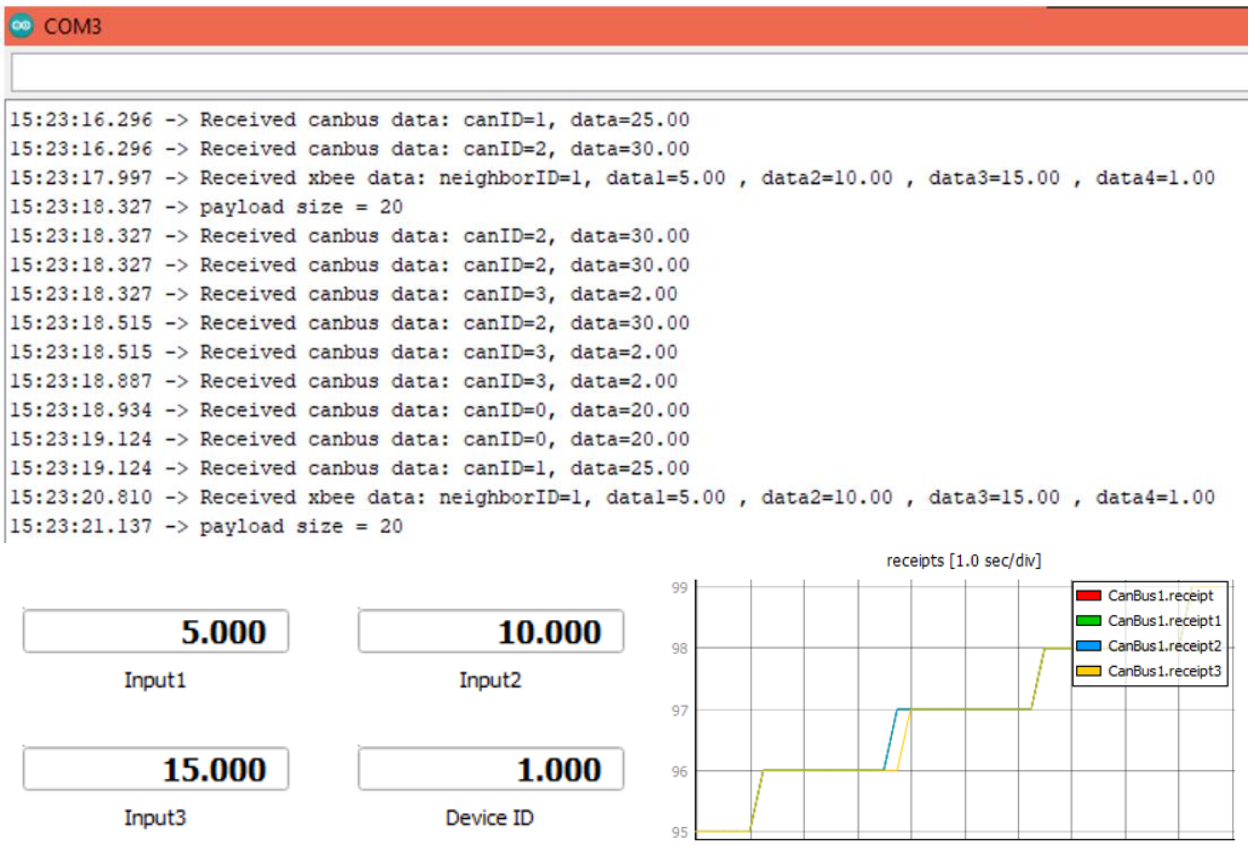

Figure 29. Node 2 CAN Bus – XBee communication test results – Arduino Serial Port Monitor (top), HIL SCADA (bottom).

Now that the distributed control algorithms have already been developed, and these communication protocols have been proved capable of wirelessly exchanging information among the nodes in the cyber network of the microgrid, we have all the elements to implement optimal dispatch in a distributed fashion. However, as it will be shown in the following section, due to time constraints and other difficulties, this could not be accomplished in practice.

3.3. Optimal Dispatch Distributed Implementation

Recall from section 3.1.4. that the optimal dispatch could be successfully achieved in a distributed fashion through the Robust Distributed Primal-Dual algorithm described in chapter 2.2. and modeled in the Typhoon HIL Schematic Editor toolchain as in Figure 20. To address its practical implementation, it is required to complete the model in section 3.1.4 by incorporating the CAN Bus functionality to each node. Therefore, and since a separate model must be uploaded to each HIL404 DGR in order to run the algorithm simultaneously in all of them, we create four parallel schematics as the one depicted in Figure 29 corresponding to node 3.

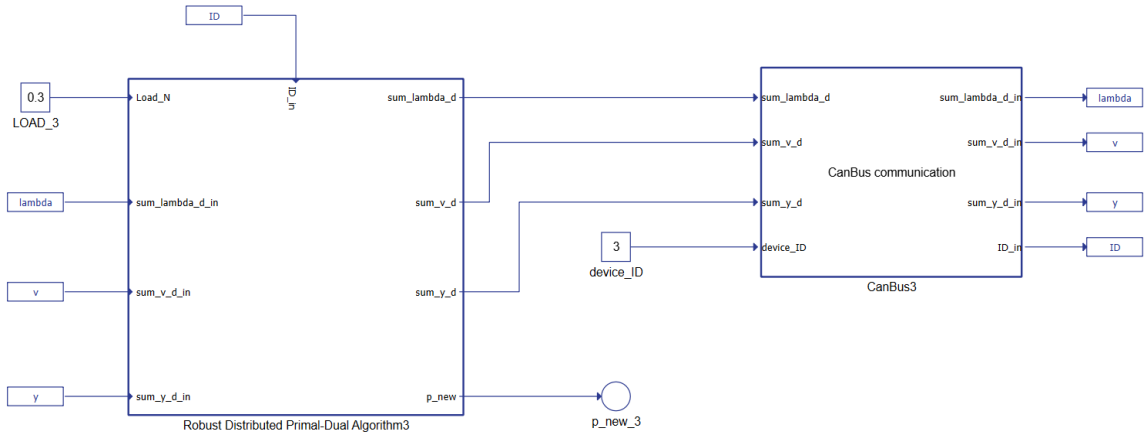

Figure 210. Node 3 model for optimal dispatch implementation – Typhoon HIL Schematic Editor.

The *CanBus communication* library on the right-hand side is identically the same described in section 3.2.2. The block on the left has been also created as a Typhoon library, and it comprises the C-function where the Robust Distributed Primal-Dual algorithm in included, together with some so-called in the model Input Functions. Figure 30 shows the model behind the library mask.

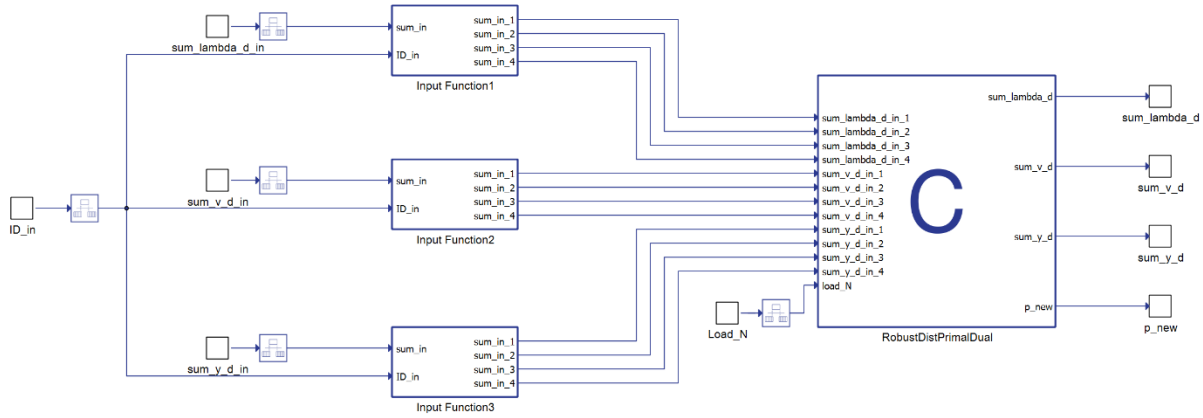

Figure 11. Robust Distributed Primal-Dual library model – Typhoon HIL Schematic Editor.

On the right, Robust Distributed Primal-Dual algorithm C-function has already been described earlier, and it has only suffered minor changes. The three Input Functions (Appendix A) on the left-hand side, which have also been included in a library (since the three of them are identical) to facilitate its modification, have two main objectives. On the one hand, they receive two inputs, i.e., one of the values broadcasted by each local controller (sum_lambda_d, sum_v_d, sum_y_d) and the ID of the node they are coming from (0 – 3), and they assign the received value to the appropriate input of the algorithm C-function block according to the identifier. On the other hand, they include four flags that are raised whenever a signal is received from a certain node. This attempts to prevent two values sent by same node from being used in the same iteration in case the local controllers are not transmitting with the same reliability. Finally, note also that rate transition components have been used in every input to account for the fact that the CanBus communication system is not operating at the same execution rate as the optimal dispatch algorithm.

Unfortunately, this setup did not allow the successful laboratory-grade implementation of the algorithm. We experienced multiple connectivity issues, mainly trying to use Ethernet to upload and run the models on the four HIL devices from a single computer; and we struggled to use the HIL standalone boot configuration to activate the models automatically with no connection to a computer. Another possible solution consisted of implementing the same four-node model -instead of four different one-node schematics- onto the four HIL404. After some adjustments regarding the HIL devices IDs and having removed the extra CAN Setup components and included some device markers to every input signal in the model, the resulting schematic is illustrated in Figure 31.

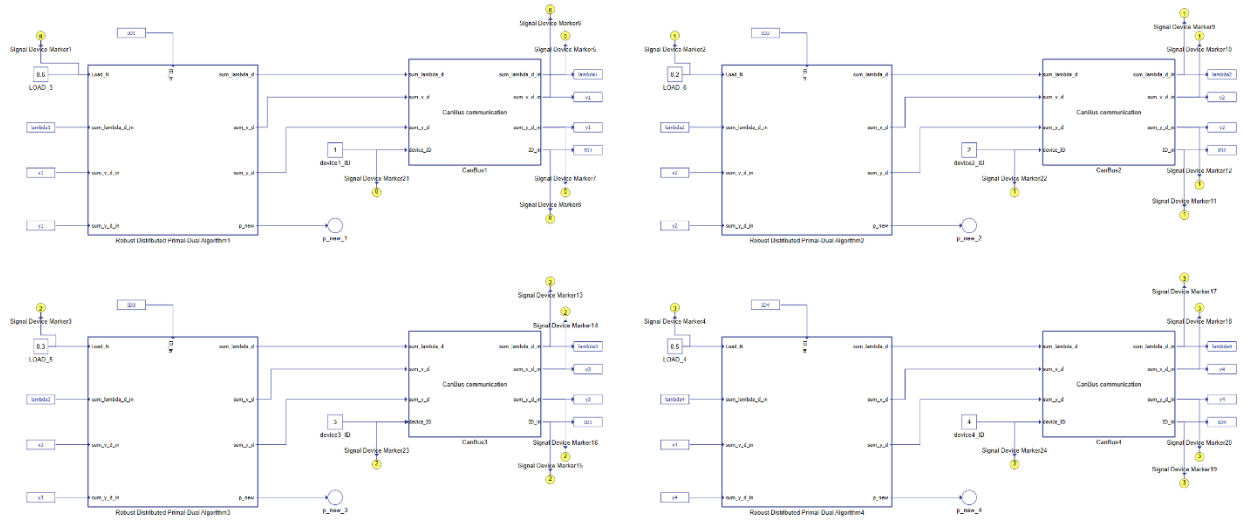

Figure 12. Four-node model for optimal dispatch implementation – Typhoon HIL Schematic Editor.

Nevertheless, this approach did not seem to yield the expected results either. The only strategy that seemed to be functional was the utilization of four different computers connected to each of the HIL404 devices to implement the four individual models depicted in Figure 29 separately. However, possibly due to the lack of a proper clock synchronization and initialization protocols as the included in [3], or maybe because of minor details in the code that happened to become crucial in practice, we often ran into data overrun issues and the appearance of NaN values.

We managed to solve most of these problems in the latest version of the models that has been described above, but the simulations resulted still in the non-convergence of the generators' power outputs. However, in previous sections, the performance of Robust Distributed Primal-Dual algorithm on the implementation the optimal dispatch function has been proved -on software. Besides, we have tested and demonstrated the effectiveness of the communication protocols described above to allow for the exchange of information through the cyber layer of the microgrid. Thus, it is reasonable to conclude that we are close to the successful implementation of distributed optimal dispatch, and this work is left open for further developments.

4. Conclusion

Microgrids are physically smaller and usually have lower power ratings and capacities than large power systems. In spite of these differences, we have seen that the operation of both microgrids and large electrical systems are subject to the same three requirements, i.e., supply must match demand, frequency must be kept close to its nominal value, and total generation cost must be minimized. In order to meet these conditions, a three-layered generation control architecture is implemented in any type of power system.

Among these layers, only the bottom one (droop control) relies on local information, whereas the other two (frequency regulation and optimal dispatch) depend on a centrally-located computer responsible for coordinating the control functions. As mentioned, this architecture is already well established in large power systems, but it is not in the emerging field of microgrids. The work proposed here has attempted to prove that a distributed generation control architecture, based on local information exchanged with the neighboring nodes and simple computations, can perform the same functions as a centralized one. A distributed approach would increase the adaptability of the system to add or remove generating units without affecting the rest of the network, while avoiding the need for resource-consuming central processor.

Even though we have not been able to complete a successful optimal dispatch simulation in a laboratory testbed, the work described above has programmed and modeled several distributed control algorithms and proved -with software simulations- the capability of Robust Distributed Primal Dual algorithm to distributively implement optimal dispatch. Moreover, we have developed and tested the performance of CAN Bus and CAN Bus – XBee communication protocols in the exchange of local information between the nodes participating in the distributed algorithms. All this adds to the existing groundwork on distributed generation control architectures for islanded AC microgrids -a piece of which can be found among the references used in the writing of this paper- and allows for further developments based on what has been proposed herein.

Appendix A: Distributed Control Algorithms

For being practically identical, only the code corresponding to node 1 in each model will be included in this appendix. It must be noted that the Typhoon HIL C-Function component organizes the code as follows: the initialization function (init_fnc) is only run once, then, the output (output_fnc) and update (update_fnc) functions are run continuously and in this order.

A.1. Ratio-Consensus Algorithm

//INIT_FNC

```
y_ini = 0.35;
z_ini = 0.15;
```

//The initial values for the internal states of the rest of the nodes can be found in Section 3.1.2.

```
y_D = y_ini/D;
z_D = z_ini/D;
```

//OUTPUT_FNC

```
gamma = y_D/z_D;
```

//UPDATE_FNC

```
yin_D[0] = yin1_D;
yin_D[1] = yin2_D;
yin_D[2] = yin3_D;
yin_D[3] = yin4_D;
zin_D[0] = zin1_D;
zin_D[1] = zin2_D;
zin_D[2] = zin3_D;
zin_D[3] = zin4_D;

sum_y = 0;
sum_z = 0;

for (i=0; i<=3; i++) {
   sum_y = sum_y + yin_D[i];
   sum_z = sum_z + zin_D[i];
}
```

```
y_D = sum_y/D;
z_D = sum_z/D;
```

A.2. Robust Ratio-Consensus Algorithm

//INIT_FNC

```
N=1;  //node number

y_ini = 0.35;
z_ini = 0.15;
```

The initial values for the internal states of the rest of the nodes can be found in Section 3.1.2.

```
mu = 0;
sigma = 0;

for (i=0; i<=3; i++) {
    vij[i] = 0;
    tauij[i] = 0;
}
```

//OUTPUT_FNC

```
mu = mu + y/D;
sigma = sigma + z/D;

gamma = y/z;
```

//UPDATE_FNC

```
mu_in[0]=mu_in_1;
mu_in[1]=mu_in_2;
mu_in[2]=mu_in_3;
mu_in[3]=mu_in_4;
sigma_in[0]=sigma_in_1;
sigma_in[1]=sigma_in_2;
sigma_in[2]=sigma_in_3;
sigma_in[3]=sigma_in_4;
```

```
for (i=0; i<=3; i++) {
    vij_new[i]=mu_in[i];
    tauij_new[i]=sigma_in[i];
}

sum_v = 0;
sum_tau = 0;

for (i=0; i<=3; i++) {
    if (i!=N-1){
        sum_v = sum_v + (vij_new[i]-vij[i]);
        sum_tau = sum_tau + (tauij_new[i]-tauij[i]);
    }
}

y = y/D + sum_v;
z = z/D + sum_tau;

for (i=0; i<=3; i++) {
    vij[i] = vij_new[i];
    tauij[i] = tauij_new[i];
}
```

A.3. Robust Distributed Primal Dual Algorithm

```
//INIT_FNC

N=1;    //node number

p = 0;
p_max = 1;
p_min = -1;

a = 0.1;    //cost function parameters
b = 0;
c = 0;
s = 0.02;
epsilon = 0.2;
gamma = 0.9;
n = 4;    //estimate of the number of nodes

x = 0;
lambda = 0;
v = 1;
```

```
y = n*(p - load_N);

for (i=0; i<=3; i++) {
        lambdaij[i] = 0;
        vij[i] = 0;
        yij[i] = 0;
}
```

//OUTPUT_FNC

```
sum_lambda_d = sum_lambda_d + lambda/d;
sum_v_d = sum_v_d + v/d;
sum_y_d = sum_y_d + y/d;

p_new = p - s*(a*p*p+b*p+c) + s*epsilon*x;

    if (p_new<=p_min) {
        p_new = p_min;
    } else if (p_new>=p_max) {
        p_new = p_max;
    }
```

//UPDATE_FNC

```
sum_lambda_d_in[0]=sum_lambda_d_in_1;
sum_lambda_d_in[1]=sum_lambda_d_in_2;
sum_lambda_d_in[2]=sum_lambda_d_in_3;
sum_lambda_d_in[3]=sum_lambda_d_in_4;

sum_v_d_in[0]=sum_v_d_in_1;
sum_v_d_in[1]=sum_v_d_in_2;
sum_v_d_in[2]=sum_v_d_in_3;
sum_v_d_in[3]=sum_v_d_in_4;

sum_y_d_in[0]=sum_y_d_in_1;
sum_y_d_in[1]=sum_y_d_in_2;
sum_y_d_in[2]=sum_y_d_in_3;
sum_y_d_in[3]=sum_y_d_in_4;

for (i=0; i<=3; i++) {
    if (i==N-1) {
        lambdaij_new[i] = lambdaij[i] + lambda/d;
        vij_new[i] = vij[i] + v/d;
```

```
        yij_new[i] = yij[i] + y/d;
    } else {
        lambdaij_new[i] = (1-gamma)*lambdaij[i] +
gamma*sum_lambda_d_in[i];
        vij_new[i] = (1-gamma)*vij[i] + gamma*sum_v_d_in[i];
        yij_new[i] = (1-gamma)*yij[i] + gamma*sum_y_d_in[i];
    }
}

sum_lambda = 0;
sum_v = 0;
sum_y = 0;

for (i=0; i<=3; i++) {
    sum_lambda = sum_lambda + (lambdaij_new[i] - lambdaij[i]) -
s*(yij_new[i] - yij[i]);
    sum_v = sum_v + (vij_new[i] - vij[i]);
    sum_y = sum_y + (yij_new[i] - yij[i]);
}

lambda = sum_lambda;
v = sum_v;
x = lambda/v;
y = sum_y + n*(p_new - p);

for (i=0; i<=3; i++) {
    lambdaij[i] = lambdaij_new[i];
    vij[i] = vij_new[i];
    yij[i] = yij_new[i];
}

p = p_new;
```

A.4. Input Function - Robust Distributed Primal-Dual Library

```
//INIT_FNC

aux1 = 0;
aux2 = 0;
aux3 = 0;
aux4 = 0;

timer = 0;
t = 0;
```

```
flag1 = 0;
flag2 = 0;
flag3 = 0;
flag4 = 0;


//OUTPUT_FNC

sum_in_1 = aux1;
sum_in_2 = aux2;
sum_in_3 = aux3;
sum_in_4 = aux4;


//UPDATE_FNC

timer += Ts;

if((timer-t) < (Ts*10)) {

    if ((ID_in == 1) && (flag1 == 0)) {
        aux1 = sum_in;
        flag1 = 1;
    }

    if ((ID_in == 2) && (flag2 == 0)) {
        aux2 = sum_in;
        flag2 = 1;
    }

    if ((ID_in == 3) && (flag3 == 0)) {
        aux3 = sum_in;
        flag3 = 1;
    }

    if ((ID_in == 4) && (flag4 == 0)) {
        aux4 = sum_in;
        flag4 = 1;
    }
}

else {
    flag1 = 0;
    flag2 = 0;
```

```
    flag3 = 0;
    flag4 = 0;

    t = timer;
}

if(timer>10000) {
    timer = 0;
    t = timer;
}
```

Appendix B: Communication Protocols

We include here the Arduino scripts utilized for the CAN Bus and CAN Bus – XBee communication tests to define pins, fix baud rates, coordinate the microcontroller incoming and outcoming information display the received data and, in the case of the CAN Bus – XBee, declare the neighboring agents for each local controller.

B.1. CAN Bus Communication

```
// demo: CAN-BUS Shield, receive data with check mode, send data

#include <SPI.h>

#define CAN_2515     // #define CAN_2518FD

// Set SPI CS Pin according to your hardware

#if defined(SEEED_WIO_TERMINAL) && defined(CAN_2518FD)
// For Wio Terminal w/ MCP2518FD RPi Hat:
// Channel 0 SPI_CS Pin: BCM 8
// Channel 1 SPI_CS Pin: BCM 7
// Interupt Pin: BCM25
const int SPI_CS_PIN  = BCM8;
const int CAN_INT_PIN = BCM25;

#else
// For Arduino MCP2515 Hat:
// the cs pin of the version after v1.1 is default to D9
// v0.9b and v1.0 is default D10
const int SPI_CS_PIN = 9;
const int CAN_INT_PIN = 2;
#endif

#ifdef CAN_2518FD
#include "mcp2518fd_can.h"
mcp2518fd CAN(SPI_CS_PIN); // Set CS pin
#endif

#ifdef CAN_2515
#include "mcp2515_can.h"
mcp2515_can CAN(SPI_CS_PIN); // Set CS pin
#endif
```

```
union {
    byte b[8];    //Convert floating variables to bytes and vice-versa
    float f;
} data;

float CanID;
unsigned char len = 0;
unsigned char info[4], sum_lambda_d[4], sum_v_d[4], sum_y_d[4], ID[4];
float sum_lambda_d_in=1,  sum_v_d_in=2, sum_y_d_in=3, ID_in=0;


void setup() {
    SERIAL_PORT_MONITOR.begin(115200);

    while (CAN_OK != CAN.begin(CAN_500KBPS)) {   // init can bus :
baudrate = 500k
        SERIAL_PORT_MONITOR.println("CAN init fail, retry...");
        delay(100);
    }
    SERIAL_PORT_MONITOR.println("CAN init ok!");
}


void loop() {

    //RECEIVE DATA

    if (CAN_MSGAVAIL == CAN.checkReceive()) {

        SERIAL_PORT_MONITOR.println("-----------------------------");

        CAN.readMsgBuf(&len, info);
        CanID = CAN.getCanId();

        if (CanID == 0) {

          for (int i = 0; i < 4; i++) {
            sum_lambda_d[i] = info[i];
          }
          SERIAL_PORT_MONITOR.print("CanID ");
          SERIAL_PORT_MONITOR.print(CanID,1);
          SERIAL_PORT_MONITOR.print(": ");
          for (int i = 0; i < len; i++) {
            data.b[i] = sum_lambda_d[i];
          }
```

```
      SERIAL_PORT_MONITOR.println(data.f,3);
    }

    if (CanID == 1) {

      for (int i = 0; i < 4; i++) {
        sum_v_d[i] = info[i];
      }
      SERIAL_PORT_MONITOR.print("CanID ");
      SERIAL_PORT_MONITOR.print(CanID,1);
      SERIAL_PORT_MONITOR.print(": ");
      for (int i = 0; i < len; i++) {
        data.b[i] = sum_v_d[i];
      }
      SERIAL_PORT_MONITOR.println(data.f,3);
    }

    if (CanID == 2) {

      for (int i = 0; i < 4; i++) {
        sum_y_d[i] = info[i];
      }
      SERIAL_PORT_MONITOR.print("CanID ");
      SERIAL_PORT_MONITOR.print(CanID,1);
      SERIAL_PORT_MONITOR.print(": ");
      for (int i = 0; i < len; i++) {
        data.b[i] = sum_y_d[i];
      }
      SERIAL_PORT_MONITOR.println(data.f,3);
    }

    if (CanID == 3) {

      for (int i = 0; i < 4; i++) {
        ID[i] = info[i];
      }
      SERIAL_PORT_MONITOR.print("CanID ");
      SERIAL_PORT_MONITOR.print(CanID,1);
      SERIAL_PORT_MONITOR.print(": ");
      for (int i = 0; i < len; i++) {
        data.b[i] = ID[i];
      }
      SERIAL_PORT_MONITOR.println(data.f,3);
    }
}
```

```
    // SEND DATA

    sum_lambda_d_in+=0.1,sum_v_d_in+=0.1,sum_y_d_in+=0.1,ID_in+=0.1;

    data.f = sum_lambda_d_in;
    CAN.sendMsgBuf(100, 0, 4, data.b);
    delay(100);

    data.f = sum_v_d_in;
    CAN.sendMsgBuf(101, 0, 4, data.b);
    delay(100);

    data.f = sum_y_d_in;
    CAN.sendMsgBuf(102, 0, 4, data.b);
    delay(100);

    data.f = ID_in;
    CAN.sendMsgBuf(103, 0, 4, data.b);
    delay(100);

    SERIAL_PORT_MONITOR.println("CAN BUS sendMsgBuf ok!");
}

// END FILE
```

B.2. CAN Bus – XBee Communication

```
#include <Streaming.h>
#include <XBee.h>
#include <mcp2515_can.h>
#include <CANBUS_XBEE.h>

// For Arduino MCP2515 Hat:
// the cs pin of the version after v1.1 is default to D9
// v0.9b and v1.0 is default D10
const int SPI_CS_PIN = 9;
const int CAN_INT_PIN = 2;
uint8_t sPin = 7;
uint8_t cPin = 48;

// OBJECTS FOR USER DEFINED LIBRARIES: canbus, xBee, and canbusxbee
mcp2515_can CAN(SPI_CS_PIN); // Set CS pinModBus
XBee xbee = XBee();
ZBRxResponse rx = ZBRxResponse();
```

```
CANBUS_XBEE cx = CANBUS_XBEE(&CAN,&xbee,&rx,1);

void setup() {

  // SERIAL PORTS
  Serial.begin(38400);
  Serial3.begin(38400);
  pinMode(cPin, OUTPUT);
  pinMode(sPin, OUTPUT);
  digitalWrite(cPin,HIGH);
  digitalWrite(sPin,HIGH);

  // ASSIGN SERIAL PORTS TO xBee AND task PROPERTIES
  xbee.setSerial(Serial3); //Specify the serial port for xbee

  // DECLARE NEIGHBORING AGENTS: must be specified by the user for every
microcontroller participating in the CAN Bus - XBee communication
  cx.addInNeighbor(2);
  cx.addInNeighbor(3);

  while (CAN_OK != CAN.begin(CAN_500KBPS)) {     // init can bus:
baudrate = 500k
      Serial.println("CAN init fail, retry...");
      delay(100);
  }
  Serial.println("CAN init ok!");

  // TURN OFF ARDUINO PINS
  digitalWrite(cPin,LOW);
  digitalWrite(sPin,LOW);
}

void loop() {

  //CANBUS to XBEE
  cx.canbus2xbee(4);

  delay(1000);

  //XBEE TO CANBUS
  cx.xbee2canbus(4);
}

// END FILE
```

Appendix C: Sustainable Development Goals (SDGs)

Established by the United Nations in 2015, the Sustainable Development Goals (SGDs) were meant to be an appeal to end poverty, inequality, war and violence, injustice, and to fight climate change and the degradation of the planet. These 17 principles lead the way to face the main global challenges and move towards a sustainable prosperity [14].

Even though the work that has been presented above cannot be clearly framed within a specific Sustainable Development Goal, it has been deemed consistent with the following SDGs and specific goals:

SDG DIMENSION	SDG IDENTIFIED	ROLE	GOAL
Biosphere	SGD13: Take urgent action to combat climate change and its effects.	Secondary	13.2 Incorporate climate change measures into national policies, strategies, and plans.
Society	SDG7: Ensure access to an affordable, reliable, sustainable, and modern energy.	Primary	7.1 By 2030, ensure universal access to affordable, reliable, and modern energy services. 7.2 By 2030, significantly increase the share of renewable energy in the global energy mix.
Economy	SDG9: Build resilient infrastructures, promote sustainable industrialization, and foster innovation	Secondary	9.4 By 2030, upgrade infrastructure and convert industries to make them sustainable, using resources more efficiently and promoting the adoption of clean and environmentally rational technologies and industrial processes.

Table 1. Project alignment with the Sustainable Development Goals (SDGs) [14]

Table 1 shows the identified SDGs -in this case, each one of them corresponding to each of the three dimensions of sustainability-, the primary or secondary role of each SGD, and the specific goal mostly

affected by the work developed throughout this document. We proceed now to suggest how the project interacts with these concrete objectives.

Regarding *SDG 13: Climate Action*, the incorporation of climate change measures must be undoubtedly made through the electricity field. Over the last decades, national policies and plans have been adopted to move towards the fossil-electricity retirement. Microgrids' inherit characteristics, with smaller power ratings and capacities and a decentralized structure, offer a great opportunity to easily integrate renewable-based generation technologies into the utility grid [10].

It has been considered that the *SDG 9: Industry, Innovation, and Infrastructure* could be also aligned with this project. In fact, microgrids could also help in the upgrade of electricity infrastructure to make it more sustainable, efficient, clean, and environmentally rational. In this respect, the distributed generation control architecture that has been worked on, could contribute to the renewal of the electricity grid with less resource-consuming and more renewable-based networks [3].

The seventh Sustainable Development Goal, i.e., *Affordable and Clean Energy,* has been estimated to be the most closely related to this project. The decarbonization of the electricity grid has become a priority within the global goals for a prosperous and sustainable future. It is key for fighting climate change and for keeping it under safe levels. In particular, the energy production is the cause of 60% of greenhouse emissions and, consequently, the main contributor to global warming. Moreover, it has become critical to facilitate the access to energy and electricity in the developing countries. United Nations has estimated that 13% of global population still do not have access to modern electricity services, and that 3000 million people still depend on wood, charcoal, or animal waste for cooking [14]. In this regard, we must take urgent action to bring a clean and efficient electrification to these developing regions.

Table 1 identifies to specific goals within this SDG. As it has been mentioned, the structural characteristics of microgrids make them particularly suitable for integrating of renewable-based generation. Besides, distributed control architectures could favor the participation of inverter-interfaced generators in generation control functions (note that, throughout this document, the term DGR has referred to both synchronous generators and their inverter-interfaced counterparts). Finally, recall that the main objective of this distributed architecture was to increase the grid efficiency by optimizing the generation control functions. In comparison with the traditional centralized approach, the distributed implementation would increase the adaptability of the system to add or remove generating units without affecting the rest of the network, while avoiding the need for resource-consuming central processor [3].

References

[1] Arduino Store Arduino Due, web page. Available at: https://store.arduino.cc/products/arduino-due

[2] D. P. Bertsekas, "Lagrange Multiplier Theory", in *Nonlinear Programming*, 2nd ed., Athena Scientific, 1999, pp. 446-472.

[3] S. T. Cady, A. D. Domínguez-García and C. N. Hadjicostis, "A Distributed Generation Control Architecture for Islanded AC Microgrids," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 5, pp. 1717-1735, September 2015.

[4] A. D. Domínguez-García, "Chapter 6: Load and Generator Modeling", in *Lecture Notes on Power System Analysis and Control*, class notes for ECE 476, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, 2021, pp. 107-122.

[5] A. D. Domínguez-García, "Chapter 9: Optimal Dispatch", in *Lecture Notes on Power System Analysis and Control*, class notes for ECE 476, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, 2021, pp. 168-180.

[6] A. D. Domínguez-García and C. N. Hadjicostis, "Distributed algorithms for control of demand response and distributed energy resources," in *Proc. 50th IEEE Conf. Decision Control*, December 2011, pp. 27–32.

[7] A. D. Domínguez-García, C. N. Hadjicostis, and N. H. Vaidya, "Resilient networked control of distributed energy resources," *IEEE J. Sel. Areas Commun.*, vol. 30, no. 6, pp. 1137–1148, July 2012.

[8] A. Gómez-Expósito, A. J. Conejo and C. Cañizares, *Electric Energy Systems: Analysis and Operation (Electric Power Engineering Series)*, 1st ed., CRC Press, 2009, pp. 355-400.

[9] C. N. Hadjicostis, N. H. Vaidya, and A. D. Domínguez-García, "Robust distributed average consensus via exchange of running sums," *IEEE Transactions on Automatic Control*, vol. 61, no. 6, pp. 1492–1507, June 2016.

[10] S. Pullins, "Why microgrids are becoming an important part of the energy infrastructure," *The Electricity Journal*, vol. 32, no. 5, pp. 17-21, June 2019.

[11] Seeedstudio CAN-BUS Shield V2.0, web page. Available at: https://wiki.seeedstudio.com/CAN-BUS_Shield_V2.0/

[12] SparkFun Electronics XBee Buying Guide, web page. Available at: https://www.sparkfun.com/pages/xbee_guide

[13] Typhoon HIL, web page. Available at: https://www.typhoon-hil.com/

[14] United Nations. Department of Economic and Social Affairs - Sustainable Development, web page. Available at: https://sdgs.un.org/goals

[15] D. B. West, *Introduction to Graph Theory*, 2nd ed., Upper Saddle River, NJ, USA: Prentice-Hall, 2001.

[16] A. J. Wood and B. F. Wollenberg, *Power Generation, Operation, and Control*, New York, NY, USA: Wiley, 1996.

[17] J. Wu, T. Yang, D. Wu, K. Kalsi, and K. H. Johansson, "Distributed optimal dispatch of distributed energy resources over lossy communication networks," *IEEE Transactions on Smart Grid*, vol. 8, no. 6, pp. 3125–3137, November 2017.

[18] M. Zholbaryssov, C. N. Hadjicostis and A. D. Domínguez-García, "Fast Coordination of Distributed Energy Resources over Time-Varying Communication Networks," 2019. Available online at: https://doi.org/10.48550/arXiv.1907.07600. Last updated 4 May 2020 06:37:38 UTC.