COMILLAS
UNIVERSIDAD PONTIFICIA

ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

ALGORITMO PARA EL CONTROL SECUNDÁRIO DE FRECUENCIA EN MICRORREDES AC AISLADAS.

Autor: Patricia Samper Lario

Director: Alejandro Donmíngez-Gracía

Madrid

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

Algoritmo para el control secundario de frecuencia en microrredes AC aisladas.

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico ……4º……. 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

Fdo.: Patricia Samper Lario Fecha: 6/ 9/ 2020

Autorizada la entrega del proyecto

Fdo.: Alejandro Domíngez-García Fecha: 6/ 9/ 2020

COMILLAS
UNIVERSIDAD PONTIFICIA

ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

ALGORITMO PARA EL CONTROL SECUNDÁRIO DE FRECUENCIA EN MICRORREDES AC AISLADAS

Autor: Patricia Samper Lario

Director: Alejandro Domínguez -García

Madrid

Agradecimientos

Me gustaría expresar mi agradecimiento al Profesor Alejandro Domíngez-García por darme la oportunidad de colaborar en este fascinante proyecto que me ha permitido adentrarme en el mundo de las microrredes y controles distribuidos.
Por otro lado también quisiera agradecer a A.Olaoluwapo ya que ha sido gracias a su ayuda y paciencia lo que ha hecho este trabajo posible.
Por último, agradecer a la universidad de Illinois por suministrar todo el equipo y las tecnologías necesarias.

ALGORÍTMO PARA EL CONTROL DE FRECUENCIA SECUNDARIO PARA MICRORREDES AC AISLADAS.

Autor: Samper Lario, Patricia.
Director: Domíngez-García, Alejandro.
Entidad Colaboradora:: Universidad de Illinois at Urbana Champaign.

RESUMEN DEL PROYECTO

Este trabajo consiste en la implementación de un control distribuido para el control secundario de frecuencia.

El grueso del proyecto se centra en implementar el algoritmo “Feasible Flow” y evaluar su rendimiento y eficacia para este tipo de redes. Para ello se empleará un banco de pruebas que ha sido diseñado para proveer un entorno de alta fiabilidad en tiempo real, para la evaluación de arquitecturas de control.

La arquitectura del sistema que se propone tiene como objetivo llevar el error de frecuencia a cero, que para las microrredes sin inercia supone asegurar que la frecuencia en cada nodo es igual al punto de operación que resulta en un sistema estable.

Finalmente se presenta una reflexión comparando los sistemas de control centralizados frente a los distribuidos.

Palabras clave: Microrred, Algoritmo, Control, Distribuido, DER´s ,Frecuencia, Resiliencia, Simulación.

1. Introducción.

El actual sistema eléctrico necesita de una transformación por varias razones, entre las que podemos encontrar: el aumento de la preocupación por el medio ambiente que deriva en un creciente uso de energías renovables, el aumento de la demanda energética y la electrificación de las zonas rurales entre otros. Todo esto está provocando una tendencia hacia la descentralización del sistema eléctrico. Es por ello por lo que las microrredes se tornan imprescindibles para cumplir con los objetivos de renovación energética, donde las renovables tengan un papel principal.

En definitiva, se puede decir que objetivo de esta transformación es conseguir un consumo más inteligente, eficiente y flexible, donde se optimicen los recursos y se facilite su implantación.

En la Universidad de Illinois se está llevando a cabo un proyecto para la creación de un sistema de control distribuido para microrredes: "Controller Hardware-in-the-Loop Testbed for Distributed Coordination and Control Architectures", liderado por el profesor Alejandro Domínguez-García.

Se trata de un trabajo innovador cuyo objetivo es desarrollar sistemas de decisión distribuidos que pueden aportar grandes beneficios para el control de estas redes, mejorando la resiliencia y adaptabilidad del sistema.

El presente proyecto de Fin de Grado es presentan en primer lugar las bases del control secundario de frecuencia en microrredes en modo aislado desarrollado en la Universidad de Illinois, estableciendo así el marco de referencia del algoritmo para el control del flujo de potencia basado en un sistema de toma de decisiones distribuido, en el que se centra este trabajo. La implementación de estos algoritmos se ha llevado a cabo empleando el lenguaje de programación C++ y mediante el uso de código orientado a objetos.

2. Definición del Proyecto

En este trabajo en primer lugar, se hablará de la transformación que está experimentando el sector de la generación y distribución eléctrica, en particular se plantea una pequeña introducción a cerca de las conocidas como "microrredes" y los nuevos sistemas de generación distribuida, junto con un análisis sobre las ventajas y desventajas que tienen estos nuevos sistemas.

Seguidamente se realiza una distinción entre los diferentes tipos de microrredes que podemos encontrar, así como los diferentes sistemas de control que estas pueden presentar.

Una vez planteado en contexto general del proyecto, el trabajo se centra en establecer las bases del control secundario de frecuencia par microrredes en modo aislado (desarrollado en la Universidad de Illinois), para finamente llegar al desarrollo de un algoritmo para el control del flujo de potencia basado en un sistema de toma de decisiones distribuido. La implementación de estos algoritmos se ha llevado a cabo empleando el lenguaje de programación C++ y mediante un código orientado a objetos que hace posible la comunicación entre los diferentes elementos de la red. El sistema de control que se propone tiene como objetivo ajustar los puntos de salida de los generadores para obtener los "sert points", de manera que se elimine el error de frecuencia consecuencia de pequeñas perturbaciones en la carga.

Estos "set points " se eligen de manera que cuando se aplican al sistema, este es estable alrededor de estos puntos de operación y el valor de frecuencia es igual a un valor de referencia.

Este algoritmo se testeará en tres modelos diferentes:

1. Sistema de tres nodos con control centralizado.
2. Sistema de tres nodos con control distribuido.
3. Simulación del control secundario de frecuencia distribuido en una microrred con seis nodos.

En los dos primeros tan solo se implementa el algoritmo para el control de flujo de potencia en un sistema de tres nodos sencillos para demostrar su eficacia. Finalmente

se hace una simulación completa del control secundario de frecuencia empleando un control distribuido para una microrred de seis nodos.

Una vez se han implementado este algoritmo, se analizarán y compararán los resultados obtenidos para llegar a una conclusión acerca de la eficacia de los sistemas de decisión distribuida frente a los sistemas tradicionales de toma de decisiones centralizada.

3. Descripción del modelo/sistema/herramienta

Con el objetivo de poder evaluar el funcionamiento del algoritmo implementado se empleará el simulador Typhoon HIL, que proporciona un entorno de pruebas eficaz, eficiente, de bajo coste, escalable y ajustable para arquitecturas de coordinación y control distribuidas.

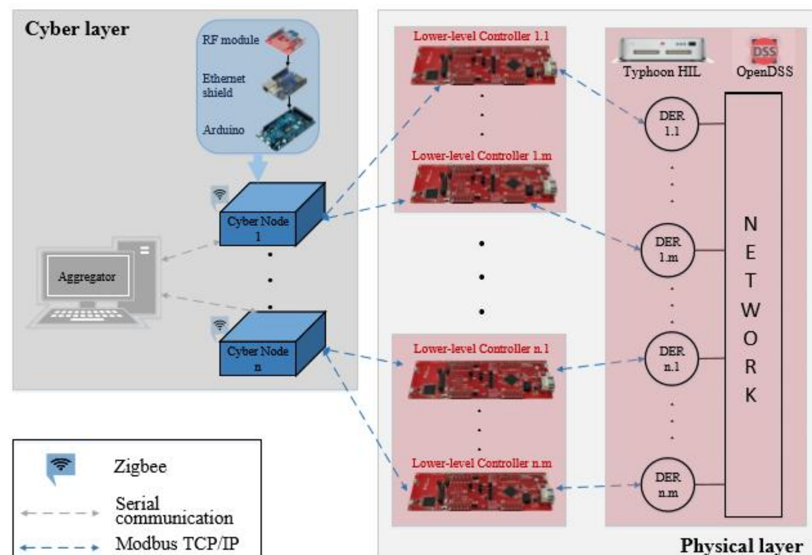


Imagen 1: Esquema del controlador del banco de pruebas Hardware in the loop.[1]

En concreto las simulaciones se realizarán para una microrred de seis nodos (tres generadores y tres cargas)como se muestra en la siguiente figura.

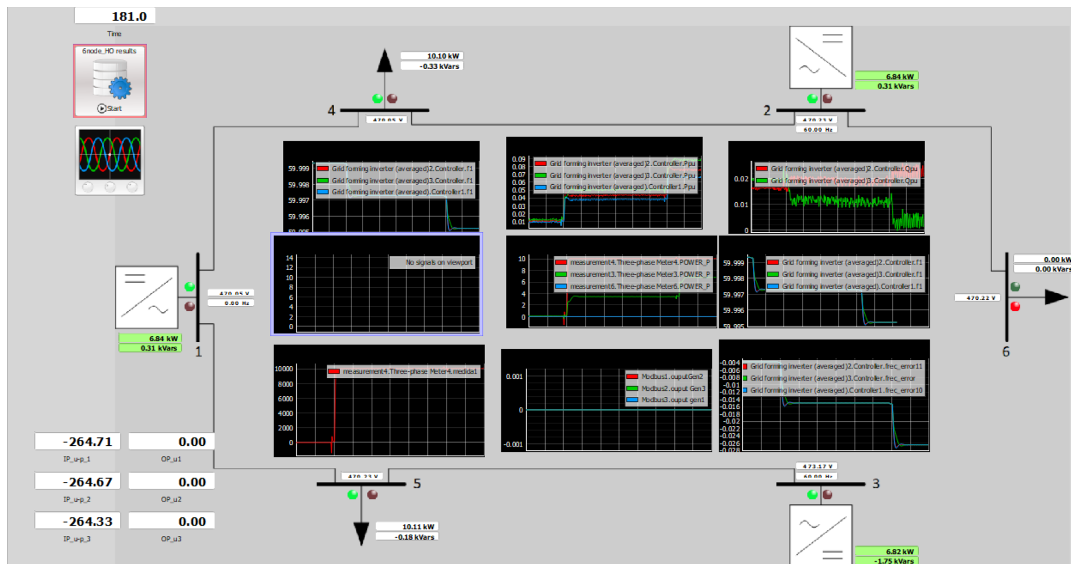


Ilustración 2: Microdred de 6 nodos(Typhoon HIL)

Por otro lado, cada nodo de la red está controlado por un controlador el cual está

compuesto por un **Arduino Due**. Los nodos cibernéticos vecinos se comunican entres ellos sin cables mediante MaxStream XB24-DMCIT-250 revB XBee interconectado al Arduino Due.

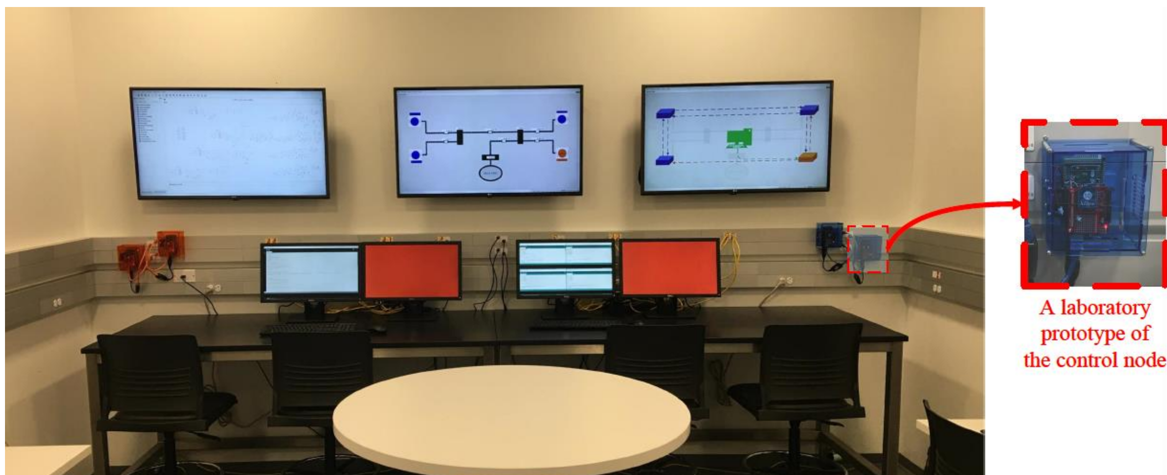


Ilustración 3:Laboratiro UIUC

4. Resultados

Tras la implementación del algoritmo "*Feasible Flow*" en los sistemas de tres nodos quedó demostrado el correcto funcionamiento y eficacia del algoritmo para determinar aquellos valores de generación individual de los generadores que suplen la demanda de las cargas en una microrred, sin violar ninguna de las restricciones de generación ni los

límites de la potencia máxima que puede circular por cada una de las ramas. Sin embargo, se puso de manifiesto un fallo común en las comunicaciones inalámbricas conocido como "drop packets" y el cual provoca que algunos de los paquetes de información enviados no lleguen al receptor. Este problema puede solucionarse mediante un algoritmo más complejo que tenga encuneta este fallo. Actualmente los ingenieros que forman parte del proyecto de la universidad de Illinois están trabajando en algoritmos que permitan el funcionamiento correcto del a pesar de este problema.

En las siguientes imágenes se muestran los resultados obtenidos en dicha prueba para el ejemplo de control distribuido, estableciendo 40 como número de iteraciones.

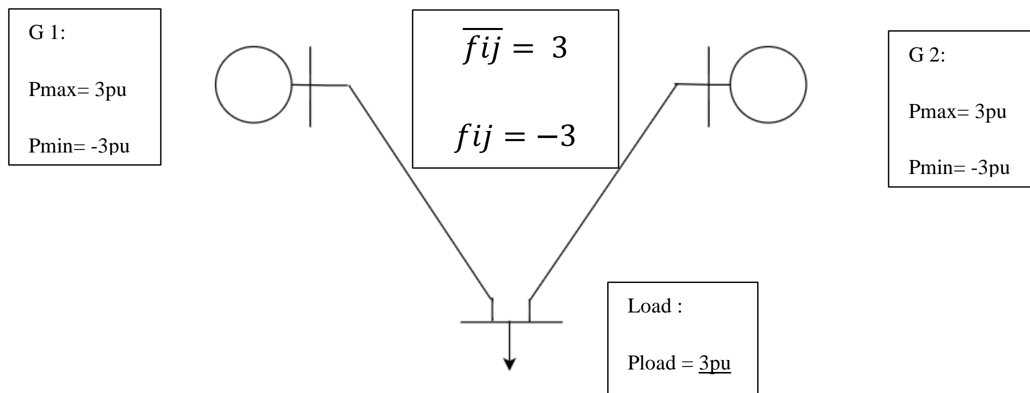


Ilustración4:Red 3 nodos

En este caso los resultados que se obtienen son los siguientes:

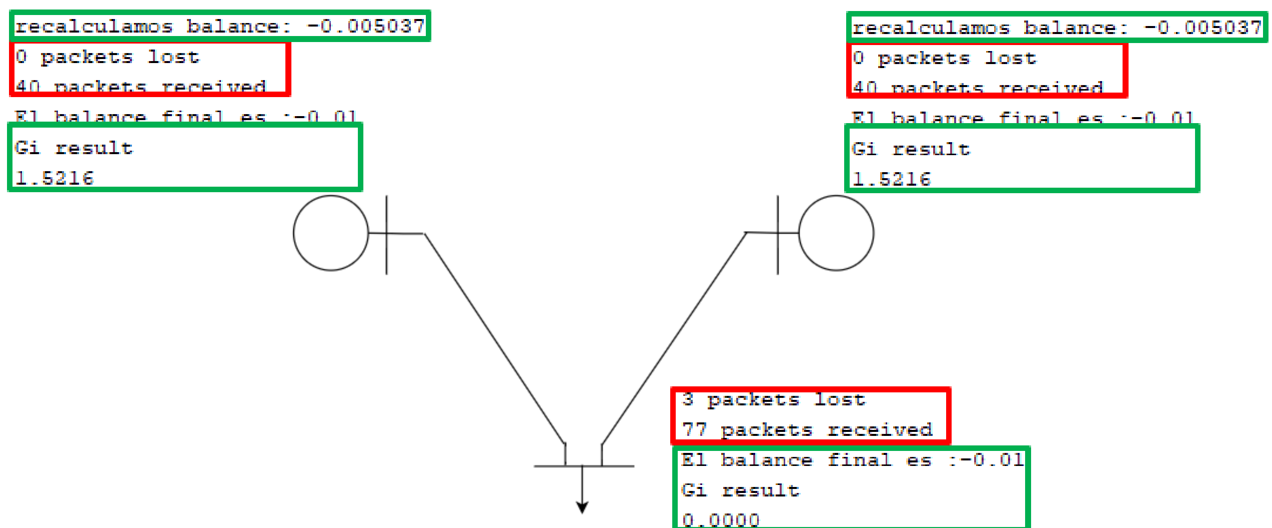
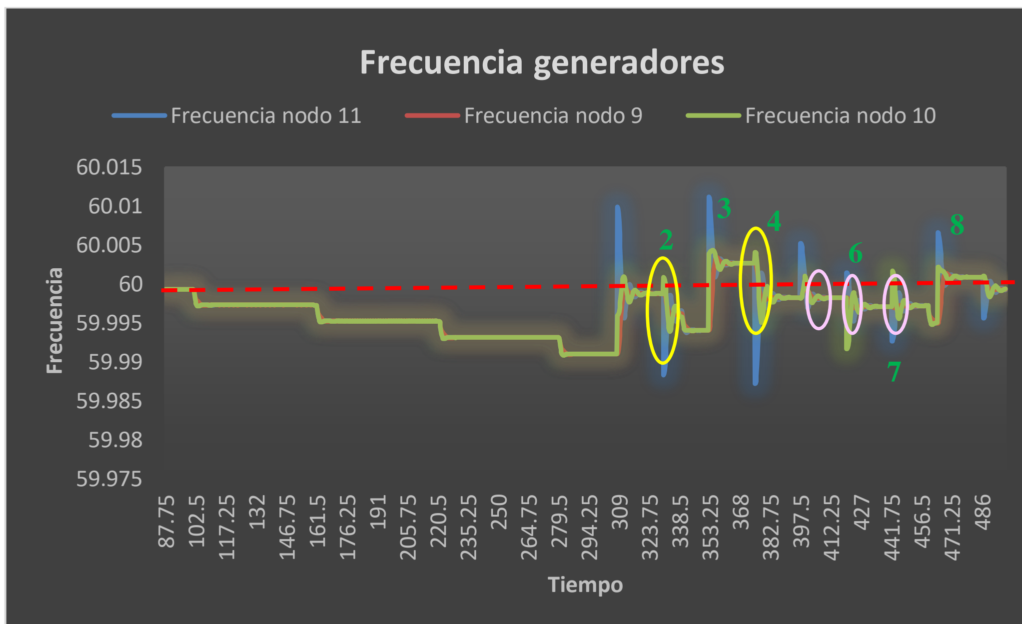


Ilustración 1:Resultados Arduinos sistema distribuido de 3 nodos

Como se puede ver, a pesar del fallo mencionado los resultados se aproximan los suficiente a los resultados esperados, que la generación de los dos generadores es prácticamente de 1.5 y el balance de los nodos próxima a cero.

Para la simulación de la microrred se han obtenido los resultados que se pueden apreciar en la siguiente gráfica en la que aparece la frecuencia junto con la evolución de la potencia demandada por las cargas. Se puede apreciar como los saltos en la frecuencia que tratan de acercar está a su valor de referencia son debidos al control secundario de frecuencia implementado.



Gráfica 1:Valores de Frecuencia de los generadores.

Observando la gráfica se aprecia también que el control secundario de frecuencia implementado no es todo lo preciso y eficiente que debería ya que quedan algunos puntos que mejorar, pero en lo que al algoritmo “Feasible Flow ” se refiere, este ha demostrado ser bastante exacto y preciso, a pesar del problema de “drop packets” causa de la comunicación inalámbrica.

5. Conclusiones

Las conclusiones y observaciones más destacables a las que se llega tras la realización de las simulaciones son:

1. La aproximación echa para determinar cuándo debe ejecutarse el algoritmo que permite recalcular los "set-points" de los generadores, que se basa en comparar los flujos de potencia a través de las líneas con sus valores límite, posiblemente no sea la más adecuada.

Dicha aproximación ha sido empleada con el objetivo de establecer un criterio para probar el funcionamiento y estructura del control, más que para establecer correctamente el punto exacto en el cual el algoritmo "Feasible Flow " este debe ejecutarse . Es por ello por lo que los resultados obtenidos nos son exactamente los ideales y se debe establecer un criterio más preciso, como el que se explica en el capítulo 3.

2. Otro posible fallo que se puede encontrar en cuanto a la ejecución del propio "Feasible Flow " son los ya mencionados "Drop packets".

3. Por último la falta de precisión que pudiese existir en el "Feasible Flow " puede ser también relacionada con el número de iteraciones necesario para asegurar la máxima precisión. El uso del algoritmo distribuido "*Max-Min Consensus*" permitiría encontrar el número óptimo de iteraciones estableciendo un equilibrio entre tiempo de ejecución y precisión.

Por otro lado, principales problemas que se han detectado tras operar el control distribuido han sido:

1. Un tiempo de reacción mucho más prolongado, debido a las iteraciones que son necesarias para el intercambio de información entre controladores, necesario para la implementación de este tipo de algoritmos.
2. El otro fallo que los sistemas distribuidos poseen frente a los centralizados es la precisión. Ya que esta es más pobre en los primeros, que en los segundos.
3. Problemas de sincronización entre los diferentes controladores.

Tras la finalización de este trabajo, la conclusión que se puede extraer es, que a pesar de que todavía queda un largo camino por recorrer para que la aplicación sistemas distribuidos para el control de microrredes puedan realmente ser instaurados en el

sistema real, parecen ser una alternativa viable y eficiente que permita la transición hacia un sistema más flexible y resiliente donde los sistemas de generación distribuidos y las renovables jueguen un papel fundamental. Y aunque es cierto que un control centralizado tiene ventajas en cuanto a la **precisión, velocidad de computación,** y **respuestas más suaves ante las variaciones**; gracias a la mejora en la resiliencia y flexibilidad del sistema que ofrecen los controles distribuidos dichos inconvenientes quedan compensados.

6. Referencias

[1] S. T. Cady, M. Zholbaryssov, A. D. Domínguez-García and C. N. Hadjicostis, "A Distributed Frequency Regulation Architecture for Islanded Inertialess AC Microgrids," in *IEEE Transactions on Control Systems Technology*, vol. 25, no. 6, pp. 1961-1977, Nov. 2017.

[2] S. T. Cady, A. D. Domínguez-García and C. N. Hadjicostis, "A Distributed Generation Control Architecture for Islanded AC Microgrids," in *IEEE Transactions on Control Systems Technology*, vol. 23, no. 5, pp. 1717-1735, Sept. 2015, doi: 10.1109/TCST.2014.2381601.

[3] Félix García y Alejandro Claderón Mateos."Diseños de Sitemas Distribuidos : Tolerancia a fallos" 2016-2017. https://www.arcos.inf.uc3m.es/infodsd/wp-content/uploads/sites/38/2017/02/3.pdf

[4] Nuñez Mata, Oscar, Diego Ortiz Villalba, Rodrigo Palma-Behnke, " *MICRORREDES EN LA RED ELÉCTRICA DEL FUTURO - CASO HUATACONDO*." Centro De Energía, Facultad De Ciencias Físicas y Matemáticas, Escuela De Ingeniería, Universidad De Chile.Santiago, Chile. Escuela De Ingeniería Eléctrica, Faculta De Ingeniería, Universidad De Costa Rica. San José, Costa Rica. Departamento Eléctrica - Electrónica, Escuela Politecnica Del Ejército. Latacunga, Ecuador., 28 Sept. 2013, file:///C:/Users/Patri/Downloads/15214-Texto%20del%20art%C3%ADculo-27849-1-10-20140707%20(1).pdf.

[5] O. Azofeifa et al., "Controller Hardware-in-the-Loop Testbed for Distributed Coordination and Control Architectures," 2019 North American Power Symposium (NAPS), Wichita, KS, USA, 2019, pp. 1-6, doi: 10.1109/NAPS46351.2019.8999980

[6] "Objetivos de desarrollo sostenible" ,Naciones unidas,15 Sep 2015, https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

[7] "¿Que es el hardware-in-the-loop?" ,National Instruments ,14 may 2020 https://www.ni.com/es-es/innovations/white-papers/17/what-is-hardware-in-the-loop-.html

[8] R. H. Lasseter, "MicroGrids: A Conceptual Solution," 2002 IEEE Power Eng. Soc. Winter Meet. Conf. Proc. (Cat.No.02CH37309), vol. 1, pp. 305–308, 2002.

[9] "Virtual HIL Device - Typhoon HIL." Abril-202, https://www.typhoon-hil.com/products/virtual-hil-device/.

SECONRARY FREQUENCY CONTROL ALGORITHM AC ISLANDED MICROGRIDS

Author: Samper Lario, Patricia.
Supervisor: Domíngez-García, Alejandro.
Collaborating Entity: Universidad de Illinois at Urbana Champaign.

ABSTRACT

This work consists on the implementation of a distributed control for secondary frequency control.

The bulk of the project focuses on implementing the "Feasible Flow" algorithm and evaluating its performance and efficiency for this type of network. For this purpose, a test bench designed to provide a highly reliable environment in real time will be used, for the evaluation of control architectures.

The proposed system architecture aims to bring frequency error to zero, which for non-inertial microgrids means ensuring that the frequency at each node is equal to the point of operation that results in a stable system.

Finally, a reflection is presented comparing centralized versus distributed control systems.

Keywords: Distributed, Control, Frequency, Power, Microgrid, Resilience, Algorithm, DER´s.

1. Introducción

The current electricity system is in need of transformation for several reasons, among which we can find : increased concern for the environment resulting in a growing use of renewable energy, increased energy demand and the electrification of rural areas, among others. All this is causing a trend towards decentralization of the electricity system. That is why microgrids are becoming essential to meet the objectives of energy renewal, where renewables have a major role.

In short, it can be said that the objective of this transformation is to achieve a more intelligent, efficient, and flexible consumption, where resources are optimized, and their implementation is facilitated.

At the University of Illinois, a project is underway to create a distributed control system for microgrids: "Controller Hardware-in-the-Loop Testbed for Distributed Coordination and Control Architectures", led by Professor Alejandro Domínguez-García.

This is an innovative work whose objective is to develop distributed decision systems that can provide great benefits for the control of these networks, improving the resilience and adaptability of the system.

The present End of Degree Project is firstly to present the bases of secondary frequency control in microgrids in islanded mode developed at the University of Illinois, thus establishing the reference framework of the algorithm for power flow control based on a distributed decision making system, on which this work is focused. The implementation of these algorithms has been carried out using the C++ programming language and using object-oriented code.

2. Project definition

In this work, we will first discuss the transformation that the electricity generation and distribution sector is undergoing, in particular a small introduction to the so-called "microgrids" and the new distributed generation systems, together with an analysis of the advantages and disadvantages of these new systems.

Next, a distinction is made between the different types of microgrids that can be found, as well as the different control systems that they can present.

Once raised in the general context of the project, the work is focused on establishing the basis of secondary frequency control for microgrids in isolated mode (developed at the University of Illinois), to finally arrive at the development of an algorithm for power flow control based on a distributed decision-making system. The implementation of these algorithms has been carried out using the C++ programming language and by means of an object-oriented code that makes possible the communication between the different elements of the network. The proposed control system aims to adjust the output points of the generators to obtain the "set points", so that the frequency error resulting from small disturbances in the load is eliminated.

These "set points" are chosen so that when these are applied to the , it is stable around these operating points and the frequency value is equal to a reference value.

This algorithm will be tested in three different models:

1. Three node system with centralized control.

2. Three node system with distributed control.

3. Simulation of secondary distributed frequency control in a micro-network with six nodes.

In the first two, only the algorithm for power flow control is implemented in a simple three-node system to demonstrate its effectiveness. Finally, a complete simulation of the secondary frequency control is made using a distributed control for a six-node microgrid.

Once this algorithm has been implemented, the results obtained will be analyzed and compared to reach a conclusion about the effectiveness of distributed decision systems versus traditional centralized decision systems.

3. Model description

First, the algorithm will be tested on a three-node system using Arduinos as controllers. But to evaluate the performance of the implemented algorithm, Typhoon HIL simulator will be used, which provides an effective, efficient, low-cost, scalable, and adjustable test environment for distributed coordination and control architectures.

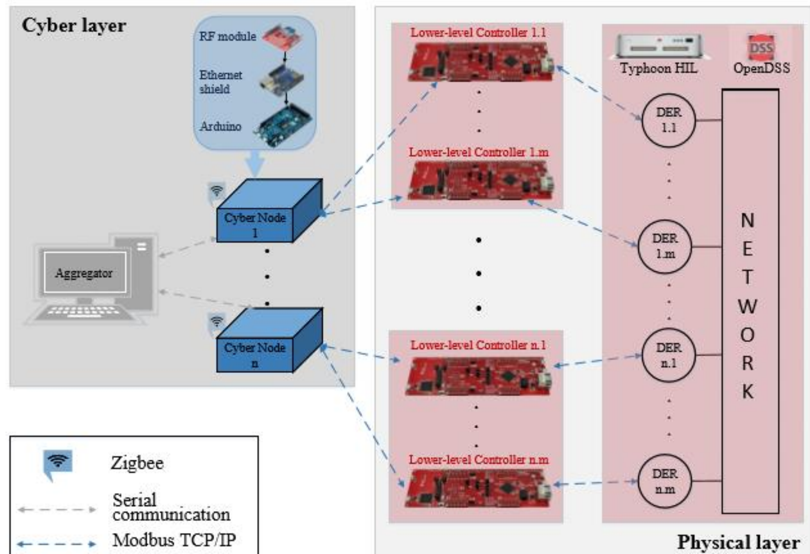


Figure 1: Schematic Hardware in the loop test bench.[1]

Specifically, the simulations will be carried out in a six-node microgrid (three generators and three loads) as shown in the following figure.

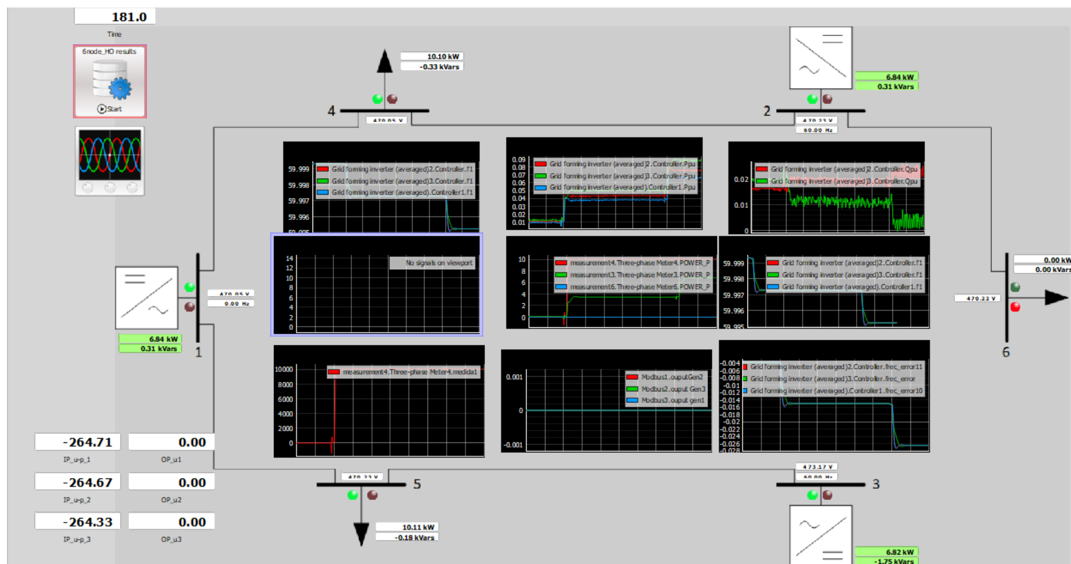


Figure 2: 6 bus microgrid (Typhoon HIL)

On the other hand, each node of the network is controlled by a controller which is

composed of an Arduino Due. Neighboring cybernetic nodes communicate wirelessly with each other through MaxStream XB24-DMCIT-250 revB XBee connected to the Arduino Due.

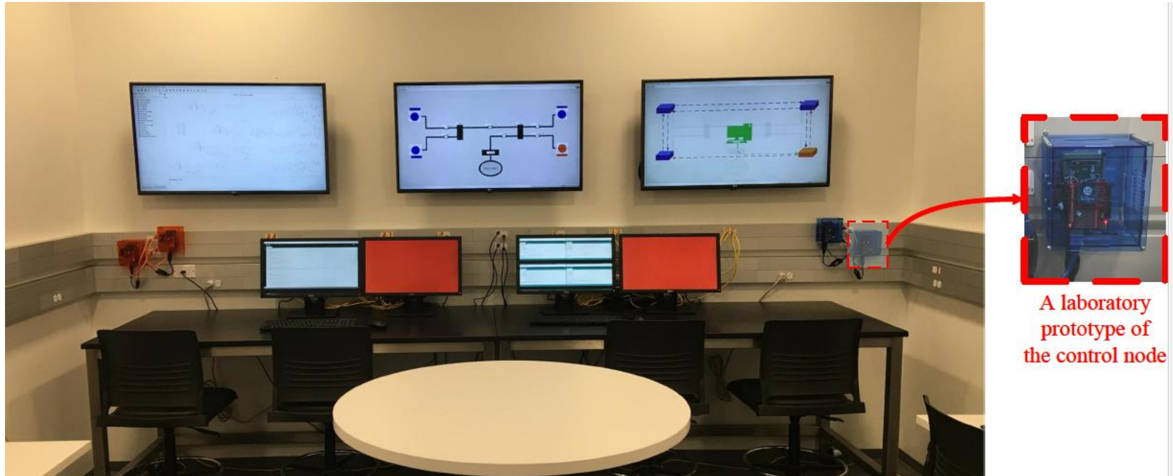


Figure 3: UIUC laboratory

4. Results

After the implementation of the "Feasible Flow" algorithm in the three-node systems, it was demonstrated the correct operation and efficiency of the algorithm to determine those individual generation values of the generators that supply the demand of the loads in a microgrid, without violating any of the generation restrictions or the limits of the maximum power that can circulate through each of the branches. However, a common failure in wireless communications known as "drop packets" was revealed, which causes some of the information packets sent not to reach the receiver. This problem can be fixed by a more complex algorithm. Currently, the engineers who are part of the University of Illinois project are working on algorithms that will allow the correct functioning of the system despite this problem.

The following images show the results obtained in this test for the distributed control example, establishing 40 as the number of iterations.

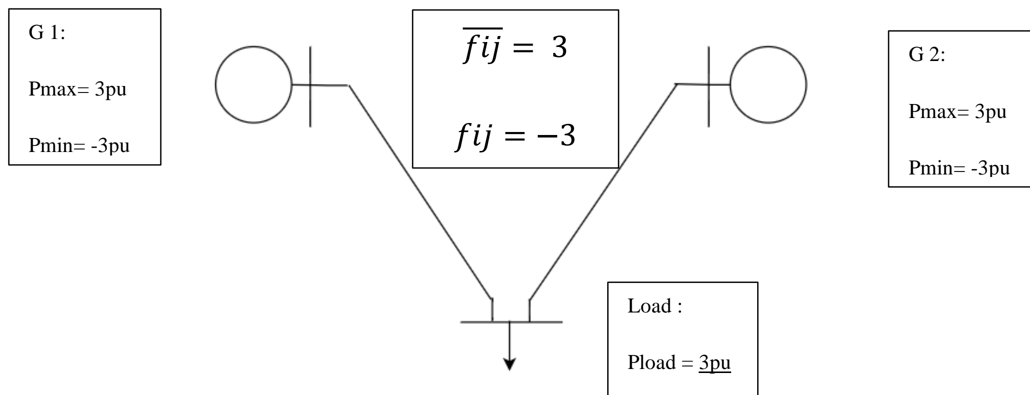


Figure 4: 3 bus System

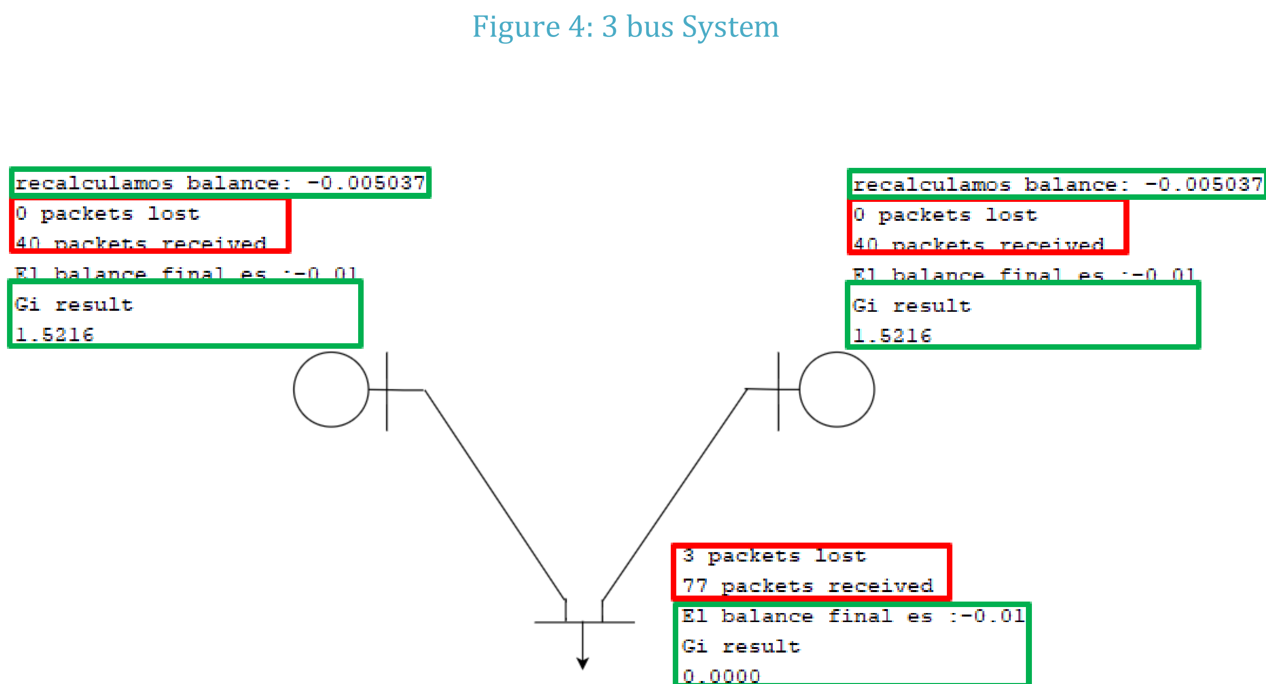
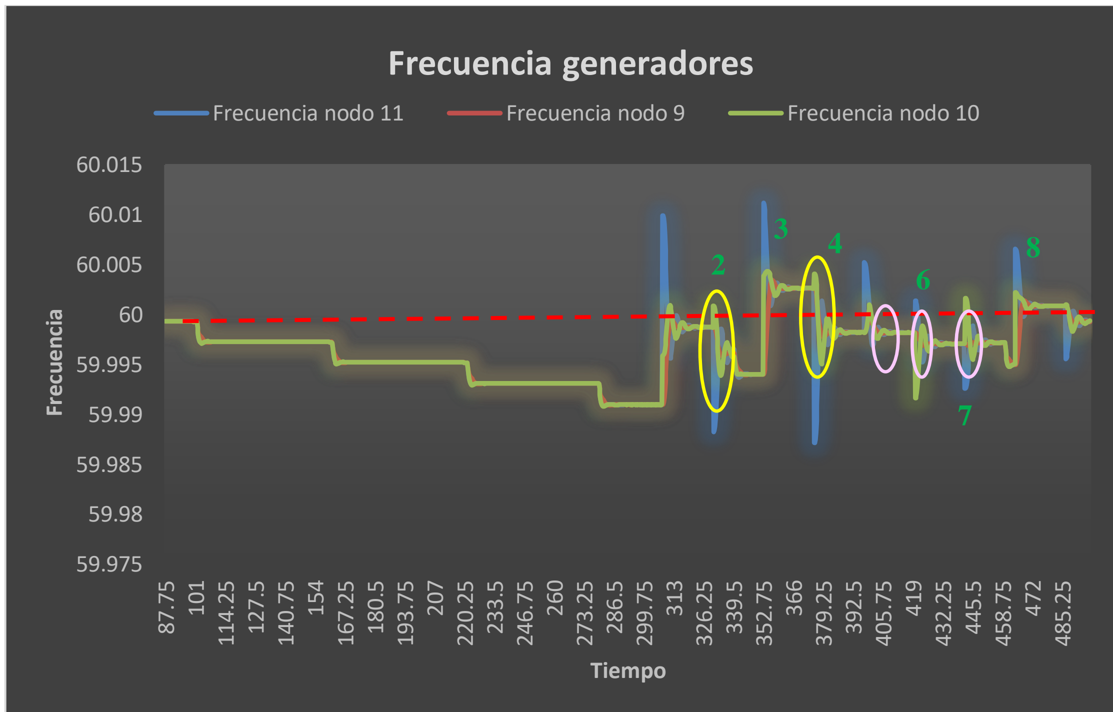


Figure 1:Arduinos results for the Trhee-node distributed control

As can be seen, despite the failure, the results are close enough to the expected results, that the generation of the two generators is practically 1.5 and the balance of the nodes is close to zero.

For the simulation of the microgrid the results obtained that can be appreciated in the following graph in which the frequency appears together with the evolution of the power demanded by the loads. It can be appreciated how the jumps in the frequency that try to approach is to its reference value are due to the secondary frequency control implemented.



Graph 1:Generators Frecuency values .

Observing the graph, it is also appreciated that the secondary frequency control is not as precise and efficient as it should be, since there are still some points to improve, but as far as the “Feasible Flow” algorithm is concerned, this has been quite accurate and precise , despite the problem of "drop packets" due to wireless communication.

5. Conclusions

The most noteworthy conclusions and observations reached after the simulations are:

1. The approximation made to determine when the algorithm for recalculating the "set-points" of the generators should be executed, which is based on comparing the power flows through the lines with their limit values, may not be the most appropriate.

This approach has been used with the objective of establishing a criterion for testing the operation and structure of the control, rather than correctly establishing the exact point at which the "Feasible Flow" algorithm should be executed. That is why the results obtained are not exactly ideal and a more precise criterion must be established, as explained in Chapter 3.

2. Another possible fault that can be found regarding the execution of the "Feasible Flow" itself is the already mentioned "Drop packets".

3. Finally, the lack of precision that could exist in the "Feasible Flow" can also be related to the number of iterations necessary to ensure maximum precision. The use of the distributed algorithm "Max-Min Consensus" would allow to find the optimal number of iterations establishing a balance between execution time and precision.

On the other hand, the main problems that we faced when operating distributed control have been:

1. The reaction time is much longer, due to the iterations that are necessary for the exchange of information between nodes, necessary for the implementation of this type of algorithms.

2. The other flaw that distributed systems present versus centralized systems is precision.

3. Synchronization problems between the different controls of each node.

After the completion of this work, the conclusion that can be drawn is that despite that there is still a long way to go before the application of distributed systems for the control of microgrids can be implemented in the real system, they seems to be a viable and efficient alternative that allows the transition towards a more flexible and resistant system where distributed generation systems and renewables play an important role. And while it is true that centralized control has advantages in terms of accuracy, computing speed, and smoother responses to variations, the improved reliability and flexibility of the system offered by distributed systems probably offset these drawbacks.

6. Referencias

[10] S. T. Cady, M. Zholbaryssov, A. D. Domínguez-García and C. N. Hadjicostis, "A Distributed Frequency Regulation Architecture for Islanded Inertialess AC Microgrids," in *IEEE Transactions on Control Systems Technology*, vol. 25, no. 6, pp. 1961-1977, Nov. 2017.

[11] S. T. Cady, A. D. Domínguez-García and C. N. Hadjicostis, "A Distributed Generation Control Architecture for Islanded AC Microgrids," in *IEEE Transactions on Control Systems Technology*, vol. 23, no. 5, pp. 1717-1735, Sept. 2015, doi: 10.1109/TCST.2014.2381601.

[12] Félix García y Alejandro Claderón Mateos."Diseños de Sitemas Distribuidos : Tolerancia a fallos" 2016-2017. https://www.arcos.inf.uc3m.es/infodsd/wp-content/uploads/sites/38/2017/02/3.pdf

[13] Nuñez Mata, Oscar, Diego Ortiz Villalba, Rodrigo Palma-Behnke, " *MICRORREDES EN LA RED ELÉCTRICA DEL FUTURO - CASO HUATACONDO*." Centro De Energía, Facultad De Ciencias Físicas y Matemáticas, Escuela De Ingeniería, Universidad De Chile.Santiago, Chile. Escuela De Ingeniería Eléctrica, Faculta De Ingeniería, Universidad De Costa Rica. San José, Costa Rica. Departamento Eléctrica - Electrónica, Escuela Politecnica Del Ejército. Latacunga, Ecuador., 28 Sept. 2013, file:///C:/Users/Patri/Downloads/15214-Texto%20del%20art%C3%ADculo-27849-1-10-20140707%20(1).pdf.

[14] O. Azofeifa et al., "Controller Hardware-in-the-Loop Testbed for Distributed Coordination and Control Architectures," 2019 North American Power Symposium (NAPS), Wichita, KS, USA, 2019, pp. 1-6, doi: 10.1109/NAPS46351.2019.8999980

[15] "Objetivos de desarrollo sostenible" ,Naciones unidas,15 Sep 2015, https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

[16] "¿Que es el hardware-in-the-loop?" ,National Instruments ,14 may 2020 https://www.ni.com/es-es/innovations/white-papers/17/what-is-hardware-in-the-loop-.html

[17] R. H. Lasseter, "MicroGrids: A Conceptual Solution," 2002 IEEE Power Eng. Soc. Winter Meet. Conf. Proc. (Cat.No.02CH37309), vol. 1, pp. 305–308, 2002.

[18] "Virtual HIL Device - Typhoon HIL." Abril-202, https://www.typhoon-hil.com/products/virtual-hil-device/.

Índice de la memoria

Capítulo 1. Introducción y planteamiento del proyecto. 1

1.1 Motivación del proyecto 2

1.2 Introducción al concepto de microrred. 3

1.3 Introdución a los diferentes sistemas de control 6

Capítulo 2. 9

2.1 Descripción de las tecnologías 9

2.2 Justificación 14

Capítulo 3. Control secundario de frecuencia 16

3.1 Control secundario de frecuencia. 16

3.2 Otros algorítmos distribuidos: 29

3.2.1 Ratio Consensus. 29

3.2.2 Max-Min Consensus. 32

3.3 Descripción del algoritmo “Feasible Flow”. 34

Capítulo 4. Desarrollo del proyecto 42

4.1 Paso 1: Decisión centralizada en red de 3 nodos. 42

4.2 Paso 2: Feasible Flow (implementación distribuida) 44

4.3 Explicación función principal: 52

4.4 Paso 3: Implementación para red de 3 nodos 60

4.5 Paso 4: Implementación En Typhoon Hill 62

Capítulo 5. Resultados. 77

5.1 Resultados sistema 3 Nodos distribuido 77

5.2 Resultados simulación en tiempo real 80

Capítulo 6. Análisis de Resultados 91

Capítulo 7. ODS 93

Capítulo 8. Planificación y Estimación Económica. 97

8.1 Coste del Proyecto....................98
8.2 Análisis de la fiabilidad....................99

Capítulo 9. Conclusiones y Trabajos Futuros....................105

Capítulo 10. Bibliografía....................107

ANEXO I 109

Índice de figuras

Ilustración 1-1:Red centralizada-Red distribuida (UMN.edu) 6
Ilustración 2-1: Simulador Typhhon HIL....................9
Ilustración 2-2: Simulador Typhoon HIL de la universidad de Illinois....................9
Ilustración 2-3:Arduino Board....................11
Ilustración 2-4:Ethernet Shield....................11
Ilustración 2-5:RF module....................12
Ilustración 2-6:Esquema del controlador del banco de pruebas Hardware in the loop.[1].13
Ilustración 2-7:Representación de las capas física y cibernética de la microrred [1]..........13
Ilustración 2-8:HIL laboratorio (UIUC)....................14
Ilustración 3-1:Esquema control secundario de frecuencia [1] 28
Ilustración 3-2:Feasible Flow(Flow chart)....................41
Ilustración 4-1:Sistema 3 nodos 43
Ilustración 4-2: FunciónFeasibleFlowAlgorithm_RLS 46
Ilustración 4-3:Función leaderfeasibleFlow_RSL....................47
Ilustración 4-4:función nonleaderfeasibleFlow_RSL....................47
Ilustración 4-5:Función broadcastBalanceFeasibleFlow 48
Ilustración 4-6:Función getBjFromPacket 49
Ilustración 4-7:Funcón getFloat32FromPacket 49
Ilustración 4-8:Propiedades OLocalVertex....................50
Ilustración 4-9:Métodos OLocalVertex 50
Ilustración 4-10 :Propiedades ORemoteVertex....................51
Ilustración 4-11:Métodos ORemoteVertex 51
Ilustración 4-12:(https://www.geeksforgeeks.org/linked-list-set-1-introduction/)..............51
Ilustración 4-13:Red 3 nodos....................60
Ilustración 4-14:Resultados "Feasible Flow" sistema de 3 nodos distribuido....................61
Ilustración 4-15:Sistema seis nodos....................62
Ilustración 4-16: Área de operación....................63
Ilustración 4-17:Modelo distribuido Typhoon 65

Ilustración 4-18:Modelo Centralizado Typhoon 66
Ilustración 4-19:Modelo generador Ilustración 4-20:Modelo Carga....67
Ilustración 4-21:Carga Typhoon....67
Ilustración 4-22:Medidor Typhoon....68
Ilustración 4-23:Controlador de uno de los nodos generadores 68
Ilustración 4-24:Controlador central sistema centralizado....69
Ilustración 4-25:Estructura código Arduino....70
Ilustración 4-26:Esquema control secundario de frecuencia....76
Ilustración 5-1:Red 3 nodos 77
Ilustración 5-2:Resultados Arduinos sistema distribuido de 3 nodos....77
Ilustración 5-3:Resultados Arduino(generador 1) 78
Ilustración 5-4:Resultados Arduino (generador 2) 78
Ilustración 5-5:Iteración 1del nodo de carga....79
Ilustración 5-6:Iteración 19 nodo de carga....79
Ilustración 5-7:Iteración 38 nodo de carga....79
Ilustración 5-8:HIL SCADA 80
Ilustración 5-9:Red con los números de identificación de cada nodo 80
Ilustración 5-10:Potencia demandada Vs frecuencia....86
Ilustración 8-1:Tabla presupuesto proyecto 98
Ilustración 8-2:Flujo de caja anuales 98
Ilustración 8-3:Curva de la bañera....100
Ilustración 8-4:Curva de fiabilidad(https://www.arcos.inf.uc3m.es/infodsd/wp-content/uploads/sites/38/2017/02/3.pdf) 100
Ilustración 8-5:Coste de desarrollo de una microrred....103
Ilustración 9-1:Modelo de la red Banshee....106

Índice de ecuaciones

Ecuación 3.1:Frecuencia Media.......18
Ecuación 3.2 : Diferencia de frecuencia 18
Ecuación 3.3:Condición sistema estable [1].......18
Ecuación 3.4 18
Ecuación 3.5 18
Ecuación 3.6 18
Ecuación 3.7:[1].......19
Ecuación 3.8:[1].......19
Ecuación 3.9:Matriz Laplaciana[1].......19
Ecuación 3.10: Dinámica nodos generadores [1] 20
Ecuación 3.11: Dinámica nodos demanda [1].......20
Ecuación 3.12 20
Ecuación 3.13: Error frecuencia [1].......22
Ecuación 3.14: Error medio de frecuencia(tiempo continuo) [1].......22
Ecuación 3.15:[1].......23
Ecuación 3.16:Error medio de frecuencia (tiempo discreto)[1] 23
Ecuación 3.17:[1].......23
Ecuación 3.18: Nuevo valor de salida del generador [1] 23
Ecuación 3.19:[1].......24
Ecuación 3.20: Cálculo ganancias [1].......24
Ecuación 3.21:[1].......24
Ecuación 3.22 25
Ecuación 3.23:[1].......25
Ecuación 3.24:[1].......25
Ecuación 3.25:[1].......26
Ecuación 3.26:[1].......26
Ecuación 3.27:[1].......26
Ecuación 3.28:[1].......26

Ecuación 3.29:[1]..29
Ecuación 3.30 29
Ecuación 3.31:[1]..29
Ecuación 3.32:[1]..30
Ecuación 3.33 31
Ecuación 3.34 31
Ecuación 3.35 32
Ecuación 3.36 32
Ecuación 3.37: Balance [1] 35
Ecuación 3.38: Flujo inicial..35
Ecuación 3.39: Generación Inicial..36
Ecuación 3.40: Cálculo Flujo [1]..36
Ecuación 3.41: Cálculo generación [1] 36
Ecuación 3.42 36
Ecuación 3.43: [1]...37
Ecuación 3.44:Comprobación límites flujos[1]..37
Ecuación 3.45:Comprobación límites generación [1]...38
Ecuación 3.46:[1]..38
Ecuación 3.47:[1]..38
Ecuación 8.1:Cálculo del valor actual del coste del proyecto 98
Ecuación 8.2:Fiabilidad sistema en paralelo 100
Ecuación 8.3: Fiabilidad sistema centralizado 101
Ecuación 8.4:Fiabilidad sistema distribuido 101

Índice de Gráficas y Tablas

Gráfica 3-1:Ejemplo simulación "Ratio Consensus" con tres nodos. 30
Gráfica 5-1: Black Start(Cargas) 81
Gráfica 5-2:Black Start(Generadores) 82
Gráfica 5-3: Black Start(Frecuencia) 82
Gráfica 5-4:Potencia demandada por las cargas 83
Gráfica 5-5:Flujo de potencia a través de las líneas de conexión 84
Gráfica 5-6:Potencia demandada frente a la generada 84
Gráfica 5-7:Frecuencia de los generadores 85
Gráfica 5-8:Error de Frecuencia 85
Gráfica 5-9:Potencia demandada. Simulación 2 87
Gráfica 5-10:Potencia generada. Simulación 2 87
Gráfica 5-11:Flujo de potencia activa a través de las líneas de conexión. Simulación 2 88
Gráfica 5-12:Valores de Frecuencia de los generadores. Simulación 2 88
Gráfica 5-13:Potencia Vs Frecuencia. Simulación 3 89
Gráfica 5-14:Potencia Vs Frecuencia. Simulación 3 90
Gráfica 5-15:Potencia Vs Frecuencia. Simulación 90

Capítulo 1. INTRODUCCIÓN Y PLANTEAMIENTO DEL PROYECTO.

Este trabajo comenzará con una breve introducción de la transformación que está experimentando el sector de la generación y distribución eléctrica, en particular se plantea una pequeña introducción a cerca de las conocidas como "microrredes" y los nuevos sistemas de generación distribuida, junto con un análisis sobre las ventajas y desventajas que tienen estos nuevos sistemas.

Seguidamente se realiza una distinción entre los diferentes tipos de microrredes que podemos encontrar, así como los diferentes sistemas de control que estas pueden presentar.

Una vez planteado en contexto general del proyecto, el trabajo se centra en establecer las bases del control secundario de frecuencia par microrredes en modo aislado, para finamente llegar al desarrollo de un algoritmo para el control del flujo de potencia basado en un sistema de toma de decisiones distribuido. La implementación de estos algoritmos se ha llevado a cabo empleando el lenguaje de programación C++ y mediante el uso de código orientado a objetos que hace posible la comunicación entre los diferentes elementos de la red.

En definitiva, se pretende diseñar un sistema para la regulación de frecuencia en microrredes que garantice la operación estable y por lo tanto que la frecuencia en cada nodo del sistema sea la deseada e igual al valor de referencia.

Finalmente, una vez se han implementado este algoritmo junto con otros algoritmos, se analizarán y compararán los resultados obtenidos para llegar a una conclusión acerca de la eficacia de los sistemas de decisión distribuida frente a los sistemas de toma de decisiones de forma centralizada.

1.1 MOTIVACIÓN DEL PROYECTO

El Sistema eléctrico tal y como lo conocemos está sufriendo una importante transformación. El futuro sistema eléctrico estará formado por tecnologías digitales fuentes renovables y redes inteligentes de generación distribuida. El alto crecimiento que están teniendo las fuentes de energía renovables y la concienciación acerca del cambio climático hacen necesaria una transición del antiguo modelo lineal del sistema eléctrico a uno nuevo que permita añadir fuentes renovables, conectar a los consumidores a la red y proporcionar un mayor control sobre el uso de energía.

Las microrredes funcionan como redes eléctricas independientes, con su propia generación y recursos de almacenamiento que pueden funcionar conectadas a la red principal o de forma aislada. Es evidente que las microrredes son el futuro y cuentan con un alto potencial de crecimiento, es por ello por lo que se debe trabajar en el desarrollo de estas para que su implantación sea posible en los próximos años.

En la Universidad de Illinois se está llevando a cabo un proyecto para la creación de un sistema de control distribuido para microrredes: "Controller Hardware-in-the-Loop Testbed for Distributed Coordination and Control Architectures", liderado por el profesor Alejandro Domínguez-García.

Se trata de un trabajo innovador cuyo objetivo es desarrollar sistemas de decisión distribuidos que pueden aportar grandes beneficios para el control de estas redes, mejorando la resiliencia y adaptabilidad del sistema.

Mientras que los sistemas centralizados son más sencillos de implementar y de mayor rapidez en la toma de decisiones; los sistemas distribuidos poseen la ventaja de no estar condicionados por el fallo de un único punto del sistema y en consecuencia son más flexibles y resilientes. Por lo que la investigación y desarrollo de este tipo de control pueden suponer una gran mejora para un sistema eléctrico basado en microrredes.

1.2 INTRODUCCIÓN AL CONCEPTO DE MICRORRED.

Empezaremos con una introducción al concepto de microrred y los distintos tipos que podemos encontrar, para luego centrarnos en las microrredes de tipo aislado en las cuales se centra el proyecto.

U.S. Department of Energy Microgrid Exchange Group, define las microredes de la sanguine forma: "A microgrid is a group of interconnected loads and distributed energy resources within clearly defined electrical boundaries that acts as a single controllable entity with respect to the grid. A microgrid can connect and disconnect from the grid to enable it to operate in both grid-connected or island-mode."

Por lo tanto, se pueden establecer las siguientes características principales que definen a las microrredes:

1. Agregación de elementos eléctricos de baja tensión.
2. Agrupados en una cierta área geográfica acotada
3. La microrred puede ser operada tanto conectadas a la red como autónomo (de forma aislada).

Para la comprensión de las microrredes es necesario introducir el término de **DER** (Distributed Energy Resources), se trata de pequeños generadores de potencia situados cerca de los consumidores finales.

Es por ello por lo que las microrredes son ideales para la integración de los DER en la red eléctrica ya que se trata de entidades autocontroladas que responden a señales de control, promoviendo un enfoque de control descentralizado.

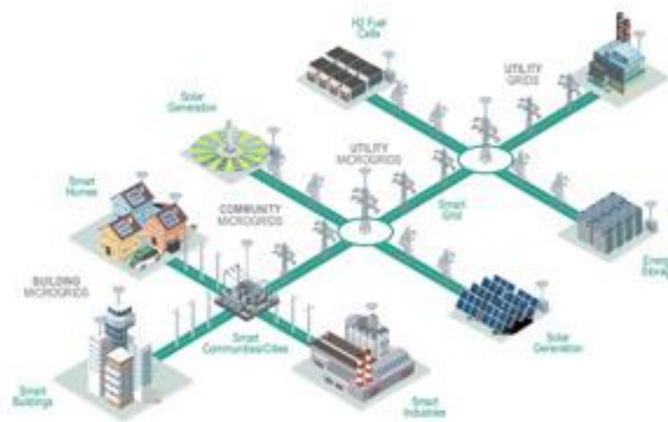


Tabla 1:Microrred (World Energy Trade, www.worldenergytrade.com/index.php/component/seoglossary/1-energia/microred?Itemid=113.)

Con la introducción de las microrredes se pretende solucionar algunos de los problemas que presentan los sistemas eléctricos tradicionales como pueden ser: las grandes pérdidas de energía debido a la trasmisión en largas distancias, participación estática por parte de los consumidores debido a la falta de comunicación entre estos y los operadores, así como la integración de la generación distribuida y la penetración de las energías renovables.

Estas microrredes pueden funcionar de dos modos:

-Conectada a la red principal.

-De forma aislada.

Cada uno de estos a su vez puede presentar dos tipos principales de sistemas de control.

En **modo conectado,** a efectos de la compañía eléctrica la microrred se ve como una sola carga. Es la red principal la que proporciona las referencias de tensión y frecuencia necesarias. De esta forma no existirán problemas en la estabilidad de la red.

En el caso de que la red sea controlada por un controlador central, una vez se haya resuelto el despacho económico, este envía a los controladores de las fuentes y cargas las consignas de potencia activa y reactiva. Posteriormente deberá comprobar que no se violan ninguna de las restricciones impuestas en la red.

Cuando la microrred funciona en **modo aislado**, los generadores deberán responder con rapidez a los cambios en la demanda para mantener la frecuencia y la tensión estables. En este caso por lo tanto se debe hacer frente a varios problemas.

En primer lugar, los generadores de la microrred no cuentan con la inercia necesaria para asumir los desequilibrios ocasionados por las diferencias puntuales entre generación y demanda, y por otro lado la respuesta de los elementos generadores de una microrred es bastante lenta lo que puede provocar problemas en la estabilización del sistema.

Debido a esto será necesario contar con sistemas de almacenamiento que compensen las diferencias entre generación y demanda.

En el caso de un control centralizado, este deberá proporcionar las referencias de frecuencia y tensión al resto de elementos de generación. Para funcionar así, los sistemas de almacenamiento deberán estar conectados a la microrred a través de un inversor con controles adecuados para mantener la estabilidad en tensión y frecuencia de la microrred.

Este ha sido por lo general el sistema de control empleado por las microrredes de forma que es un controlador principal el que suministra las consignas de los generadores del resto del sistema. Sin embargo, este proyecto se centra en las microrredes en isla(aisladas) y además implementaremos un sistema distribuido para de toma de decisiones. Más en concreto, en el control secundario de frecuencia.

Entre los varios problemas que supone el control de frecuencia, el control de frecuencia secundario es clave. El control secundario es el que sen encargará de que la frecuencia

de todo el sistema recupere su valor nominal tras un cambio en el punto de operación de la red.

Como se ha mencionado, el proyecto abarca el control de frecuencia secundario mediante un control distribuido, pero también se explica y hacen referencias al control de frecuencia mediante un sistema centralizado con el objetivo de hacer una comparativa.

Ambos sistemas presentan sus propias limitaciones: En el caso del enfoque centralizado para la toma de decisiones es susceptible a el fallo de un solo punto, mientras que el enfoque descentralizado presenta problemas en la precisión y rapidez de actuación. Sin embargo actualmente no existen demasiados estudios que cuantifiquen y comparen los rendimientos de los controles basados en la toma de decisiones distribuidas con los centralizados para el control secundario en microrredes .

1.3 INTRODUCIÓN A LOS DIFERENTES SISTEMAS DE CONTROL

En primer lugar, presentar los dp tipos de arquitecturas que podemos encontrar: ***Centralizados, Distribuido.***

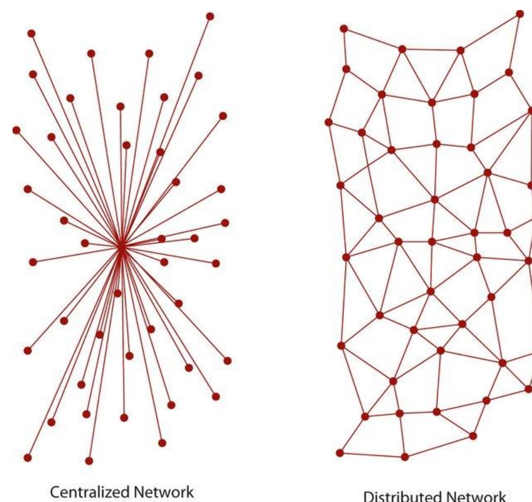


Ilustración 1-1:Red centralizada-Red distribuida (UMN.edu)

CENTRALIZADOS:

En este caso es un único controlador central que es el que envía la información al resto, es decir el reparto de energía y regulación de tensión son controlados por un único elemento central y las ordenes son enviadas al resto de elementos de la red.

Su operatividad y sencilla implementación constituyen sus principales ventajas. Sin embargo, debido a esta estructura centralizada están sujetos al fallo de un solo elemento de la red.

DISTRIBUIDOS:

"Un sistema de computación distribuido consta de múltiples procesadores autónomos que no comparten memoria principal, sino que cooperan a través de una red de comunicaciones que los interconecta."

Los sistemas centralizados tienen un único punto de fallo, el ordenador central que se encarga de la toma de decisiones. Por el contrario, los sistemas distribuidos están compuesto por varios elementos con probabilidades de fallo independientes, de tal manera que permite el funcionamiento del sistema de forma continuada.

Por ello es por lo que la mayor ventaja de los sistemas distribuidos es la fiabilidad, ya que el funcionamiento del sistema no está ligado a un solo elemento, sino que cualquier equipo puede suplir a otro en caso de que uno se estropee. La forma más fácil de llegar a esto es mediante la redundancia, es decir que la información no debe estar almacenada en un solo elemento, sino en el conjunto de ellas.

Este tipo de sistemas tiene numerosas aplicaciones en diferentes campos: bases de datos distribuidas, fabricación automatizada, supervisión remota y control, pero la que nos interesa a nosotros es la *"toma de decisiones coordinada "* donde todos los procesadores del sistema participan de forma coordinada en la toma de decisiones.

Comunicación y sincronización :

La principal diferencia entre los sistemas centralizados y distribuidos, como ya hemos visto se da en la comunicación de procesos. Por un lado, en los sistemas centralizados las iteraciones se hacen sobre memoria compartida mientras que en los sistemas distribuidos serán necesario el intercambio de mensajes.

En el caso de fallos en un sistema distribuido, cuando un nodo falla todos los nodos se deberán adaptar a las nuevas condiciones para poder seguir funcionando.

En este proyecto se explicarán tres algoritmos distribuidos:

Dos de ellos de tipo *"consensus*" (ya desarrollados por loa ingenieros del proyecto de la Universidad de Illinois) y el tercero en el cual se centra este trabajo es el algoritmo que permite el cálculo de los valores de salida de los generadores de manera que se cumpla con la demanda sin violar los límites de generación o flujos por las ramas como ya se mencionó en la introducción (“*Feasible Flow*”), en el cual los diferentes controladores deben cooperar y actuar como un solo grupo para alcanzar los objetivos comunes.

Capítulo 2.

2.1 *DESCRIPCIÓN DE LAS TECNOLOGÍAS*

En este capítulo describiremos el control **hardware in the loop** (HIL) que proporciona un entorno en tiempo real de alta fidelidad para probar arquitecturas de coordinación y control de recursos energéticos distribuidos. Este sistema es que usaremos para probar y ensayar los diferentes algoritmos que se emplearan y de esta manera poder analizar y validar su efecto y comportamiento.

Ilustración 2-1: Simulador Typhhon HIL

Ilustración 2-2: Simulador Typhoon HIL de la universidad de Illinois

HIL viene de las siglas en inglés Harware in The Loop. Las pruebas HIL son una técnica en la que las señales reales de un controlador son conectadas a un sistema de pruebas que simula la realidad, engañando al controlador para que piense que está en el producto ensamblado. La prueba y la iteración del diseño se realiza como si se estuviera utilizando el sistema del mundo real. De esta manera se pueden ejecutar miles de escenarios para poner en práctica el controlador sin el coste y el tiempo asociados a las pruebas físicas.

El banco de pruebas C-HIL proporciona un entorno de prueba eficaz, eficiente, de bajo coste, escalable y ajustable para arquitecturas de coordinación y control distribuidas.

Una microrred puede se puede caracterizar en dos capas, por un lado, la **capa física** compuesta por la infraestructura eléctrica utilizada para la generación y distribución de energía eléctrica y por otro la **capa cibernética** que comprende el hardware y software para la comunicación y el control.

Modelos de alta fidelidad de varios componentes de la capa física (DER´s líneas de distribución y cargas) se implementarán en dispositivo de emulación. Y los dispositivos de control de hardware (nodos cibernéticos) se conectarán al dispositivo en tiempo real. Los algoritmos y protocolos asociados con la coordinación distribuida y la arquitectura de control son implementados en cada nodo cibernético.

- Arquitectura para la coordinación de los DER´s :

  El objetivo de esta arquitectura es coordinar y controlar un grupo de DER´s para que de forma colectiva proporcionen servicios de control de frecuencia

A. Infraestructura de la capa cibernética.

   Se compone de *n* dispositivos de hardware a los que nos referimos como ciber nodos y una entidad virtual a la que nos referimos como agregador. Los agregadores sirven como intermediarios entre los ciber nodos y una entidad de nivel superior.

Estos elementos de la capa cibernética implementan los algoritmos y protocolos que realizan la coordinación y arquitectura de control distribuida mencionada anteriormente.

Cada ciber nodo está compuesto por un **Arduino Due**. Los nodos cibernéticos vecinos se comunican entres ellos sin cables mediante MaxStream XB24-DMCIT-250 revB XBee interconectado al Arduino Due.

Los Arduinos se componen de tres capas:

1. **Arduino Board:**

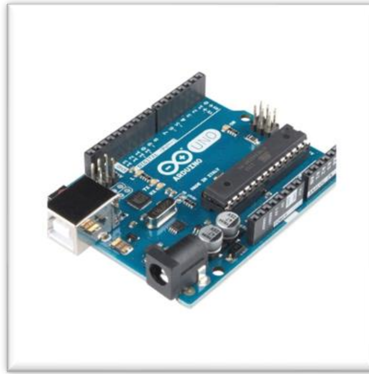


Ilustración 2-3:Arduino Board

2. **Ethernet Shield:**

Ilustración 2-4:Ethernet Shield

Este se encarga de la comunicación entre el Arduino y el ordenador. Su función es cargar el código C++ desarrollado en los Arduinos y posteriormente enviar los resultados de nuevo al ordenador.

3. **RF module**

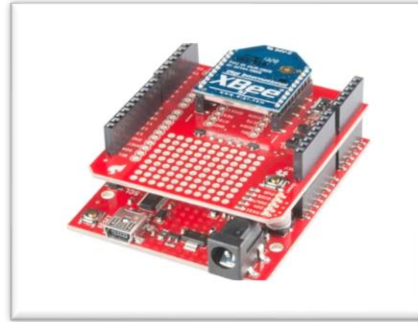


Ilustración 2-5:RF module

El módulo XBee permite la comunicación inalámbrica entre los Arduinos.

B. Infraestructura de la capa física.

En el banco de pruebas C-HIL, la capa física es emulada usando un simulador en tiempo real **Typhoon HIL.**

Como ya hemos mencionado este banco de pruebas está equipado con dispositivos de simulación de alta fidelidad en tiempo real.

Además, implementa modelos muy detallados de elementos del sistema, estos dispositivos emulan de forma muy precisa los efectos de conmutación transitorios y transitorios en el sistema de energía eléctrica.

Modelos orden reducido son también implementados en Thypoon HIL para reducir la complejidad de modelado y en consecuencia reducir el coste de computación de emular un gran número de DER´s.

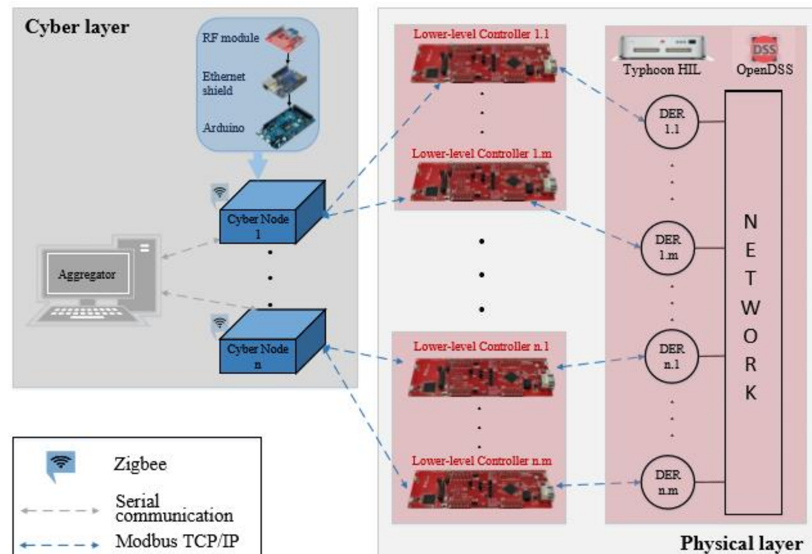


Ilustración 2-6:Esquema del controlador del banco de pruebas Hardware in the loop.[1]

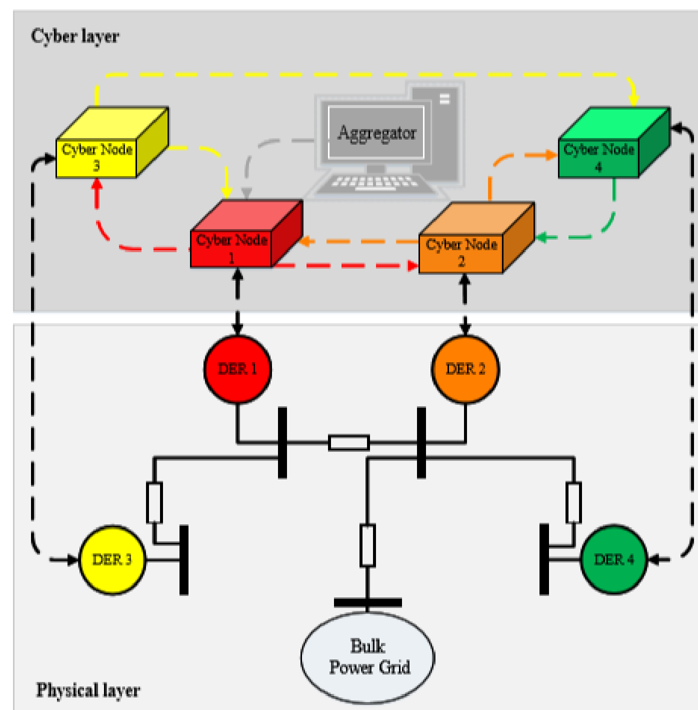


Ilustración 2-7:Representación de las capas física y cibernética de la microrred [1]

En la siguiente figura se muestra el laboratorio del que dispone la Universidad de Illinois para llevar acabo su proyecto de control de microrredes, y del que se ha hecho uso también para este trabajo.

> Cada ciber nodo está compuesto por un **Arduino Due**. Los nodos cibernéticos vecinos se comunican entres ellos sin cables mediante MaxStream XB24-DMCIT-250 revB XBee interconectado al Arduino Due.

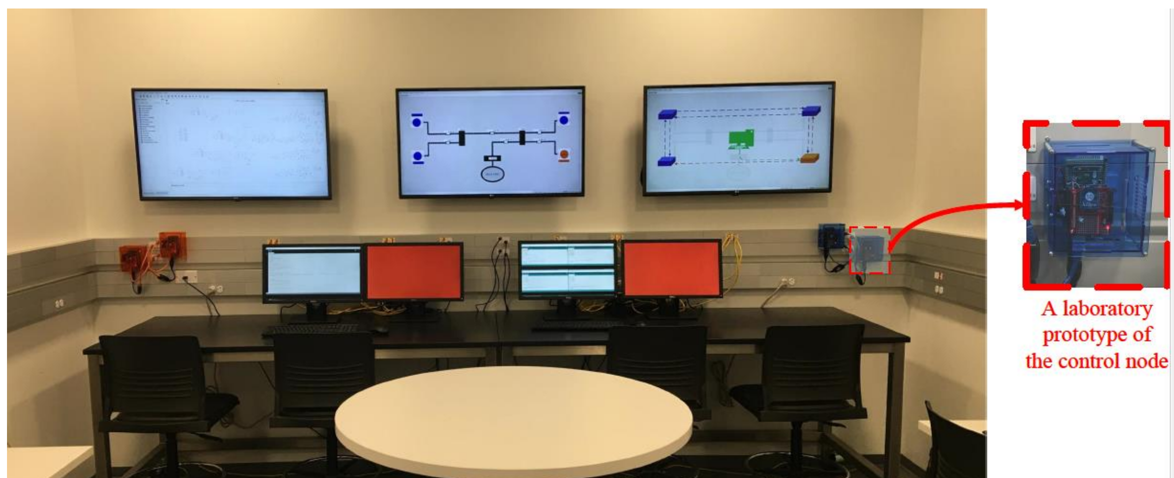


Ilustración 2-8:HIL laboratorio (UIUC)

2.2 JUSTIFICACIÓN

Como se ha mencionado el desarrollo de las microrredes parece ser muy prometedor para la integración de los DER´s (Distributed Energy Resources) además de contribuir a la mejora de la eficiencia, fiabilidad y adaptabilidad. Es por ello por lo que la investigación y el desarrollo de estas es de gran importancia dada la gran repercusión que puede tener en el futuro para la mejora del sistema eléctrico.

Por otro lado, este trabajo pretende emplear un sistema para la toma de decisiones más complejo y novedoso (sistema distribuido) que puede aportar grandes beneficios para el control de estas redes, especialmente para la mejora de la resiliencia y adaptabilidad del sistema. Además, actualmente no se han realizado muchos trabajos previos que cuantifiquen y comparen el rendimiento de los controles de microrredes basados en el

enfoque de toma de decisiones distribuidas con los centralizados especialmente en el contexto del control de frecuencia secundaria. Sin embargo, el proyecto que se está llevando a cabo por la universidad de Illinois aporta una visión interesante acerca de este tema.

En este trabajo se realiza un pequeño recorrido por los diferentes tipos de microrredes que podemos encontrar, así como los diferentes sistemas de control que pueden presentar, para posteriormente centrarse en el control de frecuencia secundario para **microrredes AC** en **isla** y **sin inercia** mediante un sistema de **control distribuido**.

El objetivo final del proyecto del que este trabajo forma parte es la implementación del control secundario de frecuencia para su implementación distribuida. El grueso del proyecto se centra en implementar el algoritmo "Feasible Flow" y evaluar su rendimiento y eficacia para este tipo de redes. Para ello se empleará un banco de pruebas que ha sido diseñado para proveer un entorno de alta fiabilidad en tiempo real, para la evaluación de arquitecturas de control.

La arquitectura del Sistema que se propone tiene como objetivo llevar el error de frecuencia a cero, que para las microrredes sin inercia (aquellos que se componen de generadores conectados por electrónica de potencia) supone asegurar que la frecuencia en cada nodo es igual al punto de operación que resulta en un sistema estable.

Finalmente se presenta una reflexión comparando los sistemas de control centralizados frente a los distribuidos.

Capítulo 3. CONTROL SECUNDARIO DE FRECUENCIA

3.1 CONTROL SECUNDARIO DE FRECUENCIA.

En este apartado se describe el modelo de microrred con que se trabaja, así como se ofrece un resumen acerca del problema de control de frecuencia secundaria desarrollada y presentada en [1]. Para finalmente explicar cómo se aborda la solución de este problema mediante un sistema distribuido.

Los controles de control secundario de frecuencia se prueban en un simulador en tiempo real de una microrred trifásica con los siguientes supuestos.

1) Las tres fases están equilibradas.

2.) Las líneas de trasporte no tienen pérdidas

3.) Los DER´s y cargas están interconectadas mediante inversores de fuente de voltaje.

4.) Todas las magnitudes están expresadas en magnitudes unitarias y los valores de cada DER y carga sirven como base del voltaje.

5.) La dinámica de cada de voltaje es mucho más rápida que el control de caída de frecuencia.

6.) La potencia reactiva del inversor es suficiente para soportar el control de voltaje.

CRITERIO DE COHESIÓN DE FASE:

El criterio de **"Phase-cohesive"** en el contexto de flujo de potencia en microrredes equivale a asegurar que la diferencia entre los ángulos en régimen permanente de cada par de nodos conectados será estrictamente menor que π/2. Además, si se cumple la condición de cohesión de fase la frecuencia natural de los osciladores y el acoplamiento entre ellos es tal que el sistema exhibe un comportamiento coherente, es decir que el ángulo de cada oscilador evoluciona a la misma velocidad.

Como se ha explicado anteriormente el sistema de control que se propone tiene como objetivo ajustar los puntos de salida de los generadores para obtener los "sert points", de manera que se elimine el error de frecuencia consecuencia de pequeñas perturbaciones en la carga.

Estos "set points " predeterminados se eligen de manera que cuando se aplican el sistema, este es estable alrededor de estos puntos de operación y el valor de frecuencia es igual a un valor de referencia.

A continuación, veremos las propiedades que caracterizan estos puntos de operación y se introducirá el problema de la elección de dichos puntos que cumplen las características mencionadas.

Los "set points" se denotan como: $u_i^*, i \in v_p^{(g)}$ (u_i es el valor de la potencia activa generada por el generador i), las cargas iniciales del sistema $l_i^0, i \in v_p^{(l)}$ y dejemos que $\theta^* = [\theta_1^*, \theta_2^*, \dots \theta_n^*]^T$ denote el punto de operación del sistema en régimen permanente.

De manera que para un valor de referencia de la frecuencia w_0 queremos que los u_i^* sean tales que, al aplicarlos a los generadores, exista el punto de operación θ_i^* y se caracterice por las siguientes propiedades.

I. La frecuencia media del sistema es igual al valor de referencia.

$$\left(\sum_{i\in v_p} \boldsymbol{\theta_i}\right)/\boldsymbol{n} = \boldsymbol{w_0}$$

Ecuación 3.1:Frecuencia Media

II. La frecuencia en cada nodo es igual.

$$\left|\dot{\theta}_i - \dot{\theta}_j\right| = 0, \, para\ todo\ i,j \in v_p$$

Ecuación 3.2 : Diferencia de frecuencia

III. El Sistema es estable alrededor del punto de operación θ*.

$$-\nabla_{\theta(t)} h(\theta(t))\Big|_{\theta=\theta^*} \preceq 0, \text{ where } \theta := [\theta_1(t), \ldots, \theta_n(t)]^{\mathrm{T}}$$

Ecuación 3.3:Condición sistema estable [1]

Donde:

$$h(\theta(t)) = h_e(\theta(t))$$

Ecuación 3.4

$$h_e(\theta(t)) \coloneqq B_i j * sin(\theta_i - \theta_j),$$

$$\boldsymbol{para\ cada\ \ e} = \{\boldsymbol{i},\boldsymbol{j}\} \in \boldsymbol{\varepsilon_p}$$

Ecuación 3.5

La primera propiedad se consigue eligiendo unos u_i^* , de forma que toda la potencia generada iguale a la potencia demandada. Si esto ocurre el sistema será “*phase cohesive”* (y por lo tanto cumplirá también las otras dos propiedades) cuando:

$$\left|\boldsymbol{\theta_i^*} - \boldsymbol{\theta_j^*}\right| \leq \boldsymbol{\varphi}\ \ \boldsymbol{para}\ \{\boldsymbol{i},\boldsymbol{j}\} \in \boldsymbol{\varepsilon_p}\ \boldsymbol{y}\ \boldsymbol{\varphi} \in [\boldsymbol{0},\boldsymbol{\pi}/\boldsymbol{2}]$$

Ecuación 3.6

Todo esto puede resumirse de la siguiente forma:

Encontrar los valores u_i sujetos a:

$$\sum_{i \in \mathcal{V}_p^{(g)}} u_i = \sum_{i \in \mathcal{V}_p^{(\ell)}} \ell_i^0,$$
$$\left\| M^{\mathrm{T}} L^{\dagger} \begin{bmatrix} u \\ -l^0 \end{bmatrix} \right\|_{\infty} \leq \sin(\phi),$$
$$\underline{u}_i \leq u_i \leq \overline{u}_i, \ \forall i \in \mathcal{V}_p^{(g)}.$$

Ecuación 3.7:[1]

Donde para cualquier ángulo $\phi \in [0, \pi/2)$:

$$\left\| M^{\mathrm{T}} L^{\dagger} \begin{bmatrix} u^* \\ -\ell^0 \end{bmatrix} \right\|_{\infty} \leq \sin(\phi),$$

Ecuación 3.8:[1]

Siendo **M** la matriz de incidencia y **L** la matriz Laplaciana ponderada y L† su pseudo inversa.

$$L = M \mathrm{diag}(\{B_{ij} : \{i, j\} \in \mathcal{E}_p\}) M^{\mathrm{T}}$$

Ecuación 3.9:Matriz Laplaciana[1]

La elección de los puntos de generación que cumplen con el criterio de **cohesión de fase** combinado con un **control de error de frecuencia** es suficiente para asegurar la estabilidad del sistema y que todo el sistema opere a la misma frecuencia.

Ahora se explicará todo el proceso de control secundario de frecuencia detalladamente.

La dinámica de la microrred se describe mediante las siguientes fórmulas:

En el caso de los buses de generación tenemos:

$$D_i \frac{d\theta_i(t)}{dt} = u_i(t) - \sum_{j \in \mathcal{N}_p(i)} B_{ij} \sin(\theta_i(t) - \theta_j(t)),$$
$$\underline{u}_i \leq u_i(t) \leq \overline{u}_i,$$

Ecuación 3.10: Dinámica nodos generadores [1]

En aquellos buses que tengan conectados una carga:

$$D_i \frac{d\theta_i(t)}{dt} = -(\ell_i^0 + \Delta\ell_i(t)) - \sum_{j \in \mathcal{N}_p(i)} B_{ij} \sin(\theta_i(t) - \theta_j(t)),$$

Ecuación 3.11: Dinámica nodos demanda [1]

Donde l_i^0 se refiere a la perturbación en la carga y li^0 hace referencia a la demanda nominal de potencia activa.

Estas dos ecuaciones describen la dinámica de la microrred. El objetivo de un sistema de control de frecuencia será satisfacer las siguientes condiciones:

- **O1.**Encontrar $ui *, i \in \upsilon_p^{(g)}$ de forma que:

$$1\ .\textstyle\sum_{i \in \upsilon_p(g)} \boldsymbol{u_i^*} = \sum_{i \in \upsilon_p(l)} \boldsymbol{l_i^0}$$

Ecuación 3.12

2. Para $i \in \upsilon_p(g)$ existe un punto de equilibrio u_i^* que satisface el criterio de fase cohesiva.

Esto quiere decir que el valor de salida de los generadores debe ajustarse de tal manera que se cumpla con la potencia demandada, sin sobrepasar los límites de generación de estos y por otro lado para todos los nodos debe existir un punto de equilibrio que haga que el sistema sea " *phase cohesive* ".

- **O2.** Para cambios suficientemente pequeños en la carga l_i^0 , hay que regular el valor de $u_i(t)$ alrededor de ui*, de manera que:

$$d\theta i/dt = 0 \; cuando \; t \rightarrow 0$$

Partiendo de unos valores iniciales para la demanda de potencia activa se calculan aquellos "set points" (u_i^*) que resultan en un estado operativo en cohesión de fase. Tras producirse pequeñas perturbaciones en la potencia demandada el sistema deberá ajustar los valores de salida de los generadores alrededor de estos puntos de operación, de forma iterativa para llevar el error de frecuencia a cero, a la vez que asegure puntos de operación que estén en cohesión de fase.

De esta manera el sistema asegura que ante pequeñas perturbaciones en la demanda el sistema seguirá siendo estable en lazo cerrado además de garantizar que la frecuencia en cada nodo sea la misma.

Los valores de salida de los generadores por tanto se irán ajustando, alejándose cada vez más de los "set points" iniciales. Estos pequeños cambios sirven para llevar la diferencia de frecuencia a cero, pero poco a poco irán llevando al sistema lejos del estado de operación en cohesión de fase. Es por ello por lo que tras varias pequeñas

perturbaciones en la carga o una perturbación mayor será necesario recalcular los "set points" que cumplan con la primera condición(**O1**)

El método para recalcular estos nuevos puntos se basa en una estimación de la cantidad por la cual el sistema se ha desviado del estado de operación en coherencia de fase.

CONTROL CENTRALIZADO SECUNDARIO DE FRECUENCIA:

En este caso un nodo central que puede comunicarse bidireccionalmente con los DER´s y las cargas de la microrred, así como almacenar información. Este mandará las instrucciones a los DER´s . En el caso de las cargas, dado que estas no se pueden controlar será suficiente con una conexión unidireccional entre estas y el sistema central.

El error de frecuencia medio se define de la siguiente forma:

$$\Delta\overline{\omega}(t) := \frac{\sum_{i=1}^{n} D_i \frac{d\theta_i(t)}{dt}}{\sum_{i=1}^{n} D_i},$$

Ecuación 3.13: Error frecuencia [1]

$$\Delta\overline{\omega}(t) = \frac{1}{\sum_{i=1}^{n} D_i}\left(\sum_{i\in\mathcal{V}_p^{(g)}} u_i(t) - \sum_{i\in\mathcal{V}_p^{(\ell)}} (\ell_i^0 + \Delta\ell_i(t))\right).$$

Ecuación 3.14: Error medio de frecuencia(tiempo continuo) [1]

El nodo central podrá calcular fácilmente este error de frecuencia si tiene acceso a la información de inyecciones de potencia en nos nodos DER´s y las cargas conectadas.

Una vez tenga este valor mediante un control PI se lleva el valor de este gradualmente hasta cero ajustando los "set-points" de los generadores.

Para hacer esto se discretiza el tiempo en rondas r , cada una de las cuales tiene una duración T_0.

$$u_i[r] := u_i(t), \quad t_r \leq t < t_{r+1},$$

Ecuación 3.15:[1]

Con $tr + 1 - tr = T0$.

De esta forma se puede expresar el error de frecuencia medio de esta manera:

$$\Delta\overline{\omega}[r] = \overline{D}\left(\sum_{i \in \mathcal{V}_p^{(g)}} u_i[r] - \sum_{i \in \mathcal{V}_p^{(\ell)}} (\ell_i^0 + \Delta\ell_i) \right)$$

Ecuación 3.16:Error medio de frecuencia (tiempo discreto)[1]

Con el objetivo de llevar el error de frecuencia a cero se ajustarán los “set poits “ de los generadores de la siguiente forma:

$$\lim_{r \to \infty} \sum_{i \in \mathcal{V}_p^{(g)}} u_i[r] = \sum_{i \in \mathcal{V}_p^{(\ell)}} \ell_i^0 + \Delta\ell_i.$$

Ecuación 3.17:[1]

De manera que los valores de potencia activa de los generadores se ajustarán alrededor de los “Phase choesive set points” predeterminados inicialmente, y siguiendo las ecuaciones siguientes:

$$u_i[r] = u_i^* + \Delta u_i[r],$$

Ecuación 3.18: Nuevo valor de salida del generador [1]

$$e_i[r+1] = e_i[r] + \kappa_i \Delta\overline{\omega}[r],$$
$$u_i[r] = u_i^* + \alpha_i e_i[r],$$

Ecuación 3.19:[1]

Donde $\boldsymbol{ei}[\mathbf{0}] = \mathbf{0}$ y $\boldsymbol{\alpha i}$ y $\boldsymbol{ki}$, son constantes que se apropiadamente calculadas.

Las ecuaciones que reúnen la condiciones para el cálculo de las ganancias que garantiza la estabilidad del sistema en lazo cerrado, son las presentadas a continuación, sin embargo, no se entrará en detalle ya que se ale de los objetivos de este trabajo.

$$\begin{bmatrix} \Delta\overline{\omega}[r+1] \\ e[r+1] \end{bmatrix} = \underbrace{\begin{bmatrix} \beta & \overline{D}\alpha^{\mathrm{T}} \\ \kappa & I_m \end{bmatrix}}_{:=\Phi} \begin{bmatrix} \Delta\overline{\omega}[r] \\ e[r] \end{bmatrix} + \begin{bmatrix} -1 \\ 0_m \end{bmatrix} \overline{D} \sum_{i \in \mathcal{V}_p^{(\ell)}} \Delta\ell_i,$$

Ecuación 3.20: Cálculo ganancias [1]

<u>ESTIMACIÓN DE LA DERIVACIÓN DEL PUNTO DE OPERACIÓN DE COHESIÓN DE FASE.</u>

Como se ha mencionado el restablecimiento de los nuevos "set points" se hace en base a una estimación de cuanto se ha desviado el sistema respecto al punto de estabilidad. A continuación, se explicará cómo se determina que el sistema se ha alejado de estos valores lo suficiente como para que sea necesario recalcular los "set points".

En primer lugar, definimos la función:

$$h^p(u,\ell) := \left\| M^{\mathrm{T}} L^{\dagger} \begin{bmatrix} u \\ -\ell \end{bmatrix} \right\|_{\infty},$$

Ecuación 3.21:[1]

Por lo que los puntos para los cuales el sistema será “phase cohesive” son aquellos donde la función anterior tome un valor menor que el $sin(\varphi)$ para un valor de $\varphi \in [0, \pi/2)$.

Se definen los vectores: $\Delta u \coloneqq [\Delta u_1, \Delta u_2, \dots \Delta u_m]^T$ y $\Delta l \coloneqq [\Delta l_{m+1}, \Delta l_{m+2}, \dots \Delta l_n]^T$ como los vectores que representan la cantidad por la cual los generadores y cargas han derivado de los valores de los “set points” iniciales. Para asegurar que el sistema sigue operando en cohesión de fase mientras evoluciona alrededor de u_i^* se debe cumplir que:

$$\boldsymbol{h_p(u^* + \Delta u, l^0 + \Delta l) \leq sin(\phi) \quad \phi \in [0, \pi/2)}$$

Ecuación 3.22

Ahora se presenta el método para estimar el valor que toma esta función. El sistema de control ajusta los valores de salida de los generadores en respuesta a los cambios en la demanda.

Se puede ver que la variación en la generación de cada generador es proporcional a la variación total de potencia demandada.

$$\Delta u_i[r] \propto \sum_{j \in \mathcal{V}_p^{(\ell)}} \Delta \ell_j .$$

Ecuación 3.23:[1]

De forma que empleado ecuaciones vistas anteriormente se puede llegar a una nueva función

$$\begin{aligned} h^l(\ell^0 + \Delta\ell) &:= h^p(u^* + \Delta u(\Delta\ell), \ell^0 + \Delta\ell), \\ &= \left\| M^{\mathrm{T}} L^{\dagger} \begin{bmatrix} u^* + \Delta u(\Delta\ell) \\ -(\ell^0 + \Delta\ell) \end{bmatrix} \right\|_{\infty}, \end{aligned}$$

Ecuación 3.24:[1]

El gradiente de esta función será:

$$\nabla h^l(\ell) = \left[\frac{\partial h^l(\ell)}{\partial \ell_{m+1}}, \quad \cdots \quad , \frac{\partial h^l(\ell)}{\partial \ell_n}\right],$$

Ecuación 3.25:[1]

Para pequeñas perturbaciones alrededor del valor de carga inicial l^0 podemos aproximar el valor de la función de la siguiente forma:

$$\begin{aligned} h^l(\ell^0 + \Delta\ell) &\approx h^l(\ell^0) + \nabla h^l(\ell)\Big|_{\ell=\ell^0} \Delta\ell, \\ &= h^l(\ell^0) + \sum_{i \in \mathcal{V}_p^{(\ell)}} \frac{\partial h^l(\ell)}{\partial \ell_i}\Big|_{\ell=\ell^0} \Delta\ell_i. \end{aligned}$$

Ecuación 3.26:[1]

Con forme el valor de la carga demandada cambia, se puede determinar el momento en el que el valor del punto en cohesión de fase se debe recalcular.

Por ejemplo, una estrategia sería recalcular el valor de los "set points" cuando $\varepsilon(c)$ supere la unidad.

$$\xi(c) := \frac{\displaystyle\sum_{i \in \mathcal{V}_p^{(\ell)}} \frac{\partial h^l(\ell)}{\partial \ell_i}\Big|_{\ell=\ell^0} \Delta\ell_i}{c}$$

Ecuación 3.27:[1]

Donde c es una constante que controla la frecuencia con la que los "set points" son recalculados y el margen de cohesión de fase.

$$0 < c < \left[\sin(\phi) - h^l(\ell^0)\right]$$

Ecuación 3.28:[1]

En consecuencia, si el valor de $\varepsilon(c)$ supere la unidad se considera que nos hemos alejado lo suficiente de los valores que hacen que el sistema sea estable y por lo tanto será necesario recalcular los “set-points” que aseguren la estabilidad de nuevo. Estos valores son los que se obtienen tras ejecutar el algoritmo “Feasible Flow”.

CONTROL DISTRIBUIDO SECUNDARIO DE FRECUENCIA:

En este Sistema también se ajustan los valores de los generadores siguiendo las ecuaciones anteriores. La diferencia es que en este caso los DER´s obtendrán las ganancias α_i y ki, de forma distribuida, igual que el valor de $\Delta\omega[r]$.

Los nodos de control usarán la información que obtienen localmente de las mediciones junto con la información adquirida mediante intercambios de información con los nodos vecinos para aplicar el algoritmo “*Ratio Consensus”.* De esta forma cada nodo calcula el error medio de frecuencia que será usado por los nodos de control para ajustar la potencia activa que cada DER debe suministrar.

Además cuando sea necesario recalcular los “set-points” en vez de ser una entidad centralizada quien ejecute los cálculos será la implementación distribuida del algoritmo “Feasible Flow ” (en el cual se basa este trabajo) el que permita la obtención de estos valores.

A continuación, se presenta un esquema con el que se pretende aclarar los diferentes pasos del proceso de control de frecuencia.

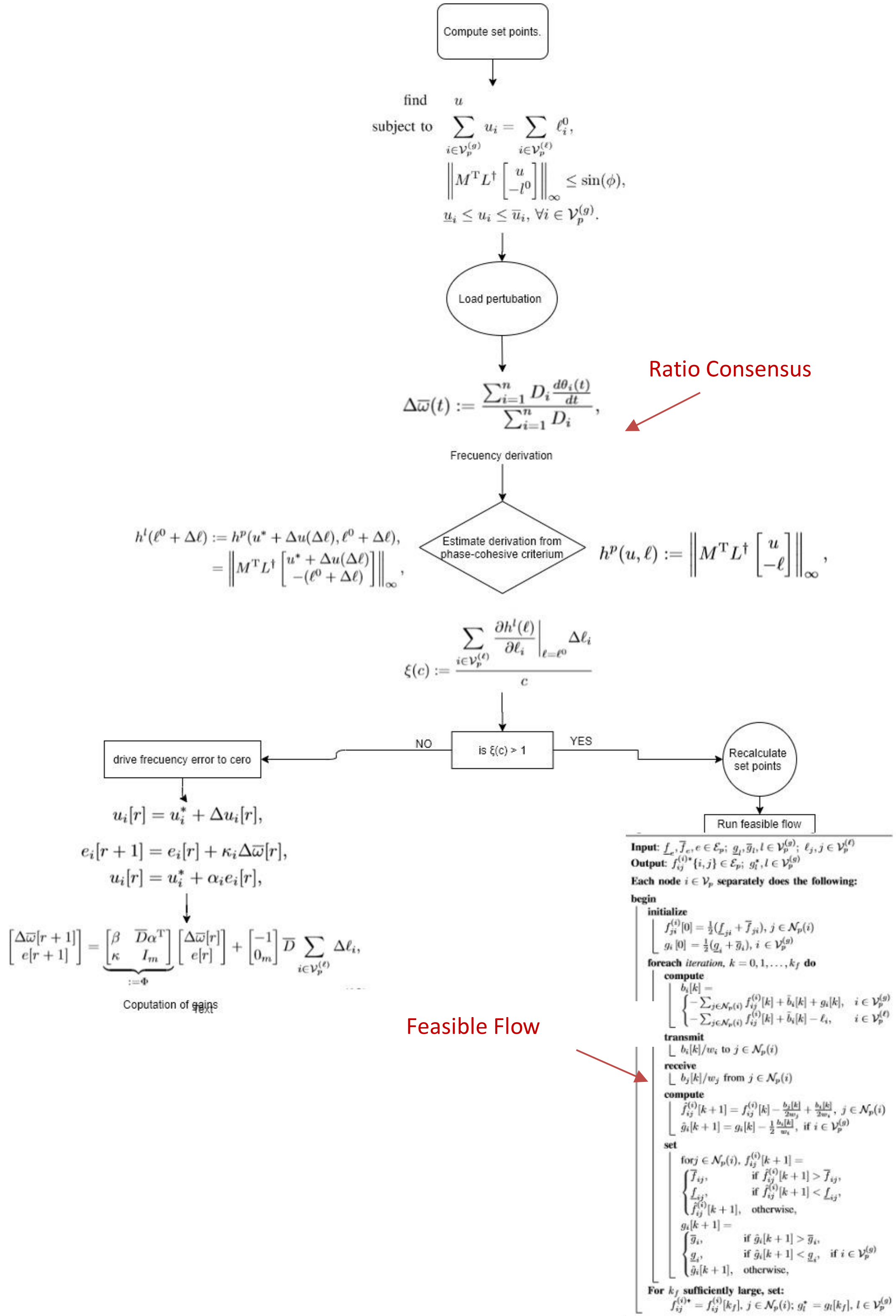


Ilustración 3-1:Esquema control secundario de frecuencia [1]

3.2 OTROS ALGORÍTMOS DISTRIBUIDOS:

Otros dos algoritmos necesarios para la toma descentralizada que se emplearán son: *Ratio Concensus* y *Max concensus.*

3.2.1 RATIO CONSENSUS

Se trata del algoritmo que emplearán los nodos para la actualización de sus variables. Este algoritmo actualizará de forma iterativa dos variables a las que denotamos por yi y zi.

Estas dos variables se van actualizando de la siguiente manera:

$$y_i[k+1] = \sum_{j \in \mathcal{N}_c(i) \cup \{i\}} \frac{1}{|\mathcal{N}_c(j)|+1} y_j[k],$$
$$z_i[k+1] = \sum_{j \in \mathcal{N}_c(i) \cup \{i\}} \frac{1}{|\mathcal{N}_c(j)|+1} z_j[k].$$

Ecuación 3.29:[1]

El cociente de estas dos variables se define como:

$$\gamma\, \mathrm{i}[K] = \frac{\mathrm{yi}[K]}{\mathrm{zi}[k]}$$

Ecuación 3.30

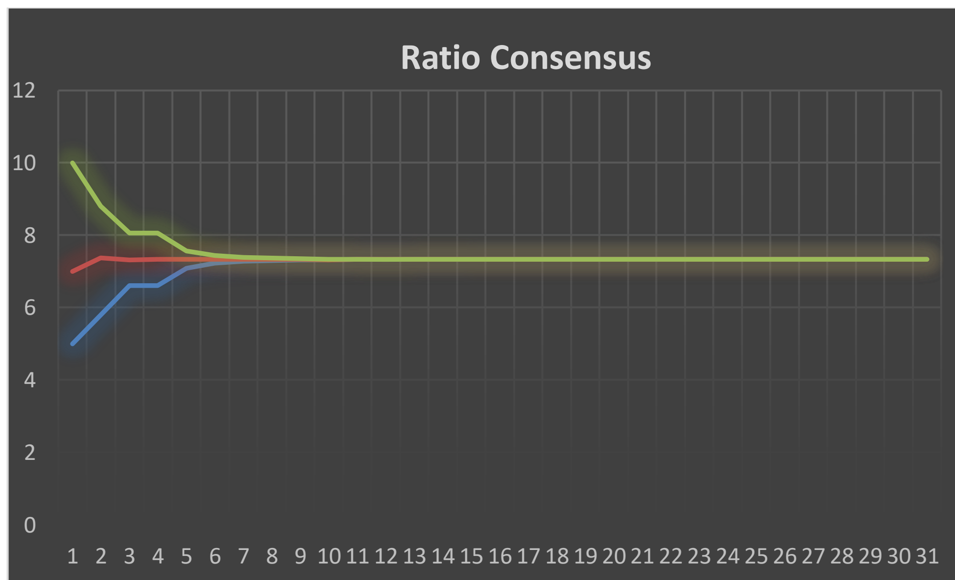
Conforme el valor de k tiende a infinito el valor de este cociente para todos los nodos i implicados en el algoritmo converge a un mismo valor, en concreto:

$$\gamma = \lim_{k \to \infty} \gamma_i[k] = \frac{\sum_{j=1}^{n} y_j[0]}{\sum_{j=1}^{n} z_j[0]}, \quad i \in \mathcal{V}_c.$$

Ecuación 3.31:[1]

Como puede verse este valor será la media aritmética del cociente de los valores iniciales de los nodos.

En la imagen inferior se muestra la evolución de este algoritmo para tres nodos, con 30 iteraciones donde se aprecia como el resultado final al que convergen es la media de sus valores iniciales.



Gráfica 3-1:Ejemplo simulación "Ratio Consensus" con tres nodos.

RATIO CONSENSUS PARA EL CÁLCULO DEL ERROR DE FRECUENCIA:

Pare el cálculo error de frecuencia se emplea este algoritmo de la siguiente forma:

$$x_i[r] = \begin{cases} u_i[r], & i \in \mathcal{V}_p^{(g)}, \\ -(\ell_i^0 + \Delta\ell_i), & i \in \mathcal{V}_p^{(\ell)}, \end{cases}$$

Ecuación 3.32:[1]

$$y_i[0] = x_i[x]$$

$$z_i[0] = D_i$$

$$\lim_{k \to \infty} (y_i[k]/z_i[k]) = \Delta\overline{w}[r]$$

Ecuación 3.33

Sin embargo, este algoritmo no es suficientemente robusto contra "communication packet drops", es decir cuando por alguna razón la información enviada no llega a su destino.

A continuación, se presenta un algoritmo desarrollado por los ingenieros que trabajan en el proyecto de la universidad de Illinois que es lo suficientemente robusto para soportar este tipo de fallos.

RATIO CONSENSUS ADAPTADO:

Dado que en las comunicaciones inalámbricas son muy comunes los "drop packages failures", que consiste en la pérdida de información entre los controladores. Para adaptar el algoritmo Ratio Concensus a este tipo de fallos de distribución entre los nodos, se ha desarrollado un algoritmo que asegura que cuando un nodo falla la contribución al resultado final pueda ser asumida por sus vecinos.

Cuando un nodo falla el DER o la carga a la que está conectado no cambia su "*set-point*", y el objetivo es que el valor su *"set-point"* se tenga encuentra al implementar el Ratio Consensus.

Esta variante de dicho algoritmo en primer lugar asegura que durante la primera iteración de cada ronda todos los nodos de control almacenan la información recibida por sus vecinos. Además, cada nodo de control seleccionará de forma aleatoria a otro nodo como heredero y cada heredero será informado de su estado.

$$\frac{yi[0]}{\delta i + 1} \; y \; \frac{yi[0]}{\delta i + 1}$$

Ecuación 3.34

Donde $\delta i = |N_c(i)|$ representa el número de nodos vecinos del nodo i.

De esta forma cuando el nodo i falla, cada nodo vecino j a excepción el nodo heredero ajustará sus valores iniciales de la siguiente forma:

$$yj[0] = yj[0] + \frac{yi[0]}{\delta i + 1}$$

$$zj[0] = zj[0] + \frac{zi[0]}{\delta i + 1}$$

Ecuación 3.35

Al mismo tiempo el nodo heredero:

$$yj[0] = yj[0] + \frac{2 * yi[0]}{\delta i + 1}$$

$$zj[0] = zj[0] + \frac{2 * zi[0]}{\delta i + 1}$$

Ecuación 3.36

De esta forma aseguramos que el algoritmo sea resiliente al fallo de un nodo de control.

3.2.2 MAX-MIN CONSENSUS

Este algoritmo nos permitirá encontrar el valor máximo o mínimo de los valores que tengan los diferentes nodos del sistema, de una forma distribuida. El procedimiento es el siguiente:

Considerando el vector:

$$\eta \coloneqq [\eta_1, \eta_2 \ldots\ldots \eta_n]^T$$

donde η_i es un valor conocido únicamente por el nodo i. Como se ha indicado el objetivo es encontrar el valor máximo o mínimo de entre los valores absolutos de los elementos de ese vector, a este valor lo denotamos como $\overline{\eta}$.

$\mu_i[K]$ es la estimación de $\overline{\eta}$ del nudo i en la iteración k. Este valor se inicializará de la siguiente forma:

- $\mu_i[0] = \eta_i$, y se actualizará de acuerdo con:

$$\overline{\mu_i}[k+1] = \max \overline{\mu_j}[k]$$

Después de un número finito de iteraciones limitado por el diámetro del sistema cada nodo puede obtener el valor de η. Para una Km ≤ Δ_c , μ_i[k] = η for K ≥ km.

Este algoritmo se usará con el objetivo de determinar el número de iteraciones necesarias en el "*Feasible Flow algorithm*". Este algoritmo deberá seguir iterando hasta que el máximo valor del balance en cada nodo sea inferior a un valor predeterminado como por ejemplo 0.001. Por lo tanto será *Max Consensus* el que indicará cuando el máximo valor de todos los balances de los nodos es inferior a ese valor preestablecido. Gracias a este podemos optimizar la precisión obtenida para el menor tiempo de ejecución posible.

3.3 DESCRIPCIÓN DEL ALGORITMO "FEASIBLE FLOW".

En este apartado se explicará el algoritmo "Feasible Flow" desarrollado y presentado también en el artículo [1].

La finalidad de este algoritmo es determinar la generación individual de los generadores que suplen la demanda de las cargas en una microrred, sin violar ninguna de las restricciones de generación ni los límites de la potencia máxima que puede circular por cada una de las ramas.
En la notación empleada fij hace referencia al flujo de potencia que circula entre los nodos i y j , la dirección del flujo será del nudo i al nudo j.

El fujo que puede circular por las ramas está restringido por un límite superior ($\overline{fij}$) y uno inferior ($\underline{fij}$) de la misma forma la potencia que puede suministrar cada uno de los nudos generadores también está limitado por límites superiores ($\overline{gi}$) e inferiores ($\underline{gi}$)
Para los nudos de demanda, li denota la potencia demanda.
Finalmente denotamos el balance de potencias en cada nudo como bi.

Como ya se ha explicado anteriormente, suponiendo que las potencias demandadas por los nudos de carga son conocidas, el objetivo es asignar valores para cada uno de los flujos de las ramas fij y los valores de salida de potencia que es suministrada por los generadores, de forma que: El balance de potencia de cada nodo sea cero, la potencia generada coincida con la demandada por las cargas y que tanto el flujo de las ramas y la potencia de los generadores se encuentren dentro de los límites establecidos.

Este algoritmo se encargará del ajuste local de las estimaciones de los flujos por las líneas y las salidas de los generadores de forma iterativa, de manera que se calculan de forma aproximada los valores asintóticos que satisfacen estas condiciones recién mencionadas.

Para sustentar la naturaleza iterativa del algoritmo propuesto, cada procesador local mantiene una estimación para el valor del flujo de potencia de las ramas que le conectan con todos sus nudos vecinos.

Para cada nodo *i* la estimación para el flujo con cada nudo *j* al que está conectado en la iteración $k = 0,1$ .... Se denota como: $\mathrm{fij}^{(\mathrm{i})}[\mathrm{K}]$.
De la misma forma para cada generador el valor estimado del valor de salida en la iteración k se denota por , $\mathrm{gi}^{(\mathrm{i})}[\mathrm{K}]$.

Finalmente basándose en el valor de los flujos y las salidas de los generadores, el valor del balance de potencia en cada nudo es calculado en cada iteración de la siguiente manera:

$$b_i[k] := \begin{cases} -\sum_{j\in\mathcal{N}_p(i)} f_{ij}^{(i)}[k] + g_i[k], & i \in \mathcal{V}_p^{(g)}, \\ -\sum_{j\in\mathcal{N}_p(i)} f_{ij}^{(i)}[k] - \ell_i, & i \in \mathcal{V}_p^{(\ell)}. \end{cases}$$

Ecuación 3.37: Balance [1]

El algoritmo propuesto para la resolución de forma distribuida es dado por el siguiente procedimiento en el cual las estimaciones del flujo y la salida de los generadores es inicializada y de forma iterativa actualizada usando un proceso de cuatro pasos.

- ***Paso 1: Inicialización.***
  Cada nodo inicializa sus estimaciones de flujo como la media de sus respectivos limites superiores e inferiores.

$$\boldsymbol{fij}[\boldsymbol{0}] = \frac{\boldsymbol{1}}{\boldsymbol{2}}\left(\overline{\boldsymbol{fij}} + \underline{\boldsymbol{fij}}\right)$$

Ecuación 3.38: Flujo inicial

De forma análoga el valor inicial para la generación de los nudos generadores es la media de sus límites.

$$\boldsymbol{gi}[\mathbf{0}] = \frac{\mathbf{1}}{\mathbf{2}} \left(\overline{\boldsymbol{gi}} + \underline{\boldsymbol{gi}}\right)$$

Ecuación 3.39: Generación Inicial

- ***Paso 2:***

  Se calculan los valores de flujo y potencia de salida en la iteración k ($bi[K]$) según la ecuación **3.34.**

  Después cada nodo i ajusta su estimación para cada flujo de tal forma que hace el balance de potencia cero a medida que aumenta el número de iteraciones.

$$\tilde{f}_{ij}^{(i)}[k+1] = f_{ij}^{(i)}[k] + \frac{b_i[k]}{w_i}, \; j \in \mathcal{N}_p(i),$$

Ecuación 3.40: Cálculo Flujo [1]

El valor de la estimación de la salida de los generadores se calcula de forma similar:

$$\hat{g}_i[k+1] = g_i[k] - \frac{b_i[k]}{w_i}.$$

Ecuación 3.41: Cálculo generación [1]

El valor de wi se define de la siguiente forma:

$$\boldsymbol{wi} \coloneqq \boldsymbol{\delta_p}(\boldsymbol{i}) + \mathbf{1}$$

Ecuación 3.42

Donde que δ_p hace referencia a el número de nodos a los que se encuentra conectado el nudo i.

- ***Paso 3:***

  Dado que cada nodo actualiza la estimación del flujo en el primer paso de forma independiente es probable que dos nudos vecinos tengan diferentes estimaciones para el flujo entre ellos. Mediante el intercambio de las estimaciones con sus nodos vecinos cada nodo actualiza localmente las estimaciones de la siguiente manera:

$$\hat{f}_{ij}^{(i)}[k+1] = \frac{1}{2}\left(\tilde{f}_{ij}^{(i)}[k+1] - \tilde{f}_{ji}^{(j)}[k+1]\right).$$

Ecuación 3.43: [1]

Con $j \in N_p(i)$.

Este paso no es necesario para estimar las salidas de los generadores ya que cada procesador solo mantiene una estimación para su propia salida.

- ***Paso 4:***

  Dado que durante los primeros pasos anteriores la estimación del flujo puede haber sido ajustada de forma que se sobrepasen alguno de los límites, debemos asegurar que esto no ocurra y todas las estimaciones permanezcan dentro de los límites:

$$f_{ij}^{(i)}[k+1] = \begin{cases} \overline{f}_{ij}, & \text{if } \hat{f}_{ij}^{(i)}[k+1] > \overline{f}_{ij}, \\ \underline{f}_{ij}, & \text{if } \hat{f}_{ij}^{(i)}[k+1] < \underline{f}_{ij}, \\ \hat{f}_{ij}^{(i)}[k+1], & \text{otherwise.} \end{cases}$$

Ecuación 3.44:Comprobación límites flujos[1]

$$g_i[k+1] = \begin{cases} \overline{g}_i, & \text{if } \hat{g}_i[k+1] > \overline{g}_i, \\ \underline{g}_i, & \text{if } \hat{g}_i[k+1] < \underline{g}_i, \\ \hat{g}_{ij}[k+1], & \text{otherwise.} \end{cases}$$

Ecuación 3.45:Comprobación límites generación [1]

- ***Paso 5:***

Teniendo en cuenta que dos nodos conectados tienen los mismos límites para el flujo que circula por la línea de conexión entre ambos se deduce que al inicio de la iteración las estimaciones de ambos nodos son aditivamente inversos ($fij^{(i)}[K] = -fij^{(j)}[K]$) por lo que de esta forma el algoritmo puede simplificarse de la siguiente forma:

$$f_{ij}[k+1] = \left[f_{ij}[k] + \frac{1}{2}\frac{b_i[k]}{w_i} - \frac{1}{2}\frac{b_j[k]}{w_j} \right]_{\underline{f}_{ij}}^{\overline{f}_{ij}}, \quad \forall \{i,j\} \in \mathcal{E}_p,$$

Ecuación 3.46:[1]

$$g_i[k+1] = \left[g_i[k] - \frac{1}{2}\frac{b_i[k]}{w_i} \right]_{\underline{g}_i}^{\overline{g}_i}, \qquad \forall i \in \mathcal{V}_p^{(g)},$$

Ecuación 3.47:[1]

El algoritmo completo queda por lo tanto de la siguiente forma:

Algorithm 1: Distributed feasible flow algorithm

Input: $\underline{f}_e, \overline{f}_e, e \in \mathcal{E}_p$; $\underline{g}_l, \overline{g}_l, l \in \mathcal{V}_p^{(g)}$; $\ell_j, j \in \mathcal{V}_p^{(\ell)}$

Output: $f_{ij}^{(i)*}\{i,j\} \in \mathcal{E}_p$; $g_l^*, l \in \mathcal{V}_p^{(g)}$

Each node $i \in \mathcal{V}_p$ **separately does the following:**

begin

initialize

$f_{ji}^{(i)}[0] = \frac{1}{2}(\underline{f}_{ji} + \overline{f}_{ji}),\ j \in \mathcal{N}_p(i)$

$g_i[0] = \frac{1}{2}(\underline{g}_i + \overline{g}_i),\ i \in \mathcal{V}_p^{(g)}$

foreach *iteration,* $k = 0, 1, \ldots, k_f$ **do**

compute

$$b_i[k] = \begin{cases} -\sum_{j \in \mathcal{N}_p(i)} f_{ij}^{(i)}[k] + \tilde{b}_i[k] + g_i[k], & i \in \mathcal{V}_p^{(g)} \\ -\sum_{j \in \mathcal{N}_p(i)} f_{ij}^{(i)}[k] + \tilde{b}_i[k] - \ell_i, & i \in \mathcal{V}_p^{(\ell)} \end{cases}$$

transmit

$b_i[k]/w_i$ to $j \in \mathcal{N}_p(i)$

receive

$b_j[k]/w_j$ from $j \in \mathcal{N}_p(i)$

compute

$\hat{f}_{ij}^{(i)}[k+1] = f_{ij}^{(i)}[k] - \frac{b_j[k]}{2w_j} + \frac{b_i[k]}{2w_i},\ j \in \mathcal{N}_p(i)$

$\hat{g}_i[k+1] = g_i[k] - \frac{1}{2}\frac{b_i[k]}{w_i}$, if $i \in \mathcal{V}_p^{(g)}$

set

for $j \in \mathcal{N}_p(i)$, $f_{ij}^{(i)}[k+1] =$

$$\begin{cases} \overline{f}_{ij}, & \text{if } \hat{f}_{ij}^{(i)}[k+1] > \overline{f}_{ij}, \\ \underline{f}_{ij}, & \text{if } \hat{f}_{ij}^{(i)}[k+1] < \underline{f}_{ij}, \\ \hat{f}_{ij}^{(i)}[k+1], & \text{otherwise,} \end{cases}$$

$g_i[k+1] =$

$$\begin{cases} \overline{g}_i, & \text{if } \hat{g}_i[k+1] > \overline{g}_i, \\ \underline{g}_i, & \text{if } \hat{g}_i[k+1] < \underline{g}_i, \\ \hat{g}_i[k+1], & \text{otherwise,} \end{cases} \quad \text{if } i \in \mathcal{V}_p^{(g)}$$

For k_f **sufficiently large, set:**

$f_{ij}^{(i)*} = f_{ij}^{(i)}[k_f],\ j \in \mathcal{N}_p(i)$; $g_l^* = g_l[k_f],\ l \in \mathcal{V}_p^{(g)}$

Tabla 2: Feasible Flow[1]

Se puede entender este algoritmo como una función que coge los límites de los flujos y de los valores de generación así como los valores de la demanda de las cargas como entradas y calcula los valores de los flujos y generación que satisfaces el problema "*feasible Flow*".

$$h^f : \left(\overline{fij}, \underline{fij}, \overline{gi}, \underline{gi}, l\right) \rightarrow (f^*, g^*)$$

El algoritmo propuesto deberá realizar la iteración un número infinito de veces para resolver el problema del flujo de potencia, sin embargo, se demuestra que parando la iteración en un numero finito de veces $\mathrm{k_f}$ el error entre las estimaciones y los valores asintóticos reales puede ser lo suficientemente pequeño.

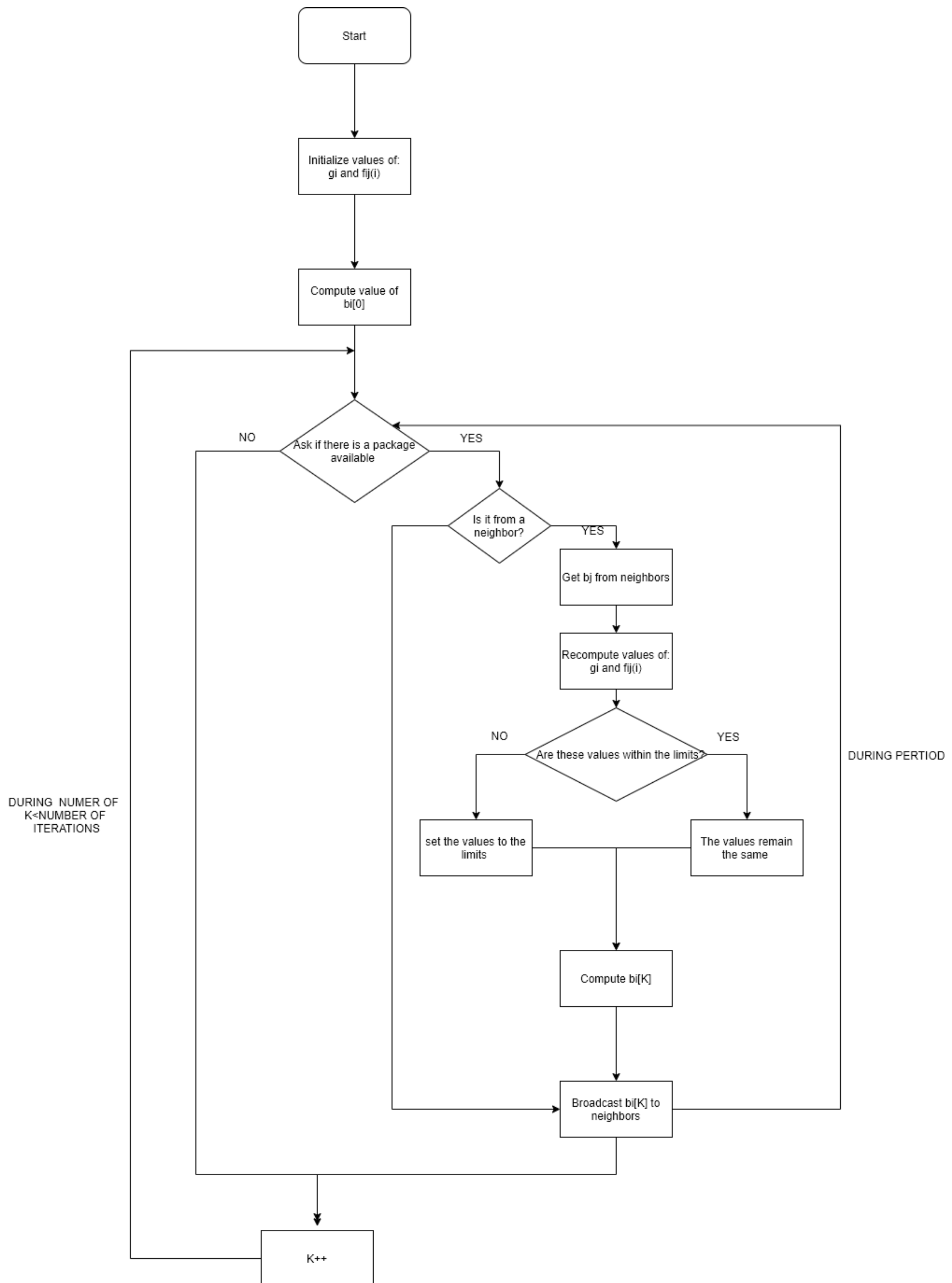


Ilustración 3-2:Feasible Flow(Flow chart)

Capítulo 4. DESARROLLO DEL PROYECTO

4.1 PASO 1: DECISIÓN CENTRALIZADA EN RED DE 3 NODOS.

Para comenzar el proyecto el primer paso es la implementación del algoritmo anteriormente mencionado para la resolución del problema de flujo de potencia en una red simple con tan solo tres nodos. El objetivo de este programa es la familiarización con el algoritmo, su función, así como la adaptación al lenguaje de programación . El desarrollo de este programa permite apreciar las diferencias entre lo que sería un sistema de control centralizado y el distribuido en el que se empleará un lenguaje orientado a objetos. Esto permite al programador entender la estructura del proceso que sigue cada sistema de control, además a apreciar las diferencias en la complicación que pueden existir en la elaboración de los programas que constituyen cada uno de los dos sistemas de toma de decisiones.

Este programa fue realizado en C, sin la utilización de clases ni objetos.

Este programa resuelve el problema del flujo de potencia de **forma centralizada**, es decir en este caso no se produce ningún intercambio de información entre los nodos, sino que es una única entidad la que recibe toda la información y realiza los cálculos.

El objetivo de esto es poder comprobar posteriormente que los resultados obtenidos tanto en el algoritmo descentralizado y centralizado son los mismos, así como poder hacer comparaciones en el tiempo de resolución y otros parámetros.

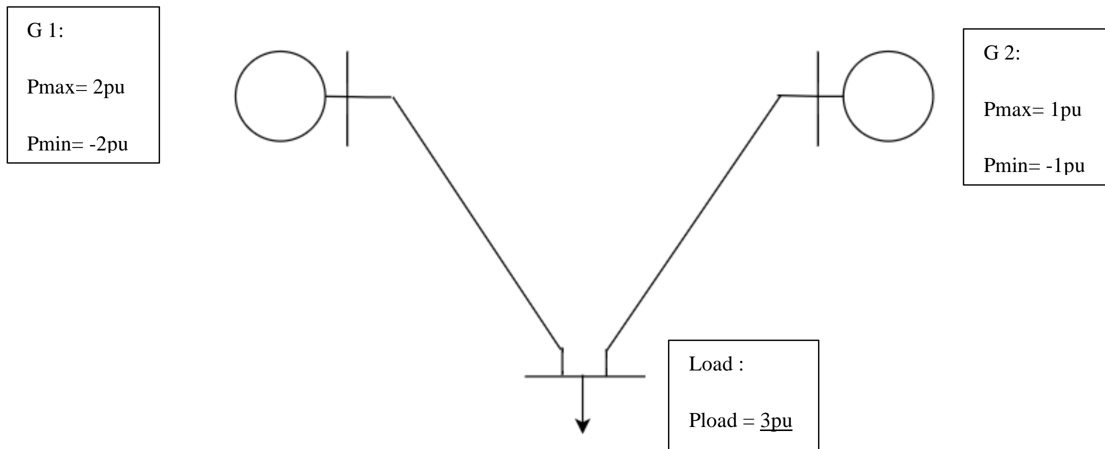


Ilustración 4-1:Sistema 3 nodos

Una vez ejecutado el código los resultados obtenidos son los siguientes:

SOLUCIONES:

El balance en cada nodo es:

$b[i] = [b1, b2, b3] =$ `-7.15256e-007   -2.38419e-007   -9.53674e-007`

La potencia activa generada en cada nodo es :

$g[i] = [g1, g2, g3] =$ `1  2  -3`

El flujo através de las líneas son:

$$\begin{matrix} f11 & f12 & f13 \\ f21 & f22 & f23 \\ f31 & f32 & f33 \end{matrix} = \begin{matrix} 0 & 0 & 1 \\ 0 & 0 & 2 \\ -1 & -2 & 0 \end{matrix}$$

Como se espera el vector que contiene los balances de flujo en cada nodo es muy próximo a cero, lo que quiere decir que la potencia que llega y sale de cada nodo se compensa.

Por otro lado, el vector generación nos dice que el nodo uno y dos generan uno y dos de potencia para suplir la demanda de 3 del tercer nodo, todo esto sin que se sobrepase ninguna de las restricciones de los generadores ni de las líneas.

4.2 PASO 2: FEASIBLE FLOW (IMPLEMENTACIÓN DISTRIBUIDA)

En este capítulo se describirá el proceso de desarrollo del algoritmo distribuido, la estructura y funcionamiento de este.
Todo el código ha sido desarrollado en el lenguaje de programación C++. Para hacer posible la implementación de un algoritmo distribuido es necesario la programación orientada a objetos.
El total del programa de halla distribuida en un total de seis librerías diferentes:

1. ***OGraph.h***
2. ***OGraph.cpp***
3. ***OAgent.h***
4. ***OAgent.cpp***
5. ***XBee.h***
6. ***XBee.cpp***

Las librerías **XBee** son las que se encargan de construir las clases y las funciones que hacen posible las comunicaciones entre los nodos.

Las librerías de **OGraph** recogen todo lo relacionado con la estructura de la red, en ella se definen las clases principales como: **OVertex o OGraph** dentro de la clase de OVertex encontramos subclases como: **OLocalVertex, ORemoteVertex…** , así como todas sus propiedades y funciones.

Por ejemplo, la clase **OVertex** hace referencia a todos los nodos del sistema, la subclase de **OLocalvertex** hace referencia al propio nodo, es decir a cada nodo en concreto mientras que **ORemoteVertex** serán el resto de los nodos que componen el sistema.
Dentro de estas subclases se definen propiedades y funciones que permiten acceder a dichas propiedades, así como múltiples funciones que permiten trabajar con ellas.

La clase **OGraph**, por ejemplo, permiten añadir o eliminar nuevos "in-neigborhs"(vecinos) a un determinado vértice. En esta carpeta se definen funciones y propiedades relacionadas con la estructura de la red.
En la carpeta de OGraph.cpp se desarrollan las funciones más extensas de las diferentes clases y subclases mencionadas.

Finalmente, en las carpetas **OAgent** se encuentran definida la clase de OAgent donde se definen muchas funciones relacionadas con los algoritmos y todas las funciones necesarias para la implementación de estos y es aquí donde se encuentra el grueso del algoritmo "*Feasible Flow*".
De todas formas, solo se entrará en detalle en aquellas las funciones que sean relevantes para la implementación de nuestro algoritmo.

FUNCIONES INICIALES:

En primer lugar, empezaremos con la función a la que llamaremos desde el Arduino cuando sea necesario recalcular los valores de salida de los generadores.
Esta función recibe el nombre de **feasibleFlowAlgorithm_RSL** y llama a la función **leaderfeasibleFlow_RSL** en el caso de que el nodo en el que se está ejecutando esta función haya sido establecido como el nodo líder y **nonleaderfeasibleFlow_RSL** en caso contrario.
Esta función recibe como entradas, un valor de tipo boolean que será verdadero en caso de que el nodo sea generador, el número de iteraciones, el periodo (tiempo de duración

de cada iteración) y la carga demandada por dicho nodo en ese instante, este valor se obtiene del simulador Typhoon Hil. Después devolverá el valor de salida de los generadores.

```
float OAgent::feasibleFlowAlgorithm_RSL(bool busgen, uint8_t iterations, uint16_t period, float load ) {
    srand(analogRead(7));
    float gamma = 0;
    if(isLeader())
    {
        gamma=leaderfeasibleFlow_RSL(busgen,iterations,period,load);
    }
    else
    {
        gamma=nonleaderfeasibleFlow_RSL(busgen,iterations,period,load);
    }
    return gamma;
}
```

Ilustración 4-2: FunciónFeasibleFlowAlgorithm_RLS

A continuación se muestran las funciones **leaderfeasibleFlow_RSL** y **nonleaderfeasibleFlow_RSL** pero no se entrará en el detalle de su funcionamiento. Lo único que es necesario saber es que ambas funciones llamarán a la función **feasibleFlowAlgorithm** donde se encuentra el cuerpo del algoritmo explicado anteriormente. Por otro lado es importante saber que la función del nodo que es considerado como **leader** es enviar la señal al resto con la información sobre cuando deben llevar a cabo el cálculo de la generación cada uno de ellos.

```
float OAgent::leaderfeasibleFlow_RSL(bool busgen, uint8_t iterations, uint16_t period, float load ) {
    unsigned long t0 = myMillis();
    unsigned long startTime = t0 + RC_DELAY;
    OLocalVertex * s = _G->getLocalVertex();
    float gamma = 0;
    bool scheduled =_waitForChildSchedulePacketRC(SCHEDULE_FAIR_SPLIT_HEADER,SCHEDULE_TIMEOUT, startTime, iterations, period);

    if (!scheduled)
    {
        Serial<<"RC scheduling was a FAIL!"<<endl;
        delay(5);
        gamma = -1;
    }
    else
    {
        Serial<<"RC scheduling was a SUCCESS!"<<endl;
        delay(5);
        Serial << "Correct Startime is " <<startTime<<", and current time is "<< myMillis()<<endl;
        delay(5);
        if(_waitToStart(startTime,true,10000))
        {
            Serial <<"My startime is "<< myMillis() <<endl;
            delay(5);
            feasibleFlowAlgorithm(busgen,iterations,period,load);
        }
    }
    return gamma;
}
```

Ilustración 4-3:Función leaderfeasibleFlow_RSL

```
float OAgent::nonleaderfeasibleFlow_RSL(bool busgen, uint8_t iterations, uint16_t period, float load ){
    unsigned long startTime = 0;
    //delay(50);
    float gamma = 0;
    bool scheduled = _waitForParentSchedulePacketRC(startTime,iterations,period,-1);
    if(scheduled)
    {
        Serial<<"FF scheduling was a SUCCESS!"<<endl;
        delay(5);
        Serial << "Correct Startime is " <<startTime<<", and current time is "<< myMillis()<<endl;
        delay(5);
        if(_waitToStart(startTime,true,10000)) {
            Serial <<"My startime is "<< myMillis() <<endl;
            delay(5);
           gamma=feasibleFlowAlgorithm(busgen,iterations,period,load);
        }
    }
    else
    {
        Serial<<"FF scheduling was a FAIL!"<<endl;
        delay(5);
        gamma = -1;
    }
    return gamma;
}
```

Ilustración 4-4:función nonleaderfeasibleFlow_RSL

Las tres funciones que se han visto se encuentran definidas en el archivo de OAgent.h y desarrolladas en la librería de OAgent.cpp , dentro de la clase OAgent.

FUNCIONES PARA LA TRANSMISIÓN DE INFORMACIÓN:

A continuación, se presenta y se explican las funciones empleadas por los nodos para **transmitir la información** sobre su balance de potencia, así como recibir esta información de los nodos vecinos.

La función encargada de enviar el valor del balance de potencia es la siguiente:

En esta función se construye un vector (donde cada elemento tiene una capacidad de 8 bits), es importante tener en cuenta que al estar en lenguaje binario no podemos enviar ni números **decimales** ni **negativos**, para ello se debe multiplicar el valor del balance por un valor BASE para eliminar los decimales, y por -1 si es negativo para eliminar valores negativos. Por último, este valor dividido por **wi** se guarda en los cuatro últimos elementos del vector. Posteriormente para hacer posible él envió de valores negativos almacenaremos en el último elemento del vector el valor 0 en caso de que el balance sea negativo y 1 si es positivo.

Finalmente, la información de este vector es enviada al resto de nodos.

```
void OAgent_OPF::_broadcastBalanceFeasibleFlow(float bi, float wi) {

    uint8_t payload[7];
    bi =  bi*BASE;
    uint8_t sigBi ;

    uint32_t Bi;
    //long Wi = long(wi);
    if(bi<0){
        sigBi = 0;
        bi = -1* bi;
        Bi = (uint32_t) (bi/wi);
   }else{
        sigBi = 1;
        Bi = (uint32_t) (bi/wi);

   }
    payload[0] = FAIR_SPLITTING_HEADER;
    payload[1] = FAIR_SPLITTING_HEADER>>8;
    payload[2] = sigBi;
    payload[3] = Bi; // guardamos el valor de mu los primeros?? 16 bits
    payload[4] = Bi>> 8;// guardamos el valor de mu los ultimos?? 16 bits
    payload[5] = Bi>> 16;
    payload[6] = Bi>> 24 ;

    _zbTx = ZBTxRequest(_broadcastAddress, ((uint8_t * )(&payload)), sizeof(payload));
    //se ha transformado nuestro vector de 16bits en uno se 8bits --> se ha doblado la
    unsigned long txTime = _xbee->sendTwo(_zbTx,false,true); // transmit with time sta
#ifdef VERBOSE
    Serial << _MEM(PSTR("Transmit time: ")) << txTime << endl;
#endif
}
```

Ilustración 4-5:Función broadcastBalanceFeasibleFlow

Para la recogida de esta información por parte de los nodos vecinos se emplearán las siguientes funciones.

```
float OAgent_OPF::_getBjFromPacket() {
    uint8_t ptr = 2;
   // uint8_t ptr1 = 5;
    float bj;
    bj = _getFloat32FromPacket(ptr)/BASE;
    return bj;
}
```

Ilustración 4-6:Función getBjFromPacket

```
float OAgent_OPF::_getFloat32FromPacket(uint8_t &lsbByteNumber) {
    lsbByteNumber += 5;
    int32_t mag = (uint32_t) ((uint32_t(_rx->getData(lsbByteNumber-1)) << 24) + (uint32_t(_rx->getData(lsbByteNumber-2))<< 16)
    + (uint16_t(_rx->getData(lsbByteNumber-3)) << 8) + uint8_t(_rx->getData(lsbByteNumber-4)));
    int8_t sign = -1 + ((_rx->getData(lsbByteNumber-5))*2);
    float value = (float) (sign*mag);

    return value;
}
```

Ilustración 4-7:Funcón getFloat32FromPacket

Para obtener la información que se encuentra en el vector que recibimos creamos un puntero **ptr,** que extrae la información del balance y posteriormente se divide por un valor base para devolver así el carácter decimal a este valor. Por otro lado, en la función que se muestra en la ilustración 4.6 se accede a la información que indica el signo del balance. De esta forma obtenemos el valor del balance en forma decimal y con el signo correspondiente.

CLASES Y OBJETOS PRINCIPALES:

Para entender el proceso de comunicación hay que distinguir dos clases de objetos , la clase **OLocalVertes** y **ORemoteVertex.**

OLocalVertex:

Para construir un objeto de esta clase se emplean constructores que deben recibir valores como: los límites de generación, nodeID(número de identificación del nodo), dirección de memoria del nodo…

Cada objeto de la clase **OLocalVertex** tiene una serie de propiedades. La siguiente imagen muestra algunas de ellas, que emplearemos en el algoritmo.

```
float _p; //per-unit
float _min;
float _max;
float _bp;
uint8_t _outDegree;
```

Ilustración 4-8:Propiedades OLocalVertex

_p : hace referencia a la potencia activa generada en el nodo.

_bp: guarda la información del balance de potencia activa en el nodo.

_max,_min: guardan la información sobre los límites de generación del nodo.(estos valores son introducidos cuando se construye el nodo).

_outDegrre: se refiere al número de nodos con los que dicho nodo está conectado más uno.

Para manejar estas variables disponemos de las siguientes funciones, que nos permitirán acceder al valor de estas, así como modificarlo.

```
inline void setGi(float p){ _p=p;}
inline float getGi(){return _p;}
inline float getMin() { return _min; }
inline float getMax() { return _max; }
inline float getActiveBalance() { return _bp; }
inline void setActiveBalance(float bp) {_bp = bp; }
```

```
inline float getActiveDemand() { return _pd; }
```

```
inline void setActiveDemand(float pd) {_pd = pd; }
```

Ilustración 4-9:Métodos OLocalVertex

<u>ORemoteVertex:</u>

Esta clase la emplearemos fundamentalmente para manejar los flujos por las líneas.

Una forma de crear un objeto de esta clase es introduciendo la dirección de memoria de uno de los nodos con el que se encuentre conectado, el ID (el número de

identificación) de este, los valores de resistencia y reactancia de la línea que los interconecta y los valores límite del flujo que pueden soportar.

Alguna de las propiedades que encontramos en esta clase y que emplearemos para la implementación del algoritmo son:

```
float _fpmax;
float _fpmin;
float _fp0;
float _fp;
```

Ilustración 4-10 :Propiedades ORemoteVertex

Y las funciones que nos permitirán obtener sus valores y modificarlos:

```
inline float getActiveFlowMax() { return _fpmax; }
inline float getActiveFlowMin() { return _fpmin; }
inline float getActiveInitialFlow() { return _fp0; }
inline float getActiveFlow() { return _fp; }
```

```
inline void setActiveInitialFlow(float fp0) {_fp0 = fp0; }
```

Ilustración 4-11:Métodos ORemoteVertex

Otra Clase que también emplearemos es la Clase **Linkedlist.**

LinkedList se trata de una estructura linear de datos donde cada elemento es un objeto separado, a estos elementos se les llama nodos y estos están conectados mediante punteros . Cada nodo contiene dos partes, por un lado, los datos y por otro la referencia del siguiente nodo.

Se trata de una estructura dinámica, donde el número de nodos no está fijado y puede aumentar y disminuir según queramos. Este tipo de estructuras puede resultar muy útiles y se usan con bastante frecuencia.

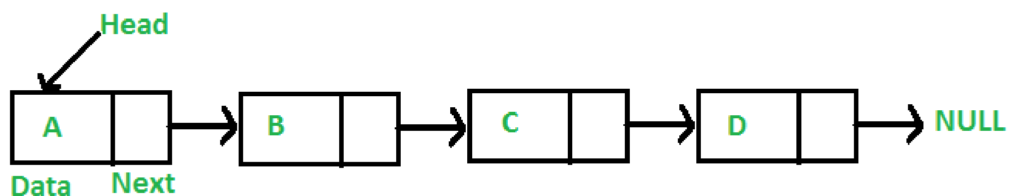


Ilustración 4-12:(https://www.geeksforgeeks.org/linked-list-set-1-introduction/)

En el desarrollo de este condigo emplearemos alguna de las funciones de esta estructura.

CÓDIGO FUNCIÓN PRINCIPAL:

Una vez se han presentado las clases y funciones básicas que permitirán formarse una idea superficial del funcionamiento del algoritmo distribuido aquí se presentarán el código de la función principal y alguna de las funciones más destacables

Antes de empezar, es necesario destacar que lo más importante para la comprensión del algoritmo es que está función se ejecutará en cada uno de los nodos del sistema en el mismo periodo de tiempo.

El código completo de la función principal puede encontrarse en el Anexo I, aunque en el siguiente punto (4.3) se explicarán la partes principales de dicha función.

4.3 EXPLICACIÓN FUNCIÓN PRINCIPAL:

- **Parte 1:**

En primer lugar vemos que la función recibe como *imputs* : una variable de tipo boolean (*busgen*) que nos indica si se trata de un nudo de generación o de demanda; las variables *iterations y period* que nos dan la información sobre el número de iteraciones que se realizarán así como la duración de cada una de estas; por ultimo recibimos la información de potencia activa demandada en el nudo en el que se está ejecutando el algoritmo (*load*). Finalmente, la función devuelve el valor de la generación (*set-points*) del nodo (que será cero en caso de que se trate de un nodo de demanda).

Al comienzo de la función se crean punteros a los tres objetos de las clases vistas anteriormente **LocalVertex** (hace referencia al nodo donde se esta ejecutando el algorítmo)**, RemoteVertex** (nos sirve para comunicarnos con los nodos vecinos) **y LinkedList** (se empleará para obtener información de los nodos vecinos).

Se definen también dos punteros más que se emplearán **neighborStatusP** (nos permitirá saber y hacer el seguimiento de si se ha recibido la información de los diferentes nodos vecinos)y **neighborP** (con este puntero podremos acceder a las propiedades de los nodos vecinos).

```
OLocalVertex * s = _G->getLocalVertex();//almacenar puntero a localVertex
ORemoteVertex * n = _G->getRemoteVertex(1);//almacenar puntero a RemoteVertex
LinkedList * l = _G->getLinkedList();//almacenar puntero a Linklist

l->resetLinkedListStatus(s->getStatusP());//se inicializan los estados de forma
que indique que no se ha recibido información de ningún nodo vecino

l->updateLinkedList(s->getStatusP());//se crea una Linkedlist con los nodos
vecinos

uint8_t * neighborStatusP = s->getStatusP();

ORemoteVertex * neighborP;   //punter a nodo vecino
uint16_t nodeID = s->getID();//se obtiene ID del nodo en el que se está
ejecutando el código.
ORemoteVertex * nodeP = (n+(nodeID-1));
// pointer to this vertex

uint8_t neighborID;
float neighbor_fp;
float bp;
float bpinicial;
float bj;
float wi = float(s->getOutDegree());     // store out degree, the +1 is to
account for the self loops
float gi;
float wj;
bool receivedPacket;
bool txDone;             // create variable to keep track of broadcasts
int timeout = 100;       // create variable to keep track of broadcasts

uint16_t packetReceiveCount = 0;
uint16_t packetsLost = 0;
uint16_t packetReceived = 0;
```

- **Paso 2:**

Inicialización de flujos.

```
l->setInitialActiveFlows(nodeID, n);
```

Está es una de las partes más complejas del código ya que cada nodo tiene que obtener información de los valores de flujo de las ramas que conectan al nodo con sus vecinos y por lo tanto dependiendo de en qué nodo estemos o cuantas conexiones tenga con otros nodos, la información que se debe manejar es diferente.

El código de dicho método es el siguiente:

```
void LinkedList::setInitialActiveFlows(uint8_t i,ORemoteVertex *n) {

    node *tmp;
    tmp = _neighborHead;
    uint8_t j = 0;
    float fp0;

    while (tmp != NULL)
    {
        j = tmp->data;

            fp0 = 0.5*((n+j-1)->getActiveFlowMax() + (n+j-1)->getActiveFlowMin());
            (n+j-1)->setActiveFlow(fp0);

        Serial << "inicializamos flows  "<<fp0<<endl;

        tmp = tmp->neighborNext;
    }
}
```

Está función irá recorriendo los nodos vecinos e inicializando los flujos de las líneas que conectan a estos con el nodo local, según los límites establecidos. Teniendo en cuenta que la dirección de los flujos está definida como positiva en la dirección del nodo con mayor *"nodeID"* al menor .

- **Paso 3 :**

Inicialización de la generación y cálculo del balance inicial.

También se inicia el cronómetro de cada nodo y se establece la ventana de tiempo del nodo para la transmisión de información de manera que la compenetración entre los nodos sea posible.

```
gi = busgen*0.5*(s->getMax()+ s->getMin());
// calculamos generación inicial a partir de los valores límite. En el caso de
que se trate de un nodo de carga será cero al multiplicarlo por busgen.
```

```
s->setGi(gi); // se asigna el valor dentro de las propiedades del nodo local.

s->setActiveDemand(load); // se asigna el carga dentro de las propiedades del
nodo local.

 float Pd = load;

 bpinicial = gi - Pd - l->addActiveInitialFlows(nodeID,n); // se calcula el
balance inicial a partir de los valores iniciales calculados arriba.
 s->setActiveBalance(bpinicial); // se guarda el valor dentro de la propiedad del
nodo.
    uint8_t frame = 25;

    unsigned long start = (millis()-period);   // initialize timer
// create variable to store iteration start time

    srand(millis());
    uint16_t txTime = (rand() % (period - 2*frame)) + frame;
```

- **Paso 4 :**

Se entra en el bucle de iteraciones y se calcula la generación de la iteración correspondiente.

```
for(uint8_t k = 0; k < iterations; k++)
    {

        Serial<<"NUEVA ITERACION : "<<k+1<<endl;
        start = millis();    // initialize timer
        txDone = false;
        packetReceiveCount = 0;
        //compute bi
        if(busgen == 0)
        {
             s->setGi(0);
        }
           gi = s->getGi() - 0.5*s->getActiveBalance()/wi;

            if(gi > s->getMax()){

                gi = s->getMax();

            }else if(gi < s->getMin())
            {
                gi = s->getMin();
            }
                s->setGi(gi);
```

- **Paso 5 :**

Se entra en el bucle *"While"* que se ejecuta durante un tiempo igual al periodo. Dentro de este bucle se recibe la información enviada por nodos vecinos y con ella se actualizan los valores de flujo de las líneas de conexión.

Una vez actualizados estos se envía la información del balance a los nodos vecinos.

```
while( uint16_t(millis()-start) < period )
        {
            receivedPacket =
_waitForNeighborPacket(neighborID,FAIR_SPLITTING_HEADER,true,100);
            if ( receivedPacket && (*(neighborStatusP+neighborID-1) == 2) )
            {
                packetReceiveCount++;
                delay(5);
                *(neighborStatusP+neighborID-1) = 3;
                    bj = _getBjFromPacket();
                    neighborP = (n+(neighborID-1)); //asignamos la direccion de
memoria del nodo vecino del que esmos reciviendo informacion
                        if(k == 0)
                        {
                        bp =  bpinicial;
                        s->setActiveBalance(bp);
                        }
       if (nodeID<neighborID)
        {
neighbor_fp = (neighborP->getActiveFlow()-0.5*bj+0.5*(s->getActiveBalance())/wi);
                if (neighbor_fp > (neighborP)->getActiveFlowMax())
                 {
                   neighbor_fp = (neighborP)->getActiveFlowMax();

                 }else if (neighbor_fp < (neighborP)->getActiveFlowMin())
                 {
                 neighbor_fp = (neighborP)->getActiveFlowMin();
                 }
         }else
         {
            neighbor_fp=((-1)*(neighborP->getActiveFlow())-0.5*bj+0.5*(s-
            >getActiveBalance())/wi);
            neighbor_fp = -neighbor_fp;
              if (-neighbor_fp > (neighborP)->getActiveFlowMax())
              {
              neighbor_fp = -(neighborP)->getActiveFlowMax();

              }else if (-neighbor_fp <(neighborP)->getActiveFlowMin())
              {
               neighbor_fp = -(neighborP)->getActiveFlowMin();
```

```
                }
            }
               (neighborP)->setActiveFlow(neighbor_fp);
        }
      if(!txDone && (uint16_t(millis()-start) >= txTime))
       {
        txDone = true;
        _broadcastBalanceFeasibleFlow(s->getActiveBalance(),wi);
         delay(5);
            }

         }
```

- **Paso 6:**

En este último paso se recalcula el balance, se resetea la lista LinkedListStatus para la siguiente iteración, se aumenta en uno el número de iteraciones y una vez completadas todas las iteraciones se devuelve el valor de generación de potencia activa del nodo.

```
if(txDone)
        {
            float bp = s->getGi() - Pd - l->addActiveInitialFlows(nodeID,n);

                s->setActiveBalance(bp);
               Serial<<"recalculamos balance: "<<_FLOAT(bp,6)<<endl;
        }

        packetReceived += packetReceiveCount;
        packetsLost += (_G->getN() - packetReceiveCount - 1);

        packetReceiveCount = 0;

        l->resetLinkedListStatus(s->getStatusP());

    }
    return (s->getGi());
}
```

Dentro de esta parte cabe destacar el método de la clase LinkedList `addActiveFlows(nodeID,n),` el cual se encarga de sumar los fulos de todas las ramas que llegan al nodo local y funciona de la siguiente forma:

```
float LinkedList::addActiveFlows(uint8_t i, ORemoteVertex *n) {

    node *tmp;
```

```
    tmp = _neighborHead;
    uint8_t j = 0;
    float fp = 0;
    while (tmp != NULL)
    {
        j = tmp->data;                                  //get ID of neighbor
        if (i < j)
  fp = fp + ((n+j-1)->getActiveFlow()); //get active flow of link associated with
neighbor
        else if (i > j)
 fp = fp - ((n+j-1)->getActiveFlow());//get active flow of link associated with
neighbor
        tmp = tmp->neighborNext;
    }
```

PRINCIPALES PROBLEMAS DEL CÓDIGO:

Los principales problemas que se presentaron en la implementación del código fueron los siguientes:

1. **Información de flujos.**

   El primer problema que se presenta cuando se trata de pasar el algoritmo a lenguaje de computación, es como hacer que cada nodo obtenga y maneje la información sobre los flujos de las ramas que le conectan con sus nodos vecinos.

   Como se ha explicado en el paso 2 , esto se resuelve mediante el uso de la clase **LinkedList**, también explicada con anterioridad. Gracias a esta en vez de guardar la información en el propio nodo, esta se almacena en Objetos de la clase ORemoteVertex, y se accede a ellos mediante el uso de las funciones de la clase LinkedList.

   De la misma forma es importante también tener en cuenta que los flujos están definidos de forma que se consideran positivos en dirección de un nodo con menor ID a otro con ID superior. Por lo tanto, en el caso de que el ID de nuestro nodo sea superior al de su vecino se debe cambiar el signo.

2. **Sincronización para el envío de información.**

El segundo problema más evidente es como estructurar el envío de información entre los diferentes nodos para que el algoritmo se ejecute correctamente a lo largo de todas las iteraciones.

Como sabemos el algoritmo se desarrollará un determinado número de iteraciones, y en cada una de ellas cada nodo debe recibir la información del balance de potencia de sus nodos vecinos, recalcular los nuevos valores de generación, flujos y balance y así mismo enviar su información a sus vecinos.

Es importante tener en cuenta que cada nodo deberá enviar la información antes de que esta se actualice, y posteriormente actualizarla empleando la información recibida. Para conseguir la buena sincronización para el envío de información entre los nodos, se establece una ventana de tiempo para cada nodo, en la cual esté recibe información de sus vecinos, una vez finalizada esta franja de tiempo, enviará su información al resto de nodos, y finalmente se procede a el cálculo del nuevo balance.

Además, en cada iteración cuando el nodo recibe la información de uno de sus vecinos se cambia el valor de la variable de estado del nodo vecino para indicar que su información ha sido recibida, y evitar de esta forma recibir varias veces la información de un mismo nodo.

3. **Packet Drops.**

Lamentablemente una vez finalizado y tras realizar la prueba con los tres Arduinos que se mostrará más adelante los resultados, aunque muy próximos a los esperados no eran exactamente los correctos debido a que alguno de los paquetes de información enviados no llegaba a los nodos vecinos. Este tipo de

problema mencionado anteriormente es frecuente en las comunicaciones inalámbricas, sin embargo, este problema puede resolverse creando un algoritmo más complejo, que haga el programa resiliente frente a este tipo de fallos. Actualmente los ingenieros que forman parte del proyecto de la universidad de Illinois están trabajando en algoritmos que permitan al sistema seguir funcionando a pesar de este problema. Es por ello por lo que, debido a la falta de tiempo y conocimientos, la solución de este problema queda fuera de este trabajo.

4.4 PASO 3: IMPLEMENTACIÓN PARA RED DE 3 NODOS

El primer paso una vez se ha creado al algoritmo distribuido, es probar este en los Arduinos con la misma configuración de tres nodos que se empleó con el algoritmo centralizado. El objetivo de esto es principalmente comprobar que el algoritmo distribuido funciona correctamente antes de implantarlo en el banco de pruebas Typhoon Hil.

La prueba se ha hecho estableciendo un número de iteraciones igual a 40.

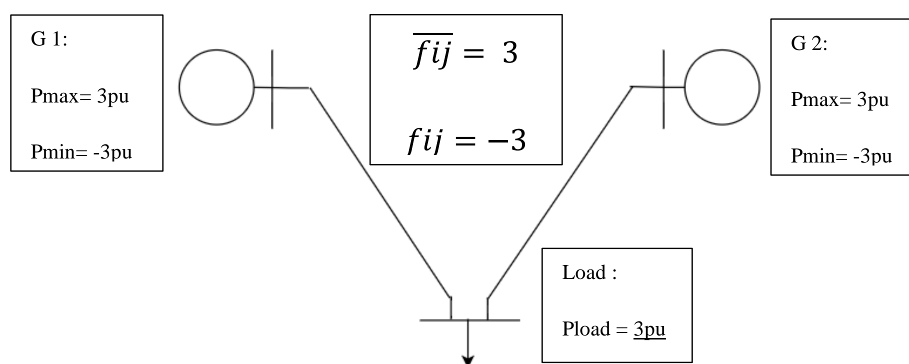


Ilustración 4-13:Red 3 nodos

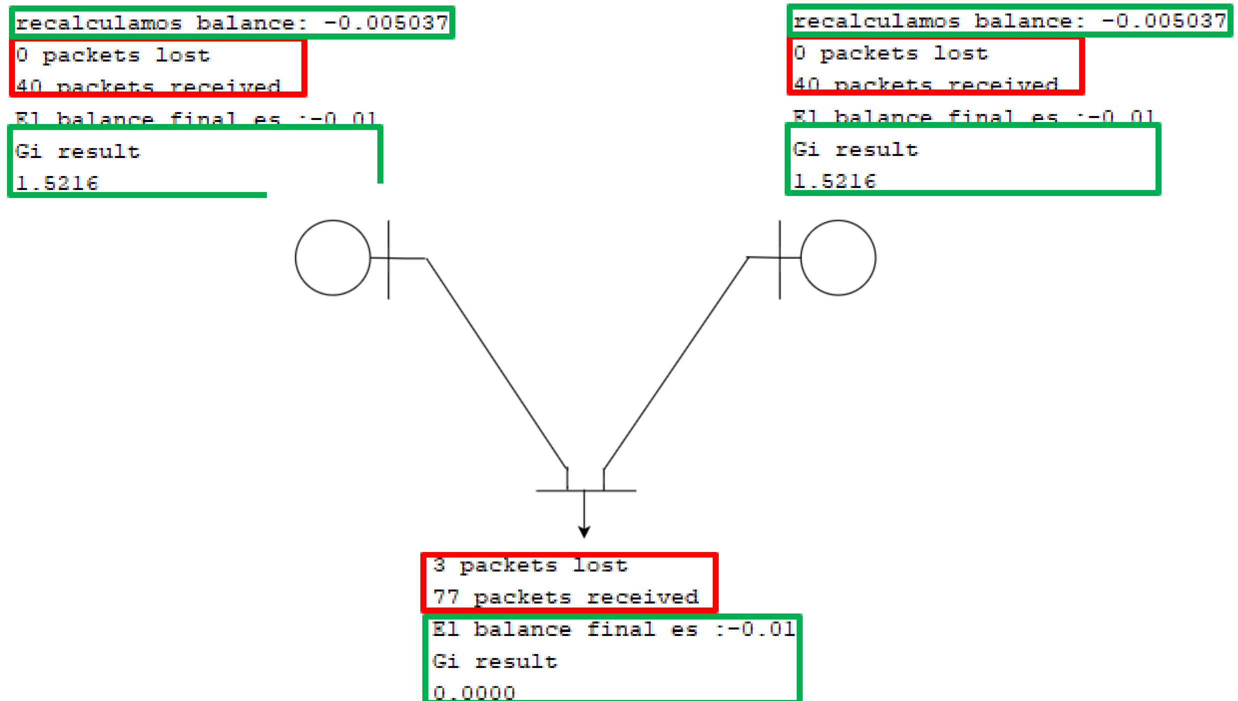


Ilustración 4-14:Resultados "Feasible Flow" sistema de 3 nodos distribuido

Se puede apreciar en los resultados como el nodo de demanda no ha recibido la información de sus dos nodos vecinos en tres ocasiones. Este problema es provocado por un error bastante común en las comunicaciones inalámbricas, conocido como "drop packets". Esto ocurre cuando uno o varios de los paquetes de información enviados no llega a su destino, se mide como un porcentaje de los paquetes perdidos respecto a los enviados.

Estos resultados obtenidos se analizan más en profundidad en el capítulo 5, y en el Anexo I se puede encontrar el código del Arduino del nodo de demanda empleado para esta prueba.

4.5 PASO 4: IMPLEMENTACIÓN EN TYPHOON HILL

Esta constituye la parte final del proyecto. Consiste en la implementación del código en el banco de pruebas Typhoon Hil que nos permitirá simular una microrred real para obtener resultados y extraer conclusiones a cerca de nuestro sistema de control distribuido.

En este caso la simulación se ejecutará en una red de seis nodos, tres de ellos generadores y los tres restantes nodos PQ. La estructura de la red que se va a testear es la siguiente:

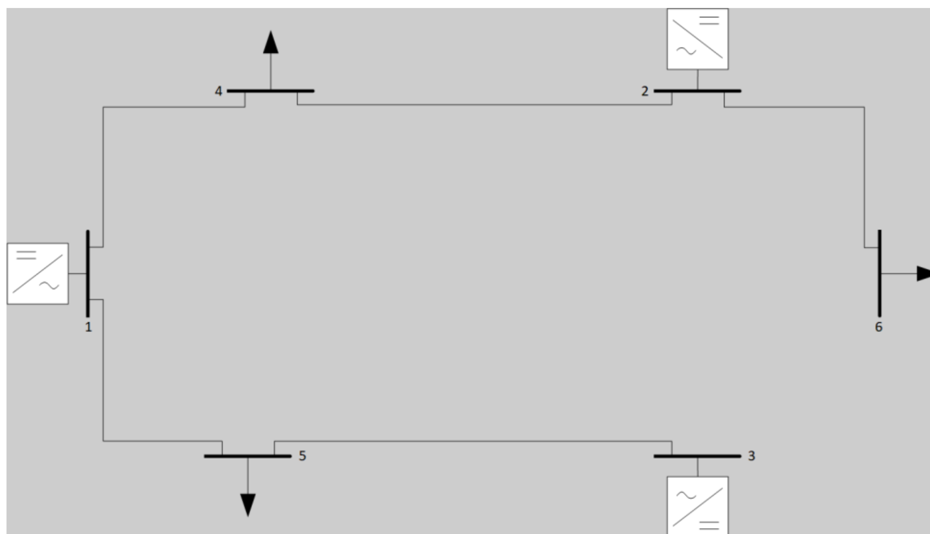


Ilustración 4-15:Sistema seis nodos

Los parámetros que caracterizan a la red son los siguientes:

Bus		Parámetros		
desde	pu	R(pu)	X(pu)	S(KVA)
1	4	2.3459	3.04623	200
1	5	2.7369	2.35838	200
4	2	2.2286	3.04623	200
2	6	2.3459	2.35838	200
5	3	2.2286	4.42195	200

Bus	Vnom(V)	S(KVA)
1	277	200
2	277	200
3	277	200
4	277	6.6
5	277	6.6
6	277	6.6

Tabla 3:Tablas valores sistema 6 nodos.

A partir de estos datos, escogemos como valores límites de potencia activa de los generadores a partir del diagrama P-Q que se muestra en la figura inferior, de manera que:

$$\boldsymbol{Base\ =\ 200KVA}$$
$$\boldsymbol{Pmax = 141.42\ =\ 0.707pu}$$
$$\boldsymbol{Pmin = -141.42 = -0.707pu}$$

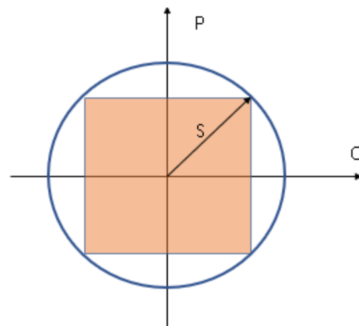


Ilustración 4-16: Área de operación

Por otro lado, como límites para los flujos de las líneas dejaremos un margen de operación de 30% en base a la potencia demandada, es decir un 30% de margen por arriba y por debajo. Teniendo en cuenta que las cargas van aumentando su potencia demandada en incrementos de 10kW emplearemos unos 20kw como la potencia máxima que alcanzarán las cargas y 10 kW como la mínima de forma que establecemos los siguientes límites de potencia para la simulación:

$$Plmax = 20 * 1.3 = 26KW = 0.13pu$$
$$Plmin = 10 * 0.7 = 7KW = 0.035pu$$

Antes de proceder a la realización de las pruebas será necesario configurar la red y el entorno de Typhoon Hil de manera que se podamos obtener los datos que necesitamos para la implementación de nuestros algoritmos del simulador, así como la implementación del código en los Arduinos para conseguir que la comunicación entre el simulador y cada uno de los Arduinos sea la correcta.

Para ello se añaden medidores que se encarguen de transmitir la información a los controladores, necesaria para la ejecución de los algoritmos.

En las dos figuras siguientes se muestra como serían el modelo de Typhoon centralizado y descentralizado para la red de 6 nodos.

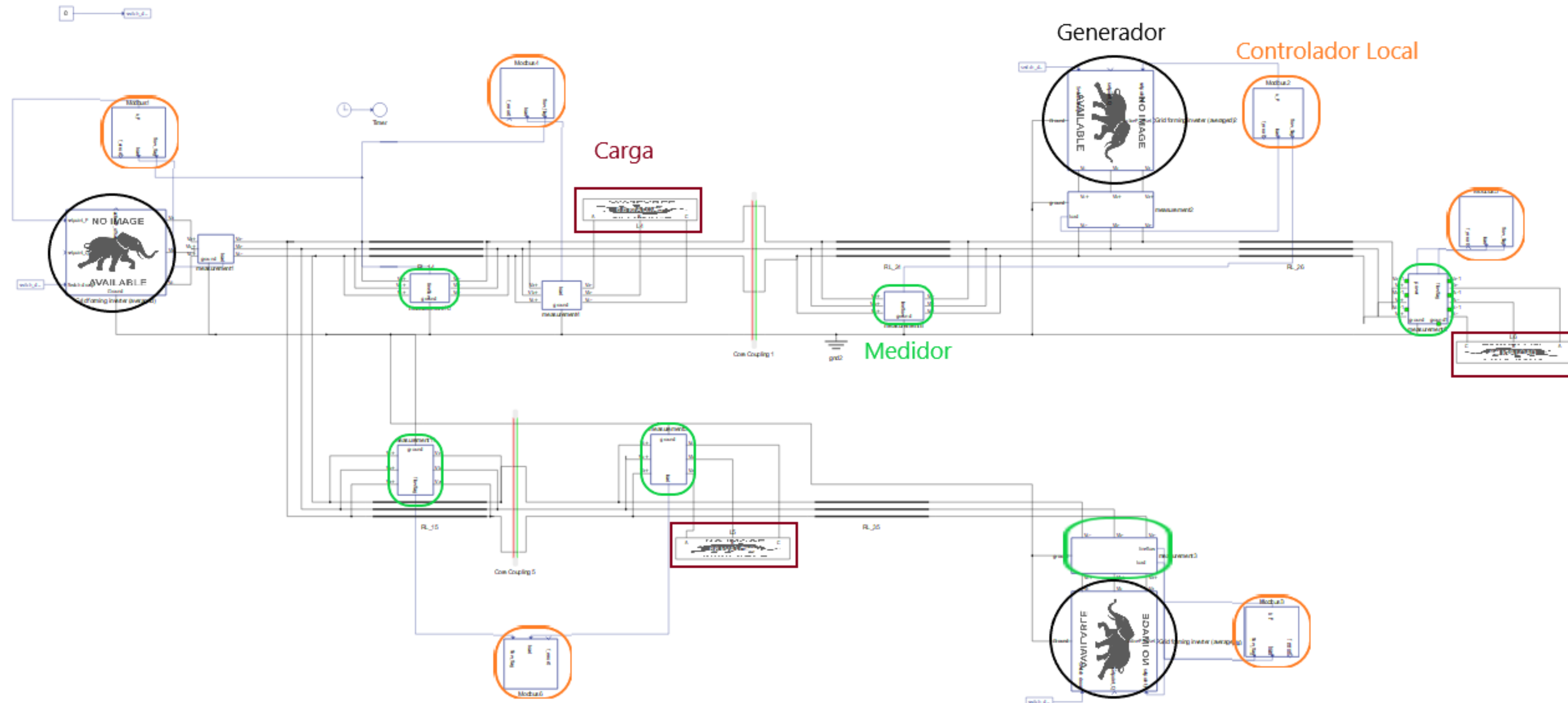


Ilustración 4-17:Modelo distribuido Typhoon

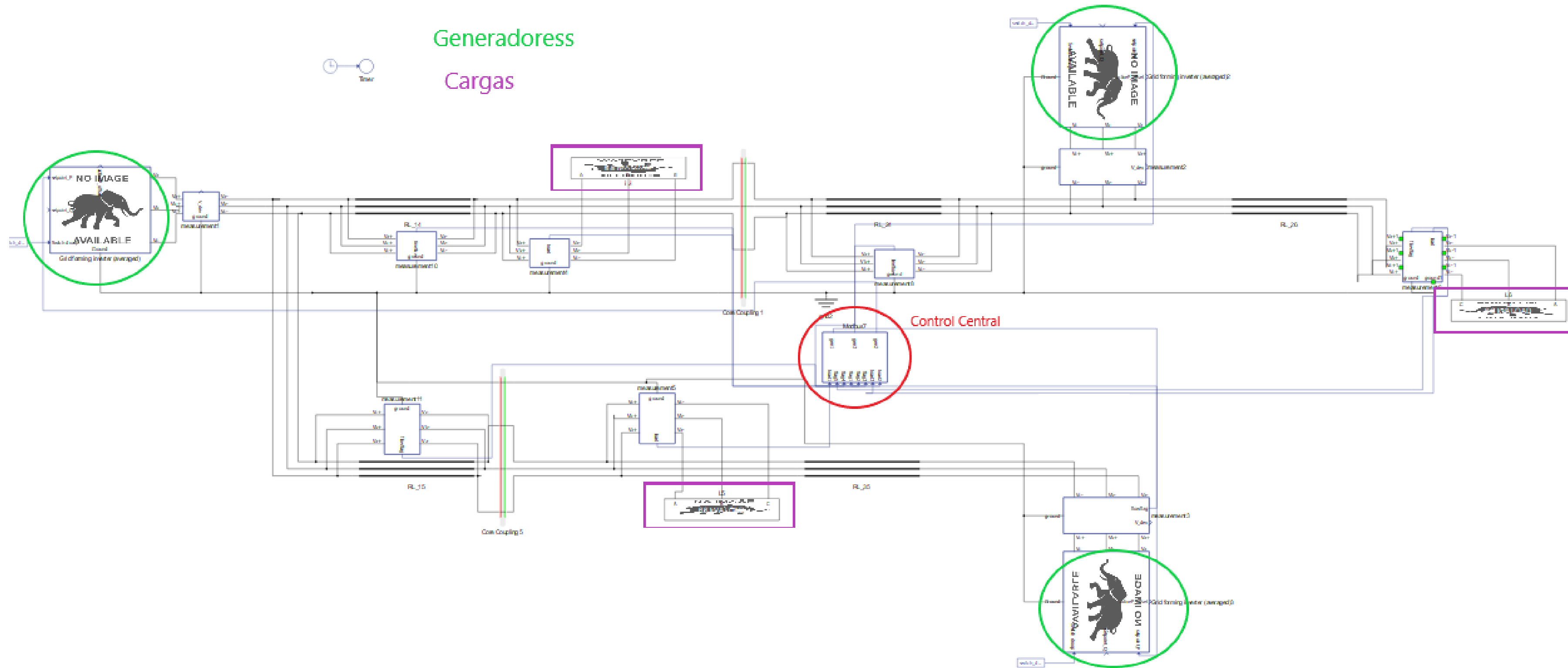


Ilustración 4-18:Modelo Centralizado Typhoon

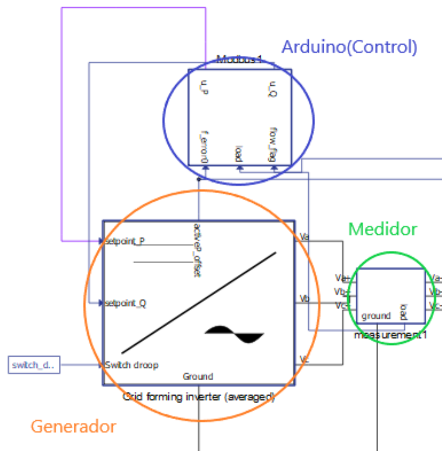


Ilustración 4-19:Modelo generador

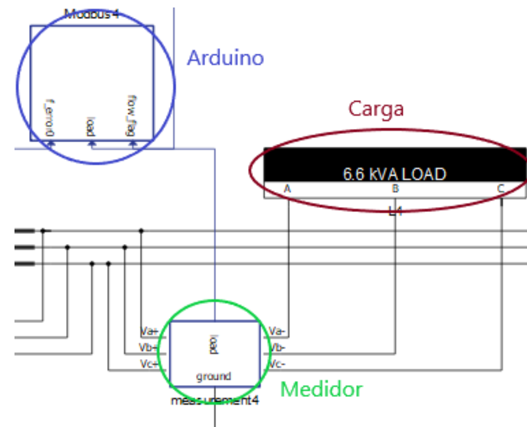


Ilustración 4-20:Modelo Carga

Modelo de las cargas:

El modelo empleado para la simulación de las cargas es el que se muestra a continuación en la siguiente figura. Como se puede ver se trata de una carga variable, y como ya se ha mencionado esta vería en saltos de 10KW.

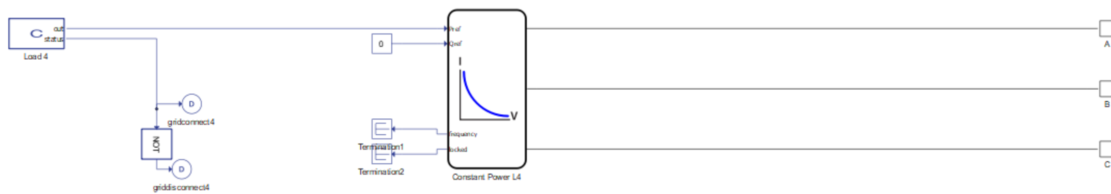


Ilustración 4-21:Carga Typhoon

Medidor:

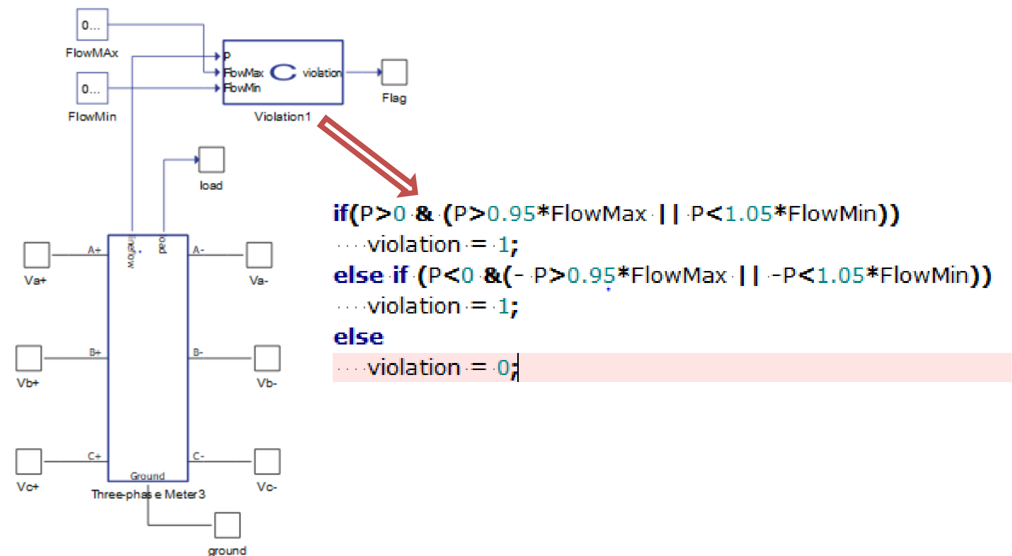


Ilustración 4-22:Medidor Typhoon

Modelo de un Arduino del sistema distribuido:

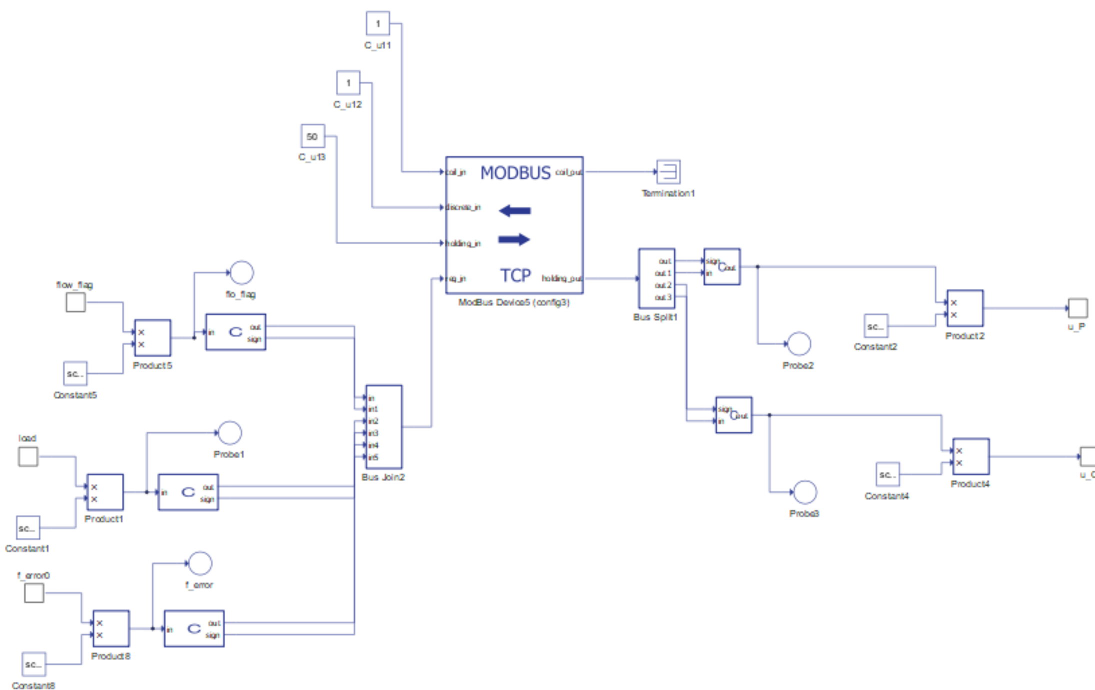


Ilustración 4-23:Controlador de uno de los nodos generadores

Modelo del Arduino de control sistema centralizado:

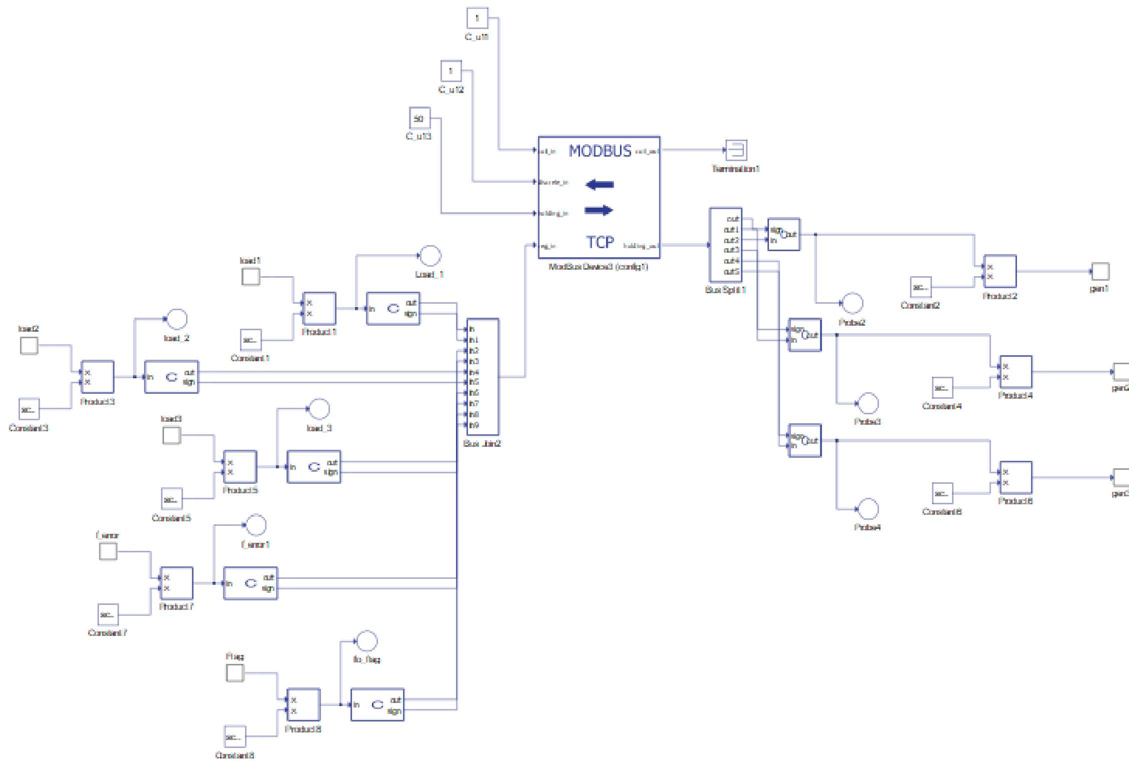


Ilustración 4-24:Controlador central sistema centralizado

Una vez el modelo en Thyphon está preparado se crea el código en Arduino que conectará el modelo en Thyphoon con los controladores para llevar a cabo el control de frecuencia secundario en toda la red.

El código completo del código en Arduino puede encontrarse en el Anexo I, aquí solo se presentará aquella parte que tiene relación directa con el control de frecuencia secundario.

A continuación, se presentará la forma en la que se ha se ha estructurado el código de Arduino para hacer posible la implementación correcta del control secundario de frecuencia.

Cualquier código de Arduino tiene la siguiente estructura :

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Ilustración 4-25:Estructura código Arduino

Una función **setup**() que solo se ejecuta una vez y la función **void loop**() que se ejecutará continuamente a lo largo del proceso de simulación.

Paso 1:

En primer lugar en la función **Void setup()**, que solo se ejecuta la primera vez que se entra en el programa, se deben sincronizar todos los Arduinos para que estos tengan sus tiempos sincronizados y el proceso de envío y captación de información funcione de forma acompasada y se llama a la función de “*Feasible Flow*” para que calcule los “set points” inciales.

Dentro de **void setup()** también se establecen cuáles serán los nodos vecinos de cada nodo local entre otras cosas.

```
void setup()  {
  Serial.begin(38400);
  Serial3.begin(38400);
  pinMode(cPin, OUTPUT);
  pinMode(sPin, OUTPUT);
  digitalWrite(cPin,HIGH);
  digitalWrite(sPin,HIGH);

  xbee.setSerial(Serial3); //Specify the serial port for xbee

    g.addInNeighbor(0x415786A9,12,0,0,0.0086,0.00462); // node 12

  g.configureLinkedList();
```

```
int NUMNEIGH = s.getOutDegree();//como se hace esto

  digitalWrite(cPin,LOW);
  digitalWrite(sPin,LOW);

 // initialize the ethernet device
  Ethernet.begin(mac, ip, gateway, subnet);   // start etehrnet interface
  for (int i=0;i<12;i++) {
     Mb.MbData[i] = 0;
  }

    if(!(a.isLeader()))
    {
      Serial.println("Still trying to sync");
      if(a.sync())
      {
        Serial.println("Communication Link established");
        Serial.println("c");
        digitalWrite(sPin,HIGH);
      }

    }
    if (a.isLeader())
    {
      Serial.println("Send letter s(r) to sync(resync)");
      while (Serial.available() == 0)
      {
        //simply makes the arduino wait until commputer sends signal
      }
      if(Serial.available())
      {
        Serial.println("got some letter");
        uint8_t b = Serial.read(); //enter the character 's'
        Serial.println(b);
        if (b == 'r')
          {
            a.setLeader(0);
          }
        if ((b == 's')||(b == 'r'))
        {
           Serial.println("got the s and about to sync");
           while(a.sync() == 0)
           {

           }
           Serial.println("Communication Link established");
           Serial.println("c");
           digitalWrite(sPin,HIGH);
           //ce = true;
        }
      }
    }
       receiveTyphoonData();
```

```
    bool feasibleflowFlag = Mb.MbData[4]*((-2*Mb.MbData[5])+1);
    int load = Mb.MbData[8]*((-2*Mb.MbData[9])+1);
    int f_error =  Mb.MbData[0]*((-2*Mb.MbData[1])+1);
      f_error0 =  float(f_error);
      load0 = float(load);
      load0 = load0/base;
      f_error0 =  f_error0/base;

      Serial.print("f error: ");
      Serial.println(float(f_error0),4);

      Serial.print("feasibleflowFlag: ");
      Serial.println(feasibleflowFlag);

      Serial.print("load value: ");
      Serial.println(float(load0),4);

      if (a.isLeader())
      {
          Serial.println("Begin feasible flow? (y/n)"); //let computer know you
want to begin ratio consensus
          while (Serial.read() != 'y')
          {
            //simply makes the arduino wait until commputer sends signal
          }
         Serial.println("got the y");
         u_f = a.feasibleFlowAlgorithm_RSL(1,30,600,0);
         Serial.println(u_f,4);
      }
      if (!(a.isLeader()))
      {
         u_f = a.feasibleFlowAlgorithm_RSL(1,30,600,0);
         Serial.println("Gi result");
            Serial.println(u_f,4);
      }
        if (u_f<0)
      {
        Mb.MbData[0]=1;
      }
      else
      {
        Mb.MbData[0]=0;
      }

       Mb.MbData[1]=base*abs(u_f);
       sendConsensusResults();
      a.resync();
}
```

Paso 2:

Se entra en la función **void loop(),** y lo primero que se hace es recoger la información obtenida de los medidores instalados en el modelo de la red, que en este caso son el **error de frecuencia** ,**valor de la potencia activa demanda por el nodo (load)** y **feasibleflowFlag,** esta variable devuelve un 1 en caso de que la línea en la que está el medidor haya detectado que el flujo a través de ella es superior al 95% de alguno de los límites. Devolverá valor cero en caso contrario. Esta señal será la que nos indique que nos hemos alejado lo suficiente de los *set-points* calculados inicialmente*.* Para ello se ha creado otro código que se encargará de enviar la señal al resto de Arduinos para que ejecuten *Feasible Flow cuando* alguno de ellos detecte que alguna de las ramas se está aproximando a sus límites.

Es importante tener en cuenta que esta aproximación se ha hecho con el fin de establecer un criterio de decisión para cuando es necesario recalcular los *"set-points"* mediante el uso de *"Feasible Flow"* que permita probar el algoritmo. Sin embargo, no se trata de un criterio exacto ni muy eficiente, como se verá más adelante en los resultados.

Posteriormente estas variables que solo pueden recibirse en forma de enteros son transformadas en valores *float* y divididos por una base para obtener los valores reales.

```
void loop() {
      receiveTyphoonData();
      int f_error =  Mb.MbData[0]*((-2*Mb.MbData[1])+1);
      int load = Mb.MbData[3]*((-2*Mb.MbData[4])+1);
      bool feasibleflowFlag = Mb.MbData[2]*((-2*Mb.MbData[3])+1);


      bool Flag;

      f_error0 =  float(f_error);
      //v_error0 =  float(v_error);
      load0 = float(load);
      load0 = load0/base;
      f_error0 =  f_error0/base;
      //v_error0 =  v_error0/base;
```

Paso 3:
Se ha creado otra función en el código de C++, en las carpetas de *OAgent* llamada *ShareFlag*, cuyo código se puede encontrar también en el Anexo I, junto con el código de otras de las funciones necesarias para la implementación de esta.

Para no entrar muy en detalle, lo principal es entender que está función devolverá el valor 1 en el caso de que alguno de los nodos de la red haya detectado que alguna de las líneas de conexión está por encima del 95% de los límites de potencia.

Para ello los nodos deberán intercambiar información de forma distribuida de nuevo.

En el sao de que la función devuelva un 1 se considerará entonces que nos hemos alejado los suficiente de los valores estables del sistema y se llamará de nuevo al algoritmo "*Feasible Flow*" que recalculará los valores de salida de los generadores que lleven al sistema a un funcionamiento estable de nuevo.

```
if (a.isLeader())
      {
        Serial.println("Begin Flag code? (y/n)");
        while (Serial.read() != 'y')
        {
          //simply makes the arduino wait until commputer sends signal
        }
           Flag = a.ShareFeasibleFlowFlag(10,200, feasibleflowFlag);
           Serial.println("Flag result");
           Serial.println(bool(Flag));
      }
      if (!(a.isLeader()))
      {
          Flag = a.ShareFeasibleFlowFlag(10,200, feasibleflowFlag);
           Serial.println("Flag result");
          Serial.println(bool(Flag));
      }

      if(Flag){
           if (a.isLeader())
          {
            Serial.println("Begin feasible flow? (y/n)");
            while (Serial.read() != 'y')
            {
           //simply makes the arduino wait until commputer sends signal
            }
                u_set = a.feasibleFlowAlgorithm_RSL(1,30,300,0);
                u_f = u_set;
```

```
        }
        if (!(a.isLeader()))
        {
        u_set = a.feasibleFlowAlgorithm_RSL(1,30,300,0);
          u_f = u_set;
        }
```

Paso 4:

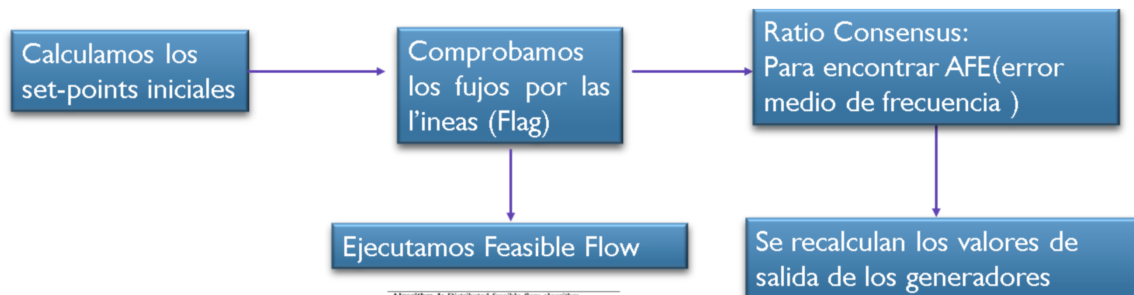
En caso de que ninguna de las líneas se encuentre cerca de los límites se procede a ejecutar el algoritmo distribuido *Ratio Consensus,* con el objetivo de hallar el valor medio del error de frecuencia. Si este valor supera el valor de ***eps_f*** (0.001) se recalcula el valor de salida de los generadores según las ecuaciones $(error = error + -1 * 0.707 * f_error1 \ y \ u_f = u_set + 0.7071 * error)$. Si por el contrario la diferencia entre el valor de frecuencia real y el de referencia no supera este valor, no se realiza cambio alguno en la generación.

Finalmente se envía el resultado del valor de salida de los generadores del Arduino al modelo de generador de Thyphoon.

```
      }else
      {

      if (a.isLeader())
      {
        Serial.println("Begin ratio consensus? (y/n)"); //let computer know you
want to begin ratio consensus
        while (Serial.read() != 'y')
        {
          //simply makes the arduino wait until commputer sends signal
        }

          Serial.println("got some letter");
          uint8_t o = Serial.read(); //enter the character 's'
          Serial.println(o);
          f_error1 = a.ratioConsensusAlgorithm(f_error0,D,30,300);
      }
      if (!(a.isLeader()))
      {
           f_error1 = a.ratioConsensusAlgorithm(f_error0,D,30,300);

      }
      }
      if(abs(f_error1) > eps_f)
      {
         error=error + -1*0.707*f_error1;
         u_f=u_set+0.7071*error;
```

```
          //Serial.println(u,4);
      }
// frequency controller code
      if(abs(f_error1) > eps_f)
      {
        error=error + -1*0.707*f_error1; // a partir del error de frecuencia
abotenido se calculan los nuevos set points de los generadores.
         u_f=u_set+0.7071*error;
         //Serial.println(u,4);
      }
      //Sending data
      if (u_f<0) // se analiza el signo de este valor para poder enviarlo
      {
        Mb.MbData[0]=1;
      }
      else
      {
        Mb.MbData[0]=0;
      }
      Mb.MbData[1]=base*abs(u_f);
```

La siguiente imagen muestra un esquema de la implementación del control secundario de frecuencia, recién explicado.



$$u_i[r] = u_i^* + \Delta u_i[r],$$
$$e_i[r+1] = e_i[r] + \kappa_i \Delta\overline{\omega}[r],$$
$$u_i[r] = u_i^* + \alpha_i e_i[r],$$

Ilustración 4-26:Esquema control secundario de frecuencia

Capítulo 5. RESULTADOS.

5.1 RESULTADOS SISTEMA 3 NODOS DISTRIBUIDO

En primer lugar, se analizarán los resultados de la prueba con Arduinos para el sistema de tres nodos con la siguiente configuración y un número de 40 iteraciones:

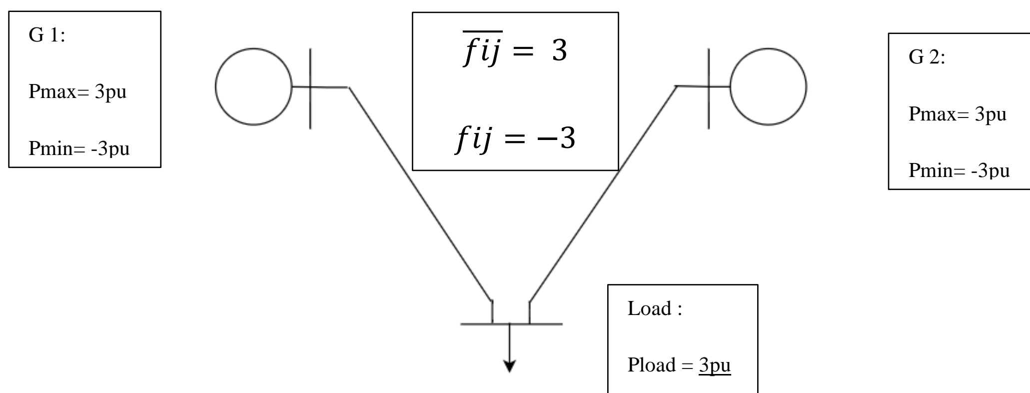


Ilustración 5-1:Red 3 nodos

En este caso los resultados que se obtienen son los siguientes:

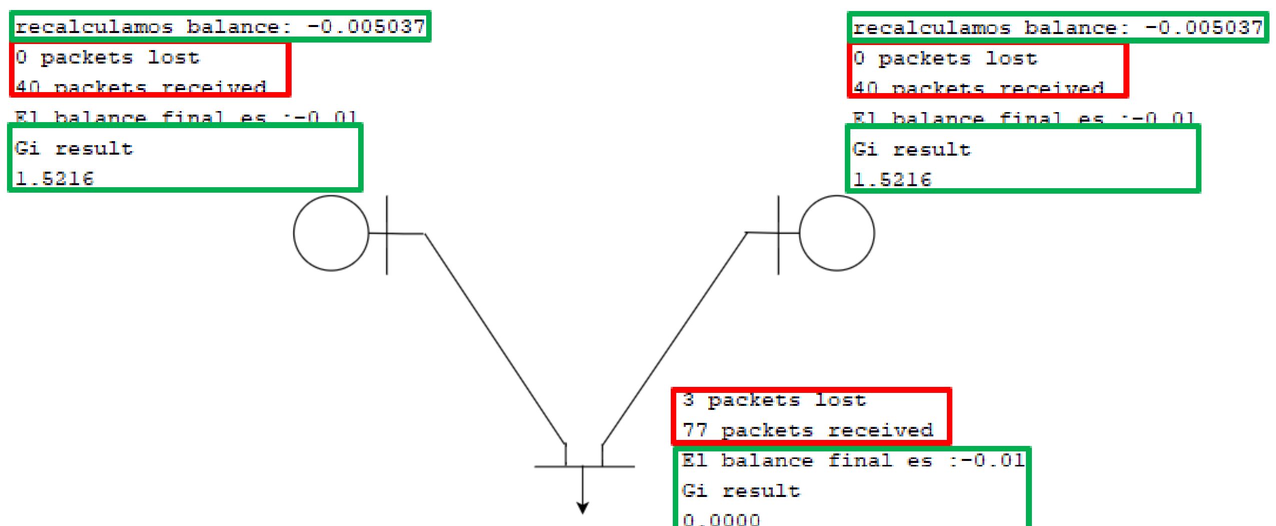


Ilustración 5-2:Resultados Arduinos sistema distribuido de 3 nodos

El valor de salida de los generadores obtenida ha sido igual a 1,5216 y un balance de -0.005037 y en la carga de 0.01 aproximadamente, como se puede apreciar el resultado es muy próximo al ideal de 1.5 de generación en cada generador y balance 0 en cada nodo, a pesar de que se ha perdido la información enviada en tres ocasiones.

El generador 1(nodo 9) vemos que ha recibido la información del nodo 14 (demanda) en todas las ocasiones, como puede verse en la imagen inferior:

```
recalculamos balance: -0.005037
0 packets lost
40 packets received
El balance final es :-0.01
Gi result
1.5216
```

Ilustración 5-3:Resultados Arduino(generador 1)

El generador 2 (nodo 11) vemos que al igual que el generador 1 ha recibido la información del nodo 14 (demanda) en todas las iteraciones, como puede verse en la imagen inferior:

```
recalculamos balance: -0.005037
0 packets lost
40 packets received
El balance final es :-0.01
Gi result
1.5216
```

Ilustración 5-4:Resultados Arduino (generador 2)

En el caso del nodo 14 (carga)si que tenemos pérdida de información en tres ocasiones, que tiene lugar en las iteraciones **1,19,38.**

Como se ve en la siguiente imagen de la primera iteración que tiene lugar en el nodo de demanda este, solo recibe la información del generador 1(nodo 9), pero no del generador 2 (nodo 10).

```
NUEVA ITERACION : 1
gen k :0.000000
Received from Node 9: -0.50
get active flow:0.92
get active flow:1.00
recalculamos balance: -1.083333
```

Ilustración 5-5:Iteración 1del nodo de carga

Lo mismo ocurre en la iteración número 19, donde solo se recibe la información del nodo 9.

```
NUEVA ITERACION : 19
gen k :0.000000
Received from Node 9: -0.04
get active flow:1.45
get active flow:1.39
recalculamos balance: -0.15612
NUEVA ITERACION : 20
gen k :0.000000
Received from Node 9: -0.03
Received from Node 10: -0.03
get active flow:1.46
get active flow:1.40
```

Ilustración 5-6:Iteración 19 nodo de carga

En la iteración 38 ocurre al contrario y es el nodo 10 del cual se recibe información, pero no del 9, como se ve a continuación.

```
NUEVA ITERACION : 38
gen k :0.000000
Received from Node 10: -0.00
get active flow:1.52
get active flow:1.46
recalculamos balance: -0.015195
NUEVA ITERACION : 39
gen k :0.000000
Received from Node 9: -0.00
Received from Node 10: -0.00
get active flow:1.53
get active flow:1.46
```

Ilustración 5-7:Iteración 38 nodo de carga

Tanto el número de "*packet drops*" como el momento en que estos se producen son completamente aleatorios, de forma que si se volviese a ejecutar el código se obtendrían resultados diferentes.

5.2 RESULTADOS SIMULACIÓN EN TIEMPO REAL

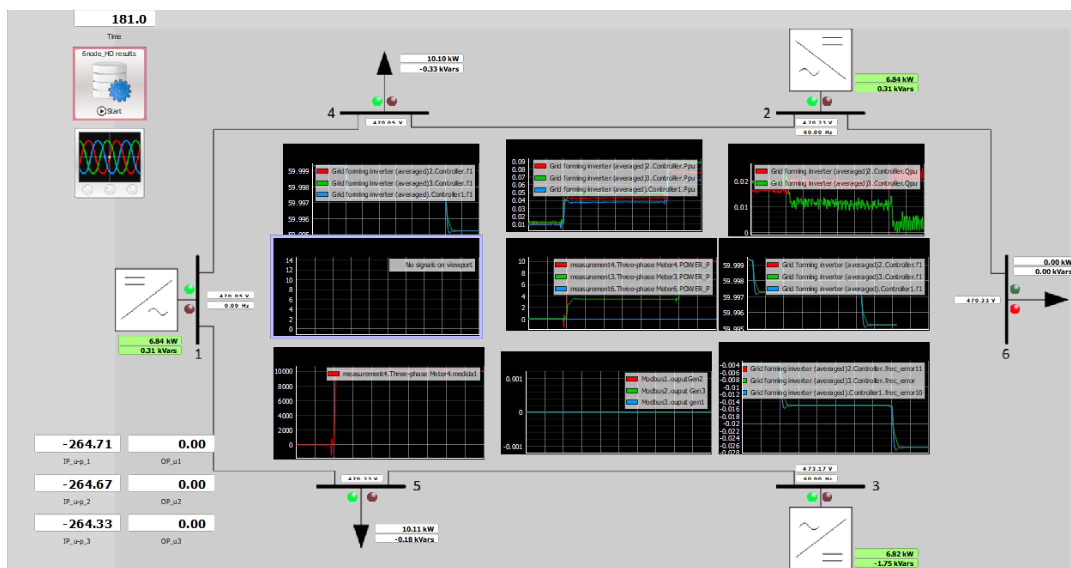


Ilustración 5-8:HIL SCADA

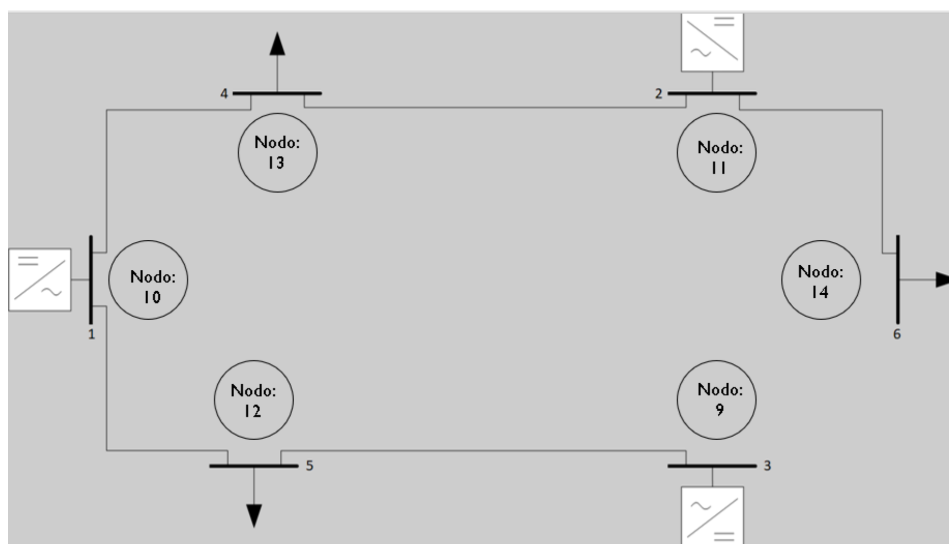
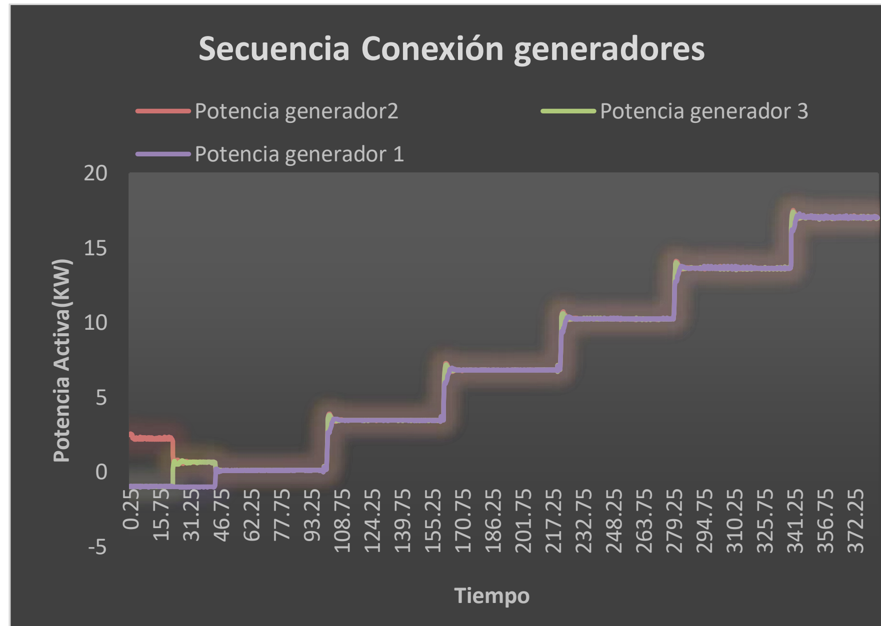


Ilustración 5-9:Red con los números de identificación de cada nodo

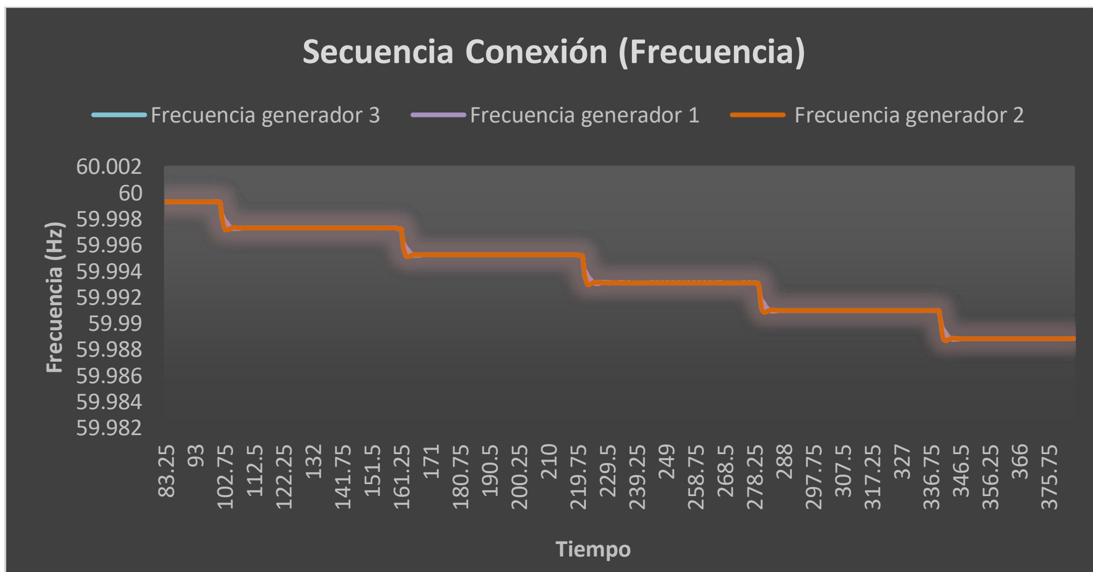
Una vez comenzamos la simulación en Typhoon se inicia un **arranque en negro** (Black start) que consiste en la restauración de un sistema eléctrico para el funcionamiento sin depender de la red externa de trasmisión de energía eléctrica. En las tres gráficas siguientes se muestran los valores de potencia activa generada por los generadores, potencia demandada por las cargas y la frecuencia de los generadores durante este proceso de inicialización del sistema.

La potencia en dichas gráficas se encuentra en valores unitarios, empleando como base 200KVA.

Gráfica 5-1: Black Start(Cargas)



Gráfica 5-2:Black Start(Generadores)

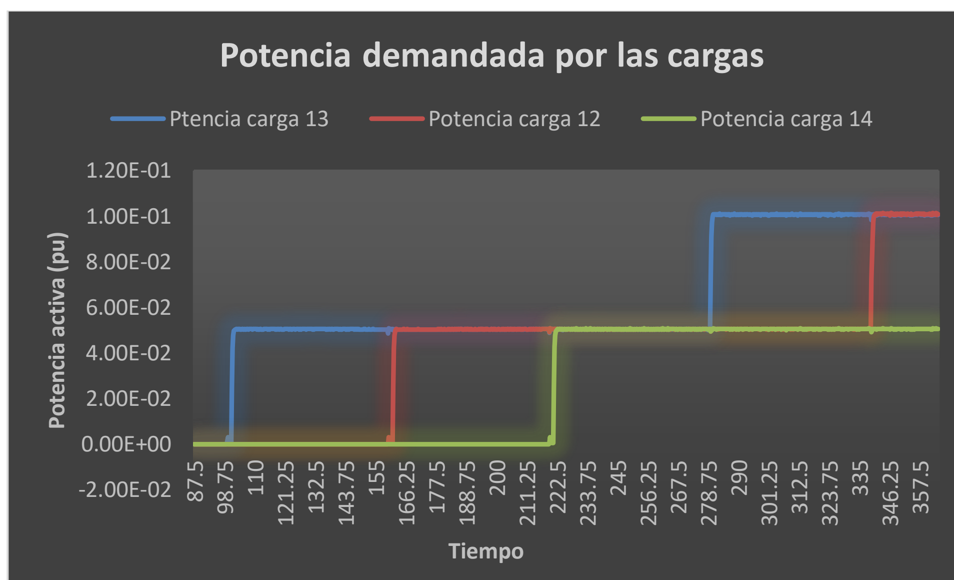


Gráfica 5-3: Black Start(Frecuencia)

La regulación primaria de frecuencia tiene como objetivo estabilizar la frecuencia a un valor constante cuando se produce un desequilibrio entre generación y demanda, y recuperar el equilibrio entre potencia generada y potencia consumida. Sin embargo, tras su actuación la frecuencia del sistema será estable, pero no igual al valor de

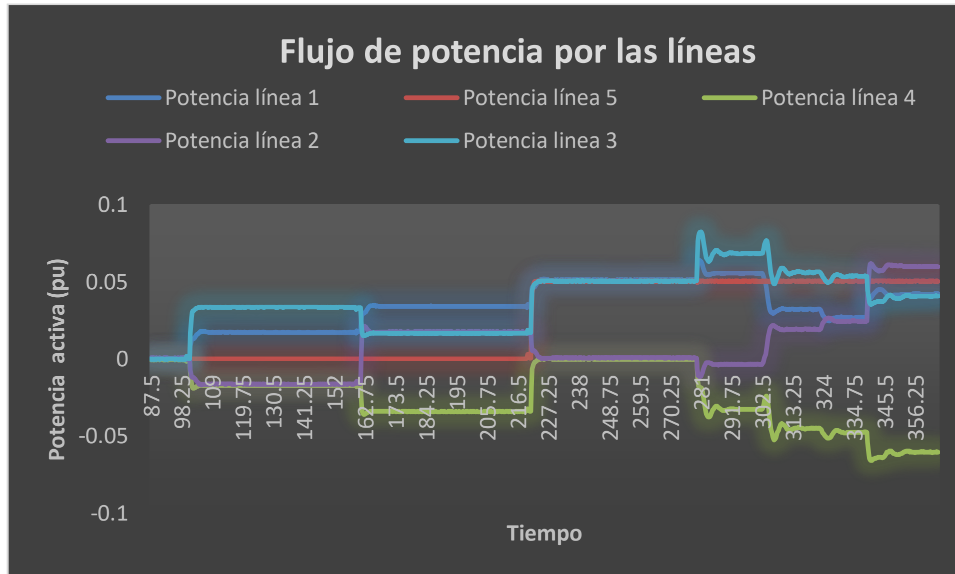
referencia (60Hz). Es aquí donde entra en juego la regulación secundaria, cuyo objetivo es eliminar el error en régimen permanente de la frecuencia.

Veamos ahora los resultados obtenidos para una de las simulaciones, que se muestran en las siguientes gráficas:



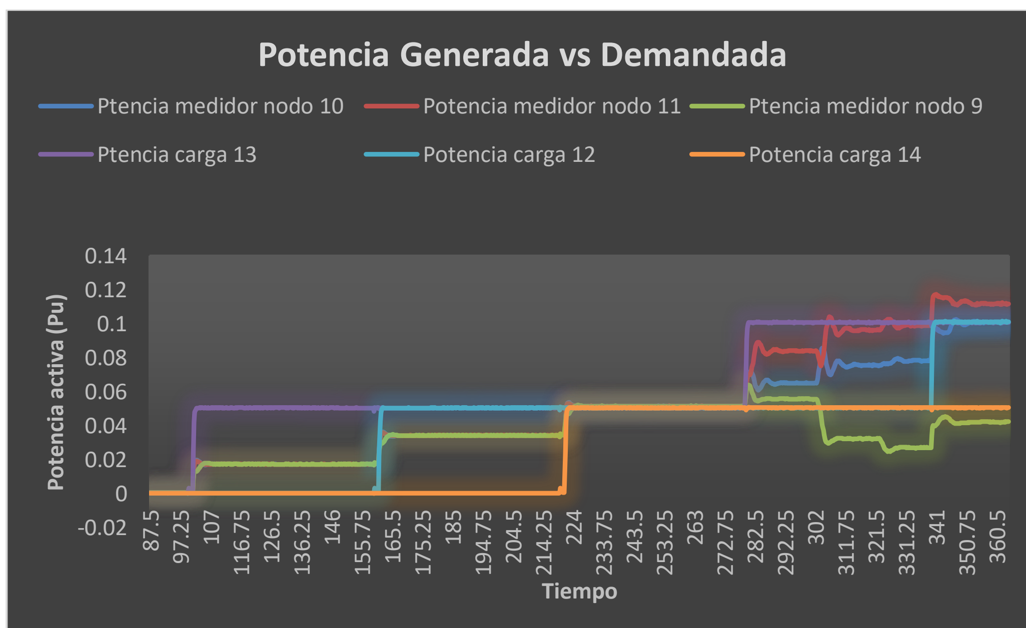
Gráfica 5-4:Potencia demandada por las cargas

En la gráfica superior se muestra cómo evoluciona la potencia demandada por las cargas. En un primer lugar durante el arranque las cargas se van conectando una a una a 10 KW, conforme avanza el tiempo estas irán aumentando su potencia demandada en saltos de 10 KW.



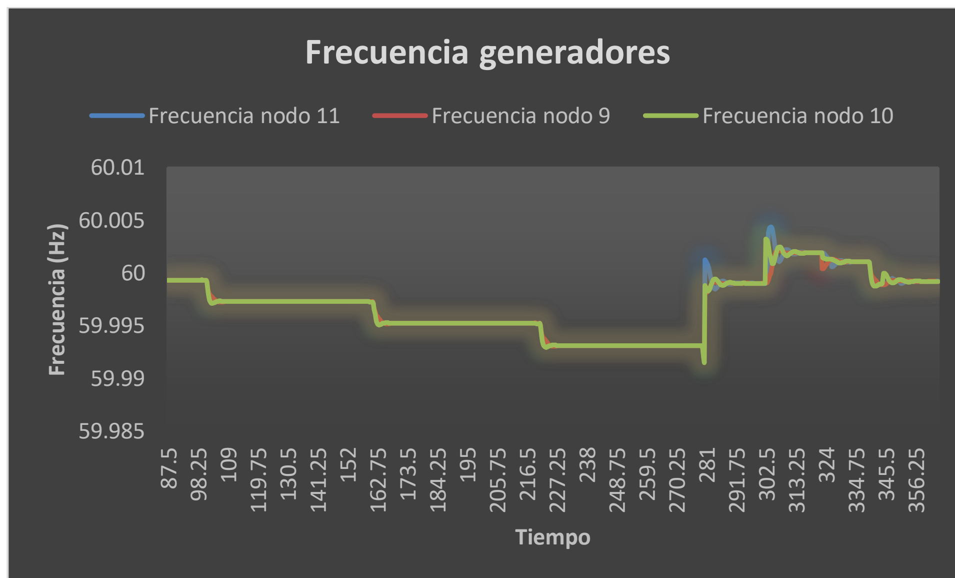
Gráfica 5-5:Flujo de potencia a través de las líneas de conexión

Si observamos la evolución de los flujos a través de las líneas se observa como en el segundo 281 aproximadamente, corresponde con el instante el cual el algoritmo ha recalculado la generación de los generadores y que se ha ejecutado una vez todas las cargas estaban conectadas.

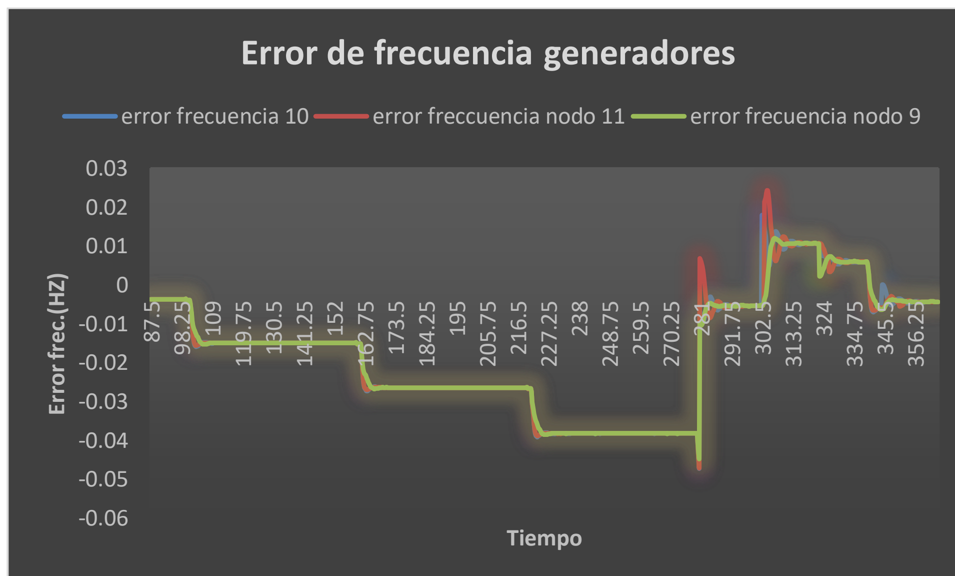


Gráfica 5-6:Potencia demandada frente a la generada

En la imagen superior también se aprecia que es una vez todas las cargas han sido conectadas y se ha iniciado el control de frecuencia cuando se ve un trazo más irregular en la generación.



Gráfica 5-7:Frecuencia de los generadores



Gráfica 5-8:Error de Frecuencia

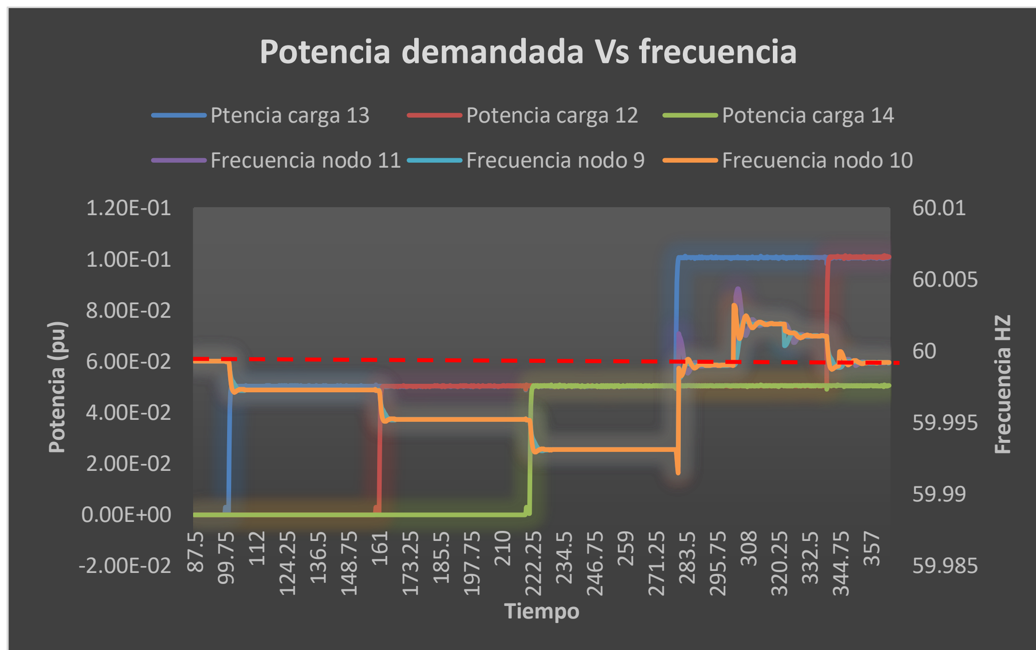


Ilustración 5-10:Potencia demandada Vs frecuencia

En la gráfica superior se presentan superpuestos los valores de carga junto con la frecuencia.

Es interesante ver como durante el arranque a medida que las cargas se van conectado la frecuencia disminuye para estabilizarse a un valor inferior una vez la potencia generada por los generadores iguala a la demandada.

También se puede apreciar con mucha claridad como en el segundo 281 se produce un gran salto hacia arriba en la frecuencia, que eleva su valor de nuevo a los 60HZ y como se ha mencionado anteriormente corresponde con el instante en el que se ha recalculado el valor de generación empleando el algoritmo “Feasible Flow”. Dado que se ha establecido un tiempo de duración del periodo de 300 milisegundos y un número de iteraciones de 50 el tiempo de duración del algoritmo completo es de 15 segundos.

El siguiente salto hacia arriba de la frecuencia se debe a que se ha ejecutado de nuevo el código “Feasible Flow” cuando probablemente no fuese necesario. Esto se debe al criterio establecido que tiene en cuenta los límites a través de las ramas . Por otro

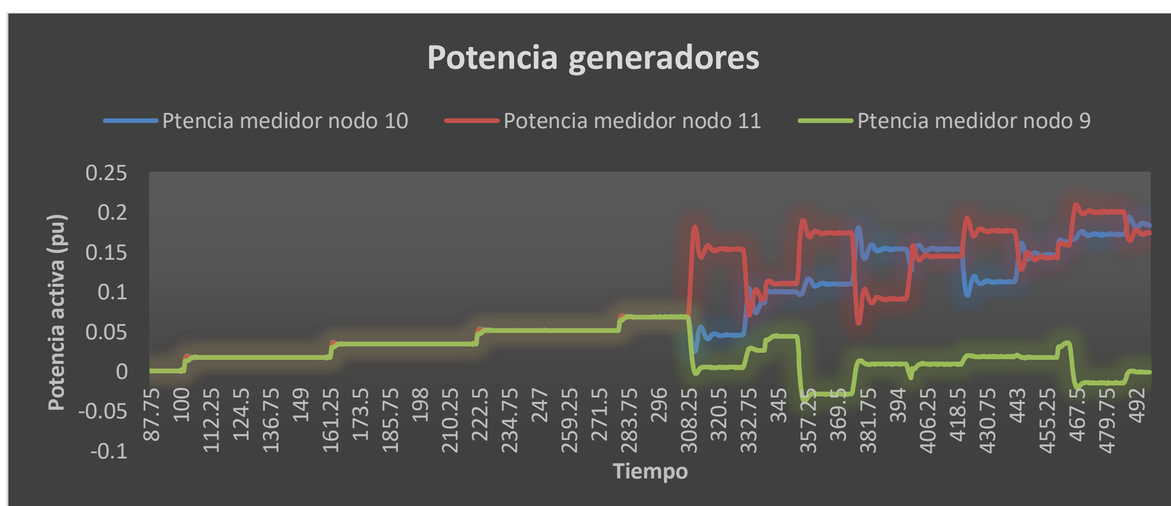
lado debido al problema ya mencionado de los "packet Drops" y tal vez a que el número de iteraciones no es suficiente, se obtiene un valor de frecuencia más elevado que su valor ideal.

En cualquier caso, en el siguiente capítulo se expondrán los principales fallos y puntos a mejorar que se han detectado tras la realización de las simulaciones.

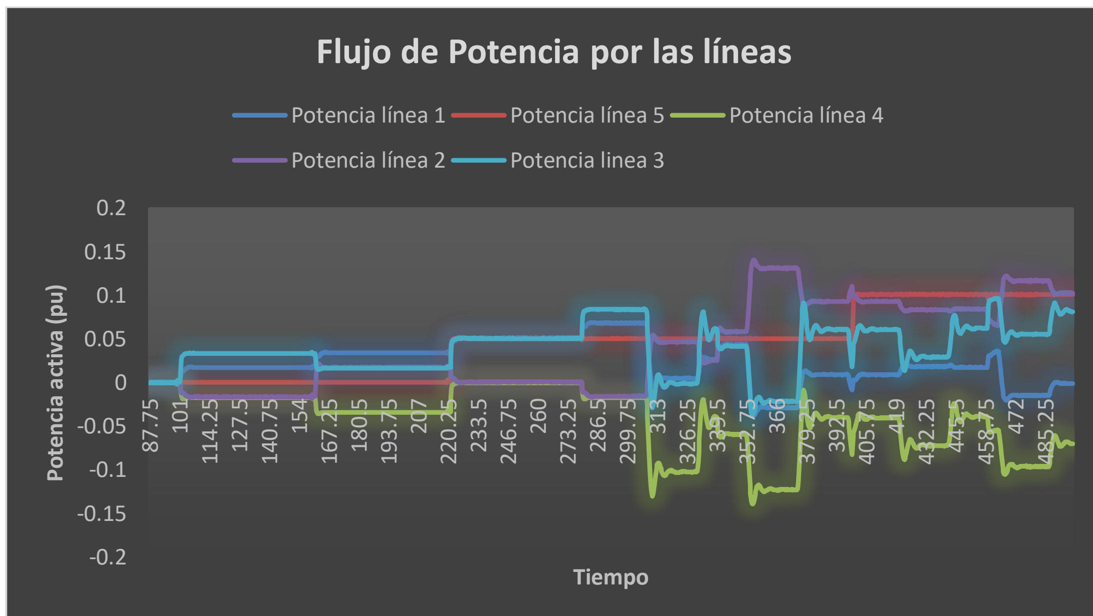
A continuación, se muestran los resultados obtenidos para tres simulaciones más:

- **Simulación 2:**

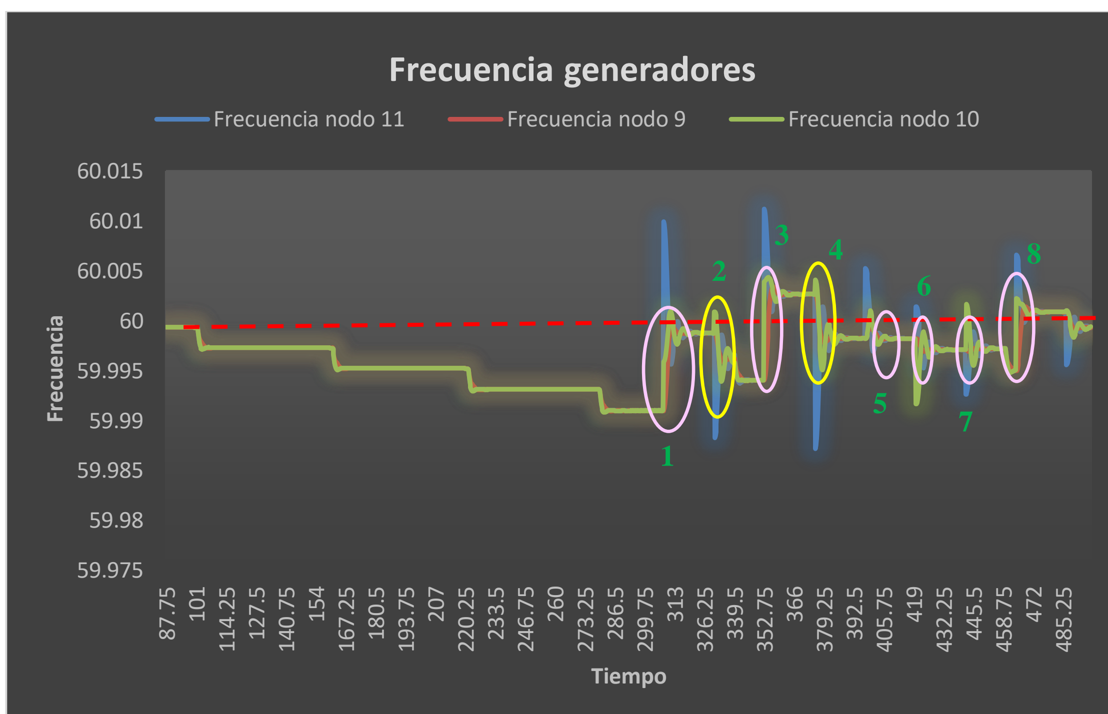
Gráfica 5-9:Potencia demandada. Simulación 2



Gráfica 5-10:Potencia generada. Simulación 2



Gráfica 5-11:Flujo de potencia activa a través de las líneas de conexión. Simulación 2



Gráfica 5-12:Valores de Frecuencia de los generadores. Simulación 2

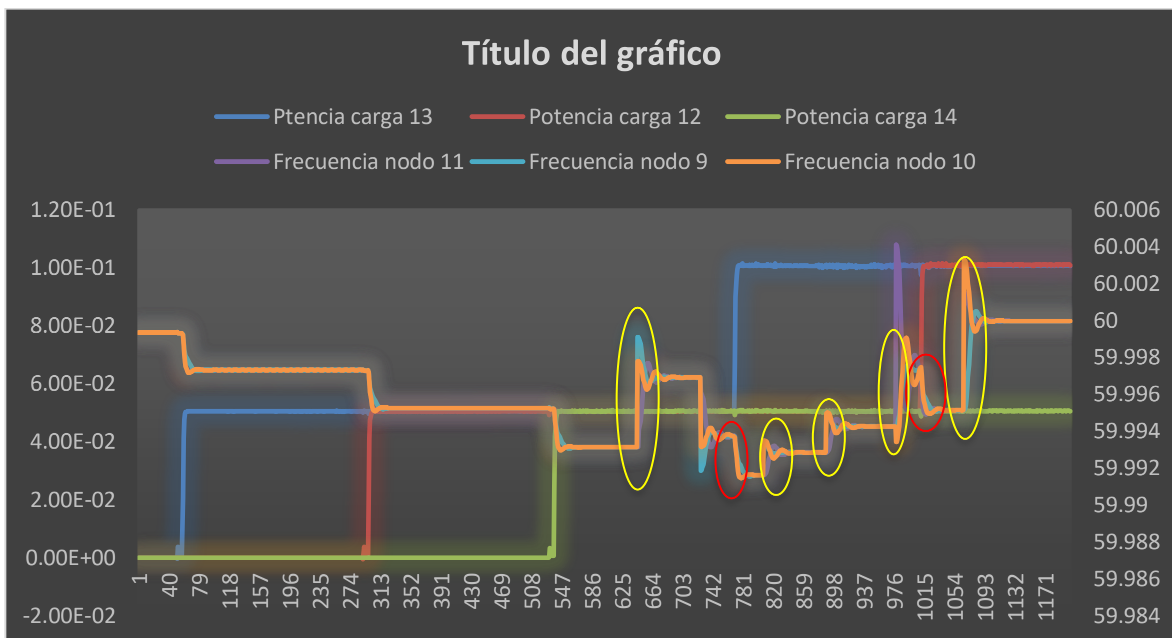
1. En el punto 1 es cuando se ha terminado de ejecutar el algoritmo “Feasible Flow” que llevar la frecuencia a un valor más cercano a los 60 Hz

2. Los puntos 3 y 4 son caídas de frecuencia que se producen debido al salto de 10 kW a 20 kW de dos de las cargas del sistema.

3. Los puntos 3 y 8 también son los instantes donde ha terminado la ejecución de "Feasible Flow".

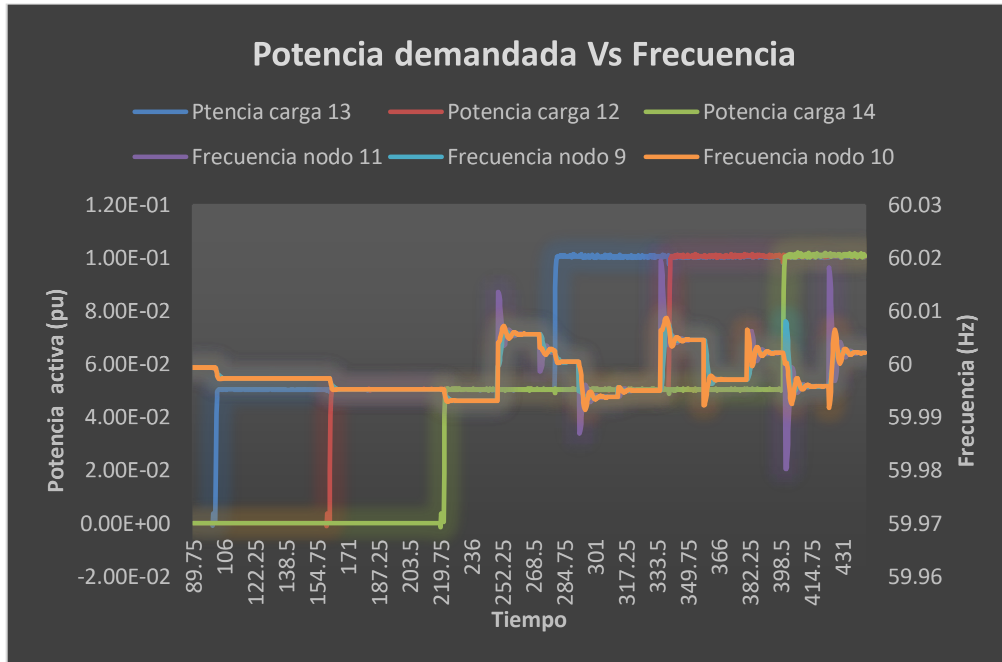
4. Los puntos 5,6 y 7 son saltos más pequeños de frecuencia donde se ha empleado el error de frecuencia para determinar los nuevos valores de salida de los generadores.

- **<u>Simulación 3:</u>**



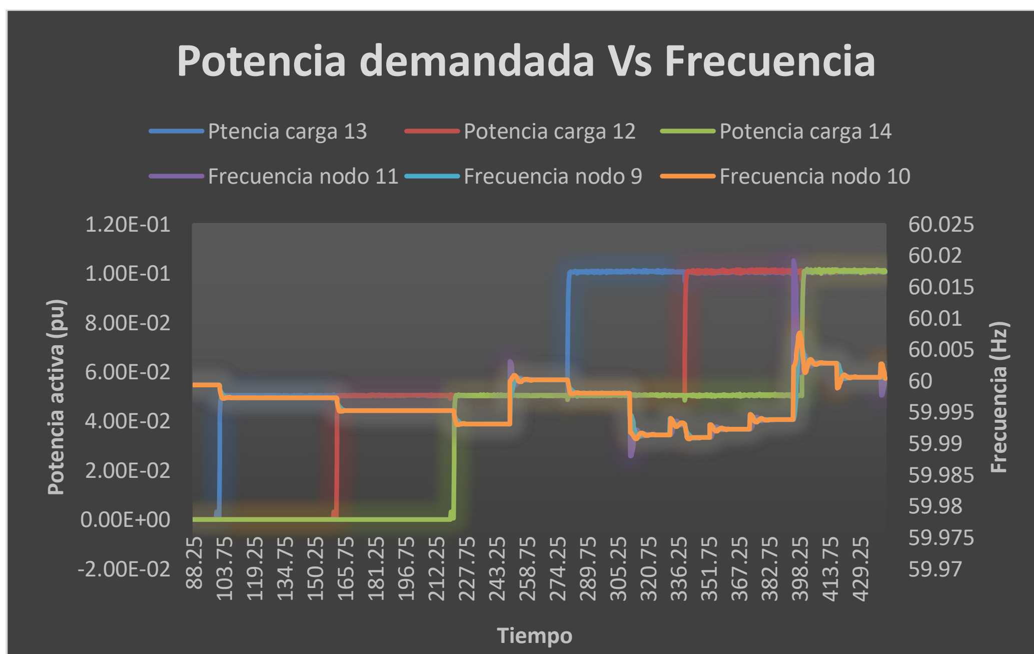
Gráfica 5-13:Potencia Vs Frecuencia. Simulación 3

- **Simulación 4:**



Gráfica 5-14:Potencia Vs Frecuencia. Simulación 3

- **Simulación 5:**



Gráfica 5-15:Potencia Vs Frecuencia. Simulación

Capítulo 6. ANÁLISIS DE RESULTADOS

Las conclusiones y observaciones más destacables a las que se llega tras la realización de las simulaciones son:

4. La aproximación echa para determinar cuándo debe ejecutarse el algoritmo que permite recalcular los "set-points" de los generadores, que se basa en comparar los flujos de potencia a través de las líneas con sus valores límite, posiblemente no sea la más adecuada.

Dicha aproximación ha sido empleada con el objetivo de establecer un criterio para probar el funcionamiento y estructura del control, más que para establecer correctamente el punto exacto en el cual el algoritmo "Feasible Flow " este debe ejecutarse . Es por ello por lo que los resultados obtenidos nos son exactamente los ideales y se debe establecer un criterio más preciso, como el explicado en capítulo 3.

5. Otro posible fallo que se puede encontrar en cuanto a la ejecución del propio "Feasible Flow " son los ya mencionados "Drop packets".

6. Por último la falta de precisión que pudiese existir en el "Feasible Flow " puede ser también relacionada con el número de iteraciones necesario para asegurar la máxima precisión. Pero como se explicó también anteriormente esto se resuelve empleando el algoritmo "*Max-Min Consensus*", que permitirá encontrar el número óptimo de iteraciones necesarias que establece la mejor relación entre tiempo de ejecución y precisión.

En lo que se refiere a los algoritmos distribuidos, se han llegado a las siguientes conclusiones:

4. Uno de los principales problemas de los algoritmos distribuidos es que debido al tiempo que lleva la ejecución de estos, el tiempo de reacción es mucho más prolongado.

Además, si durante la ejecución del algoritmo se produce otra variación en la carga, los resultados de este no serán correctos y se tendrá que volver a ejecutar retrasando aún más el cálculo de los valores correctos.

5. El otro fallo que los sistemas distribuidos poseen frente a los centralizados es la precisión. Ya que esta es más pobre en los primeros.

6. Otro problema que se ha presentado en la ejecución de este control han sido los problemas de sincronización entre los diferentes controles de cada nodo.
   En alguna ocasión, uno o varios de los Arduinos ha presentado problemas a la hora de sincronizarse, haciendo necesario la repetición de la simulación.
   Seguramente con equipos más sofisticados se pondría fin a este problema, sin embargo, puede ser un problema para tener en cuenta especialmente si existe un gran número de nodos en la red.

Capítulo 7. ODS

En este punto analizaremos la relación entre este proyecto con los objetivos para el desarrollo sostenible y como este trabajo junto con futuras investigaciones en esta área puede contribuir a la consecución de estos.

Este proyecto está fuertemente relacionado con 2 de los 17 ODS.

OBJETIVO 7: GARANTIZAR EL ACCESO A UNA ENERGÍA ASEQUIBLE, SEGURA, SOSTENIBLE Y MODERNA PARA TODOS.

La importancia del papel de la energía en la actualidad es innegable, es por ello que es vital apoyar iniciativa que aseguren el acceso universal a la energía, mejoren el rendimiento y aumenten el uso de renovables para combatir el cambio climático, como persigue el objetivo número 7 de los ODS.

Las metas que incluye este objetivo que están relacionadas de alguna forma con este proyecto son:

7.1 De aquí a 2030, garantizar el acceso universal a servicios energéticos asequibles, fiables y modernos.

7.2 De aquí a 2030, aumentar considerablemente la proporción de energía renovable en el conjunto de fuentes energéticas.

7.3 De aquí a 2030, duplicar la tasa mundial de mejora de la eficiencia energética.

7.a De aquí a 2030, aumentar la cooperación internacional para facilitar el acceso a la investigación y la tecnología relativas a la energía limpia, incluidas las fuentes renovables, la eficiencia energética y las tecnologías avanzadas y menos contaminantes de combustibles fósiles, y promover la inversión en infraestructura energética y tecnologías limpias.

Claramente el ODS número 7 es el que mayor relación con este proyecto, ya que como se ha visto, este tipo de microrredes y sistemas de control son una gran opción para la **inclusión de las energías renovables** en el sistema y en consecuencia reducción de las **emisiones contaminantes**, así como mejora de la **eficiencia** y **costes** energéticos previsibles a largo plazo y mejora de la fiabilidad.

Especialmente el desarrollo de los sistemas de decisión distribuidos puede ser decisivos para le mejora de la resiliencia de las redes.

OBJETIVO 9: INDUSTRIA, INNOVACIÓN E INFRAESTRUCTURAS.

Este Objetivo se centra en el desarrollo de infraestructuras fiables, sostenibles y de calidad con el objetivo de mejorar la calidad de vida y la creación de empleo. Para ello será necesario disponer de las infraestructuras adecuadas.

Las diferentes metas del objetivo 9 más relacionadas con el proyecto son:

9.1 Desarrollar infraestructuras fiables, sostenibles, resilientes y de calidad, incluidas infraestructuras regionales y transfronterizas, para apoyar el desarrollo económico y el bienestar humano, haciendo hincapié en el acceso asequible y equitativo para todos.

9.4 De aquí a 2030, modernizar la infraestructura y reconvertir las industrias para que sean sostenibles, utilizando los recursos con mayor eficacia y promoviendo la adopción de tecnologías y procesos industriales limpios y ambientalmente racionales, y logrando que todos los países tomen medidas de acuerdo con sus capacidades respectivas.

9.5 Aumentar la investigación científica y mejorar la capacidad tecnológica de los sectores industriales de todos los países, en particular los países en desarrollo, entre otras cosas fomentando la innovación y aumentando considerablemente, de aquí a 2030, el número de personas que trabajan en investigación y desarrollo por millón de habitantes y los gastos de los sectores público y privado en investigación y desarrollo.

El desarrollo de las microrredes nos permitirá crear un nuevo sistema de infraestructuras modernas y sostenibles que permitirán la utilización de los recursos con mayor eficiencia, así como permitir la creación de nuevos puestos de trabajo y oportunidades educativas.

Aunque de forma menos evidente que en el caso de los objetivos 9 y 7, el objetivo número 11 de los ODS se encuentra también relacionado de alguna forma con este proyecto.

OBJETIVO 11: LOGRAR QUE LAS CIUDADES Y LOS ASENTAMIENTOS HUMANOS SEAN INCLUSIVOS, SEGUROS, RESILIENTES Y SOSTENIBLES.

Cuyas metas puedan estar relacionadas con este proyecto son:

11.6 De aquí a 2030, reducir el impacto ambiental negativo per cápita de las ciudades, incluso prestando especial atención a la calidad del aire y la gestión de los desechos municipales y de otro tipo.

11.b De aquí a 2020, aumentar considerablemente el número de ciudades y asentamientos humanos que adoptan e implementan políticas y planes integrados para promover la inclusión, el uso eficiente de los recursos, la mitigación del cambio climático y la adaptación a él y la resiliencia ante los desastres.

El desarrollo de las microrredes puede tener un impacto fundamental para la reducción del impacto medioambiental de las ciudades ya que permitirá un uso mucho más eficiente de los recursos y la implantación de las renovables, reduciendo así las emisiones nocivas

Capítulo 8. PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA.

Actualmente países de todo el mundo están explorando el potencial de las microrredes con el objetivo de desarrollar una red menos centralizada, estable y menos vulnerable a incidentes e interrupciones.

Los beneficios que pueden llegar a tener las microrredes en el futuro incluyen no solo el beneficio económico, sino también: ahorro energético, reducción de emisiones, reducción de perdidas, mejora del rendimiento y seguridad del sistema …

De forma que un análisis económico del impacto de las microrredes debería tener en cuenta además del beneficio financiero, también las repercusiones económicas globales que se derivan de las mejoras anteriormente mencionadas. Obviamente se trata de un análisis muy complejo, y en este caso el análisis se centrará en los beneficios de los sistemas de control distribuidos más que en las microrredes en general.

En este capítulo se presentará un análisis del coste del proyecto que se está llevando a cabo por la universidad de Illinois y se explicará de forma cualitativa el beneficio que este proyecto podría generar.

La razón por la cual no se hará un estudio cuantitativo de los posibles beneficios, es debido a que en el estado primario actual de desarrollo es muy difícil poner números concretos que puedan representar de forma adecuada el impacto real de un primer proyecto de investigación como este.

8.1 COSTE DEL PROYECTO

	Año 0	Año 1	Año 2	Año 3	Año 4
Gastos Equipo:					
Typhoon HIL Microgrid Testbed	100000				
Arduinos	426				
Material de Computación:	5000				
Salarios Ingenieros:	90000	90000	90000	90000	90000
Total:	195426	90000	90000	90000	90000

	Precio($)
Adruinos:	
Arduino Board	18
Ethernet Shield	13
RF module	40
	71
nº arduinos	6
Coste total	426

Ilustración 8-1:Tabla presupuesto proyecto

Cálculo Valor actual de la inversión total:

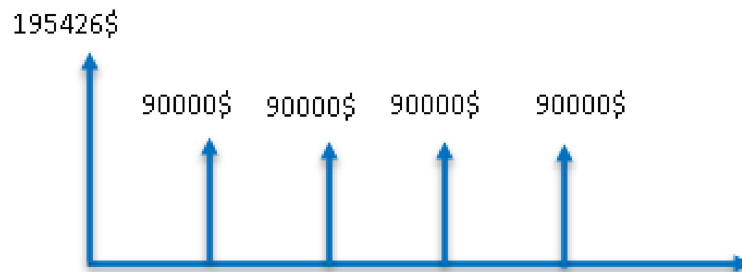


Ilustración 8-2:Flujo de caja anuales

$$\boldsymbol{NPV} = 195426 + 90000/(1+0.03) + 90000/(1+0.03)^2 + 90000/(1+0.03)^3 + 90000/(1+0.03)^4 = \mathbf{529964.85\$}$$

Ecuación 8.1:Cálculo del valor actual del coste del proyecto

Por lo tanto, el valor actual de un primer proyecto de investigación de este tipo está en torno al medio millón de dólares.

Aunque como ya se ha mencionado debido a la incertidumbre acerca de cuándo será posible la implantación de estos sistemas o cuando se convertirán en una realidad hace que se trate de una inversión arriesgada ya que cuanto

más lejos esté este futuro menor es el retorno de la inversión, sin embargo, se trata de un proyecto con un gran futuro a largo plazo.

8.2 ANÁLISIS DE LA FIABILIDAD

Como ya se ha mencionado en varias ocasiones, el beneficio principal que ofrecen los sistemas distribuidos de control frente a los centralizados es que se trata de sistemas resilientes en tanto que al tratarse de varios dispositivos de control no están sujetos al fallo de un único elemento del sistema, es decir, aunque falle uno de los dispositivos de control el sistema podrá seguir funcionando mientras el elemento dañado es reparado, sin grandes consecuencias. Por el contrario, el sistema centralizado, cuando el dispositivo central falla, el sistema se viene abajo, con las pérdidas económicas que eso supone. Es por ello por lo que el beneficio de esta implementación no se basa en un beneficio directo o en una reducción del coste inicial, sino más bien en la reducción de pérdidas futuras o la reducción de coste a largo plazo.

A continuación, se plantea un ejemplo del análisis de fiabilidad de un sistema sencillo que permitirá entender lo mencionado anteriormente de forma sencilla.

- El tiempo de vida de un sistema se expresa como una variable aleatoria X.
  Y se define la fiabilidad del sistema como una función $R(t)$(probabilidad de que el sistema funcione correctamente).
  $$R(t) = P(X > t)$$
  A partir del estudio de los fallos de los componentes del sistema se obtiene la fiabilidad.

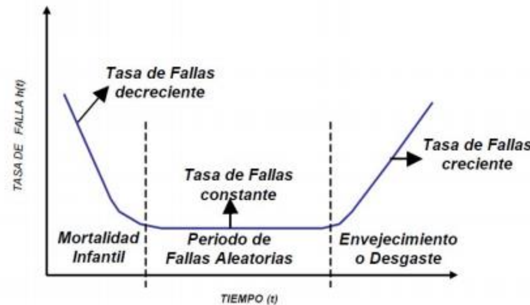


Ilustración 8-3:Curva de la bañera

Si la tasa de errores es constante, la fiabilidad sigue una exponencial.

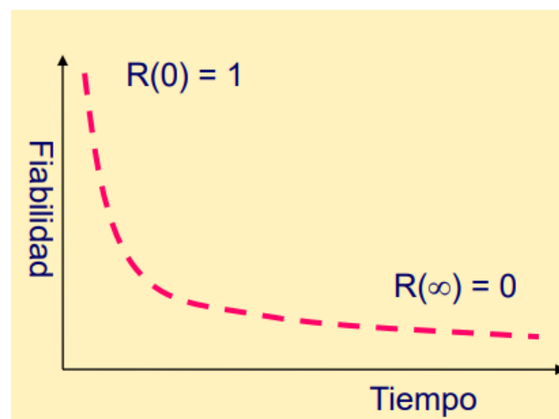


Ilustración 8-4:Curva de fiabilidad(https://www.arcos.inf.uc3m.es/infodsd/wp-content/uploads/sites/38/2017/02/3.pdf)

En este caso podemos asimilar el sistema distribuido a un sistema en paralelo, es decir, el sistema falla cuando lo hacen todos los componentes. Para este tipo de sistemas obtenemos:

$$\boldsymbol{R(t) = 1 - \prod_{k=1}^{N} Q_k(t)\, , donde \quad Q_k(t) = 1 - R(t)}$$

Ecuación 8.2:Fiabilidad sistema en paralelo

- Se define la disponibilidad de un sistema A(t) como la probabilidad de que en sistema está funcionando correctamente en el instante t.
  TMF : tiempo medio hasta el fallo .
  TMR: tiempo medio de reparación.

$$disponibilidad = TMF/(TMF + TMR)$$

Si consideramos que la fiabilidad del componente central es la misma que la de cada uno de los componentes individuales del sistema descentralizado, y también lo son el tiempo medio hasta el fallo y de reparación obtenemos:

$$\boldsymbol{Rc(t) = R(t)}$$

Ecuación 8.3: Fiabilidad sistema centralizado

$$\boldsymbol{Rd(t) = 1 - \left(1 - R(t)\right)^{N}}$$

Ecuación 8.4:Fiabilidad sistema distribuido

Se puede ver como la fiabilidad del distribuido es superior al central. Por ejemplo con R(t)= 0.9 y un sistema distribuido de 3 componentes tendríamos:

$$Rc(t) = 0.9$$

$$Rd(t) = 1 - \left(1 - R(t)\right)^{N} = 0.999$$

Además, se puede comprobar que cuanto mayor sea el número de controladores del sistema distribuido mayor será su fiabilidad, frente al centralizado.

De la misma forma la disponibilidad del sistema central es también significativamente superior al centralizado. Como cada vez que un dispositivo falla este es reparado, el sistema distribuido siempre funcionará, excepto en el improbable caso de que todos los controladores fallen en el intervalo de tiempo en el que el primero que se ha dañado es reparado.

La aproximación aquí presentada se trata de una aproximación muy simple ya que el sistema distribuido no puede asemejarse del todo a un sistema en paralelo, ya que se requiere que exista al menos un porcentaje mínimo de controladores operativos, además de que estos controladores pueden tener fallos adicionales asociados al sistema de comunicación y sincronización que por el contrario el sistema central no tiene.

Sin embargo, este análisis sirve para dar una idea de la gran ventaja que estos sistemas distribuidos pueden aportar, en términos de fiabilidad.

En contra, es cierto que el coste de mantenimiento será mayor en el caso distribuido, pero es compensado sobradamente ya que se compensa las enormes pérdidas que supone un **apagón** del sistema provocado por el fallo del sistema central, y que en el caso del distribuido es prácticamente inexistente. Otra ventaja económica del sistema distribuido es la reducción del coste de **cableado**, y a pesar de que el sistema distribuido necesite de un mayor número de **controladores**, estos serán considerablemente más baratos que uno central que necita de mayor capacidad para manejar una gran cantidad de información.

Además se puede ver como se puede ver en el gráfico inferior el controlador de una microrredes está en torno al 15%. Si además tenemos encuenta la reducción del coste de cableado y dado que la mayor parte coste es de generación, la cual permanece igual en ambos sistemas de control, podemos decir que seguramente la diferencia en el coste de un control distribuido frente a uno cental no será demasiado significativa.

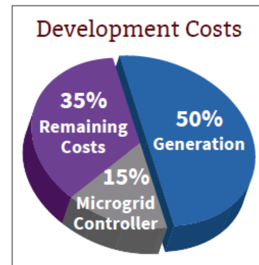


Ilustración 8-5:Coste de desarrollo de una microrred

(https://microgridknowledge.com/optimized-microgrid/)

Lamentablemente es muy complicado hacer un análisis exacto que nos permita cuantificar esto ya que las posibles diferencias en el precio entre ambos sistemas se deberán en gran medida de la dimensión y complejidad de la microrred y por otro lado se trata de una tecnología en desarrollo y no se sabe con certeza los futuros precios de dichos controladores.

En cualquier caso, los sistemas distribuidos parecen tener más ventajas que desventajas económicas frente a los sistemas centralizados y es por ello por lo que merece la pena seguir invirtiendo en la investigación y desarrollo de estos, debido al gran potencial que parecen tener y los futuros beneficios económicos, sociales y medioambientales que pueden aportar.

Capítulo 9. CONCLUSIONES Y TRABAJOS FUTUROS

Tras la finalización de este trabajo, la conclusión que se puede extraer es que a pesar de que todavía queda un largo camino por recorrer para que la aplicación sistemas distribuidos para el control de microrredes puedan realmente ser instaurados en el sistema real, parecen ser una alternativa viable y eficiente que permita la transición hacia un sistema más flexible y resiliente donde los sistemas de generación distribuidos y las renovables jueguen un papel importante. Y aunque es cierto que un control centralizado tiene ventajas en cuanto a la **precisión, velocidad de computación,** y **respuestas más suaves ante las variaciones**, la mejora en la resiliencia del sistema que ofrecen los sistemas distribuidos puede compensar dichos inconvenientes.

Los siguientes pasos para continuar con el proyecto serían los siguientes:

1. En primer lugar y los más importante desarrollar un código que tenga en cuenta el problema de los "packet drops", es decir que sea resiliente frente a estos para así obtener siempre el resultado correcto para la generación de potencia activa de cada generador.

2. El segundo paso sería la implementación del *Max Consensus* en el algoritmo de forma que determine el número necesario de iteraciones. Como se mencionó en el apartado donde se explicaba el algoritmo de *Max-Min Consensus,* el algoritmo se ejecutarías hasta asegurar que ningún nodo del sistema tuviese un valor superior al predeterminado.

3. Una vez hecho lo anterior se debe volver a testear y hacer diferentes ensayos que permitan determinar la eficacia y eficiencia del algoritmo, así como hacer una comparativa con un sistema centralizado.

En la investigación que se está llevando a cabo en la Universidad de Illinois, se implementarán todos los algoritmos distribuidos necesarios para el control de una microrred en al siguiente modelo de microrred más complejo. Para extraer
resultados más concluyentes acerca del futuro de estos sistemas de control para microrredes.

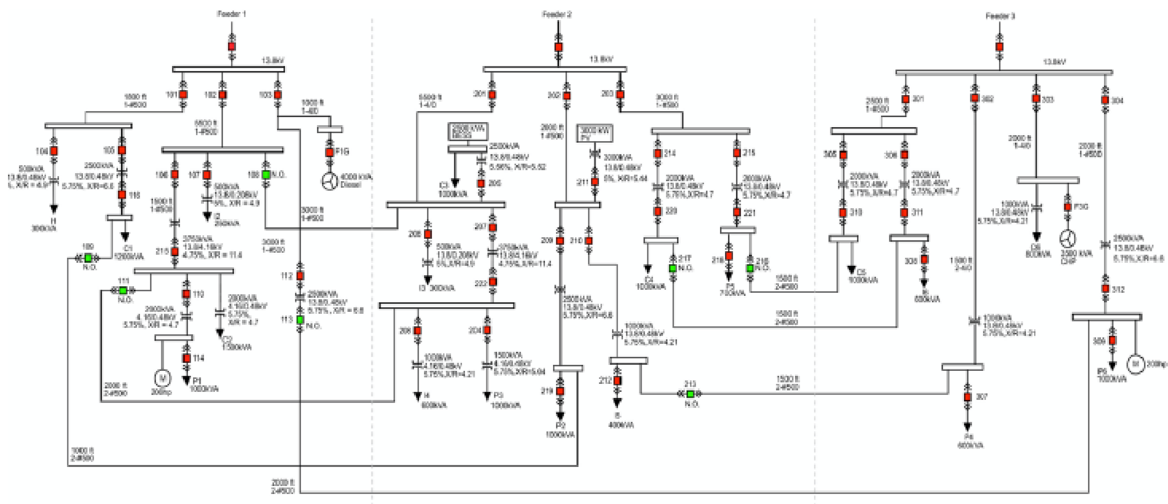


Ilustración 9-1:Modelo de la red Banshee

Capítulo 10. BIBLIOGRAFÍA

[19] S. T. Cady, M. Zholbaryssov, A. D. Domínguez-García and C. N. Hadjicostis, "A Distributed Frequency Regulation Architecture for Islanded Inertialess AC Microgrids," in *IEEE Transactions on Control Systems Technology*, vol. 25, no. 6, pp. 1961-1977, Nov. 2017.

[20] S. T. Cady, A. D. Domínguez-García and C. N. Hadjicostis, "A Distributed Generation Control Architecture for Islanded AC Microgrids," in *IEEE Transactions on Control Systems Technology*, vol. 23, no. 5, pp. 1717-1735, Sept. 2015, doi: 10.1109/TCST.2014.2381601.

[21] Félix García y Alejandro Claderón Mateos."Diseños de Sitemas Distribuidos : Tolerancia a fallos" 2016-2017. https://www.arcos.inf.uc3m.es/infodsd/wp-content/uploads/sites/38/2017/02/3.pdf

[22] Nuñez Mata, Oscar, Diego Ortiz Villalba, Rodrigo Palma-Behnke, " *MICRORREDES EN LA RED ELÉCTRICA DEL FUTURO - CASO HUATACONDO*." Centro De Energía, Facultad De Ciencias Físicas y Matemáticas, Escuela De Ingeniería, Universidad De Chile.Santiago, Chile. Escuela De Ingeniería Eléctrica, Faculta De Ingeniería, Universidad De Costa Rica. San José, Costa Rica. Departamento Eléctrica - Electrónica, Escuela Politecnica Del Ejército. Latacunga, Ecuador., 28 Sept. 2013, file:///C:/Users/Patri/Downloads/15214-Texto%20del%20art%C3%ADculo-27849-1-10-20140707%20(1).pdf.

[23] O. Azofeifa et al., "Controller Hardware-in-the-Loop Testbed for Distributed Coordination and Control Architectures," 2019 North American Power Symposium (NAPS), Wichita, KS, USA, 2019, pp. 1-6, doi: 10.1109/NAPS46351.2019.8999980

[24] "Objetivos de desarrollo sostenible" ,Naciones unidas,15 Sep 2015, https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

[25] "¿Que es el hardware-in-the-loop?" ,National Instruments ,14 may 2020 https://www.ni.com/es-es/innovations/white-papers/17/what-is-hardware-in-the-loop-.html

[26] R. H. Lasseter, "MicroGrids: A Conceptual Solution," 2002 IEEE Power Eng. Soc. Winter Meet. Conf. Proc. (Cat.No.02CH37309), vol. 1, pp. 305–308, 2002.

[27] “Virtual HIL Device - Typhoon HIL.” Abril-202, https://www.typhoon-hil.com/products/virtual-hil-device/.

ANEXO I

1. Funciones relacionadas con el algoritmo *Feasible Flow.*

- **LeaderfeasibleFlow_RSL**

```
float OAgent_OPF::leaderfeasibleFlow_RSL(bool busgen, uint8_t iterations,
uint16_t period, float load ) {
    unsigned long t0 = myMillis();
    unsigned long startTime = t0 + RC_DELAY;
    OLocalVertex * s = _G->getLocalVertex();
    float gamma = 0;
    bool scheduled = _waitForChildSchedulePacketRC(startTime, iterations,
period);

    if (!scheduled)
    {
        Serial << "FF scheduling was a FAIL"<<endl;
        gamma = -1;
    }
    else
    {
        Serial << "FF scheduling was a SUCCESS"<<endl;
        if(_waitToFinishSchedule(startTime,true,10000))
        {
            Serial << "Correct Startime is " <<startTime<< ". My startime is "<<
myMillis() <<endl;
            gamma = feasibleFlowAlgorithm(busgen,iterations,period,load);
        }
    }
    return gamma;
}
```

- **NonleaderfeasibleFlow_RSL**

```
float OAgent_OPF::nonleaderfeasibleFlow_RSL(bool busgen, uint8_t iterations,
uint16_t period, float load ) {
    unsigned long startTime = 0;
    //delay(50);
    float gamma = 0;
    bool scheduled = _waitForParentSchedulePacketRC(startTime, iterations,
period);
    //Serial<<"Schedule done at "<<myMillis()<<"\n";

    //bool stat = startTime>myMillis();
    //Serial<<"Startime > Time? "<<stat<<"\n";

    //Serial<<"NonLeader: Startime= "<<startTime<<", Time= "<<myMillis()<<"\n";
```

```
    if(scheduled)
    {
        Serial << "FF scheduling was a SUCCESS"<<endl;
        if(_waitToFinishSchedule(startTime,true,10000))
        {
            Serial << "Correct Startime is " <<startTime<< ". My startime is "<<
myMillis() <<endl;
            gamma=feasibleFlowAlgorithm(busgen,iterations,period,load);
        }

        //digitalWrite(48,LOW);
    }
    else
    {
        Serial << "FF scheduling was a FAIL"<<endl;
        gamma = -1;
    }
```

- **FeasibleFlowAlgorithm_RSL**

```
float OAgent_OPF::feasibleFlowAlgorithm_RSL(bool busgen, uint8_t iterations,
uint16_t period, float load) {
    srand(analogRead(7));                          //moved this instruction here from
fairSplitRatioConsensus() - Sammy
    OLocalVertex * s = _G->getLocalVertex();
    // int leader_id = s->getleaderID();
    uint8_t nodeID = s->getID();
    // if (leader_id==nodeID)
    //  setLeader(0);
    float gamma = 0;

    if(isLeader())
    {
         gamma=leaderfeasibleFlow_RSL(busgen,iterations,period,load);
    }
    else
    {
        gamma=nonleaderfeasibleFlow_RSL(busgen,iterations,period,load);
    }
        //Serial<<"Sup bro?! "<<getbufferdata(0)<<"\n";

       return gamma;
}
```

- **FeasibleFlow**

```
float  OAgent_OPF::feasibleFlowAlgorithm(bool busgen, uint8_t iterations,
uint16_t period, float load ) {
    OLocalVertex * s = _G->getLocalVertex();
    ORemoteVertex * n = _G->getRemoteVertex(1);
    LinkedList * l = _G->getLinkedList();
    l->resetLinkedListStatus(s->getStatusP());
    l->updateLinkedList(s->getStatusP());
    uint8_t * neighborStatusP = s->getStatusP();

    ORemoteVertex * neighborP;

    uint16_t nodeID = s->getID();
    ORemoteVertex * nodeP = (n+(nodeID-1));

    uint8_t neighborID;
   int node_check[NUM_REMOTE_VERTICES];
   float neighbor_fp;
   float bp;
   float bpinicial;
   float bj;
   float wi = float(s->getOutDegree());
   float gi;
   float wj;
   bool receivedPacket;
   bool txDone;
   int timeout = 100;

    uint16_t packetReceiveCount = 0;
    uint16_t packetsLost = 0;
    uint16_t packetReceived = 0;


   l->setInitialActiveFlows(nodeID, n);

   gi = busgen*0.5*(s->getMax()+ s->getMin());// calculamos generacion inicial
   s->setGi(gi);
   s->setActiveDemand(load);
   float Pd = load;

   bpinicial = gi - Pd - l->addActiveInitialFlows(nodeID,n);
   s->setActiveBalance(bpinicial);
    uint8_t frame = 25;
    unsigned long start = (millis()-period);

    srand(millis());
    uint16_t txTime = (rand() % (period - 2*frame)) + frame;

    for(uint8_t k = 0; k < iterations; k++)
    {

        Serial<<"NUEVA ITERACION : "<<k+1<<endl;
```

```
        start = millis();   // initialize timer
        txDone = false;
        packetReceiveCount = 0;
        //compute bi
        if(busgen == 0)
        {
             s->setGi(0);
        }
           gi = s->getGi() - 0.5*s->getActiveBalance()/wi;

            if(gi > s->getMax()){

                gi = s->getMax();

            }else if(gi < s->getMin())
            {
                gi = s->getMin();
            }
                s->setGi(gi);

        while( uint16_t(millis()-start) < period )
        {
            receivedPacket =
_waitForNeighborPacket(neighborID,FAIR_SPLITTING_HEADER,true,100);
            if ( receivedPacket && (*(neighborStatusP+neighborID-1) == 2) )
            {
                packetReceiveCount++;
                 delay(5);
                *(neighborStatusP+neighborID-1) = 3;
                    bj = _getBjFromPacket();
                    neighborP = (n+(neighborID-1));

                        if(k == 0)
                        {
                        bp =  bpinicial;
                        s->setActiveBalance(bp);
                        }

                        if (nodeID<neighborID)
                        {
                            neighbor_fp = (neighborP->getActiveFlow() - 0.5*bj +
0.5*(s->getActiveBalance())/wi);
                            if (neighbor_fp > (neighborP)->getActiveFlowMax())
                            {
                                neighbor_fp = (neighborP)->getActiveFlowMax();

                            }else if(neighbor_fp<(neighborP)->getActiveFlowMin())
                            {
                                neighbor_fp = (neighborP)->getActiveFlowMin();
                            }
                        }else
                        {
```

```
neighbor_fp = ((-1)*(neighborP->getActiveFlow()) - 0.5*bj + 0.5*(s-
>getActiveBalance())/wi);
                            neighbor_fp = -neighbor_fp;
                            ;
                                if (-neighbor_fp>(neighborP)->getActiveFlowMax())
                                {
                                 neighbor_fp=-(neighborP)->getActiveFlowMax();

                           }else if(-neighbor_fp<(neighborP)->getActiveFlowMin())
                                {
                              neighbor_fp = -(neighborP)->getActiveFlowMin();
                                }
                        }
                         (neighborP)->setActiveFlow(neighbor_fp);
            }
            if(!txDone && (uint16_t(millis()-start) >= txTime))
            {
                txDone = true;
               _broadcastBalanceFeasibleFlow(s->getActiveBalance(),wi);
            }
        }

        if(txDone)
        {
            float bp = s->getGi() - Pd - l->addActiveInitialFlows(nodeID,n);

                s->setActiveBalance(bp);
               Serial<<"recalculamos balance: "<<_FLOAT(bp,6)<<endl;
        }

        packetReceived += packetReceiveCount;
        packetsLost += (_G->getN() - packetReceiveCount - 1);

        packetReceiveCount = 0;

        l->resetLinkedListStatus(s->getStatusP());

    }
    Serial<<packetsLost<<" packets lost"<<endl;
    delay(5);
    Serial<<packetReceived<<" packets received"<<endl;
    delay(5);
    Serial<<"El balance final es :"<<s->getActiveBalance()<<endl;
    return (s->getGi());
}
```

- **broadcastBalanceFeasibleFlow**

```
void OAgent_OPF::_broadcastBalanceFeasibleFlow(float bi, float wi) {
```

```
    uint8_t payload[7];
    bi =  bi*BASE;
    uint8_t sigBi ;

    uint32_t Bi;
    //long Wi = long(wi);
    if(bi<0){
        sigBi = 0;
        bi = -1* bi;
        Bi = (uint32_t) (bi/wi);
   }else{
        sigBi = 1;
        Bi = (uint32_t) (bi/wi);

   }
    payload[0] = FAIR_SPLITTING_HEADER;
    payload[1] = FAIR_SPLITTING_HEADER>>8;
    payload[2] = sigBi;
    payload[3] = Bi; // guardamos el valor de mu los primeros?? 16 bits
    payload[4] = Bi>> 8;// guardamos el valor de mu los ultimos?? 16 bits
    payload[5] = Bi>> 16;
    payload[6] = Bi>> 24 ;

    _zbTx = ZBTxRequest(_broadcastAddress, ((uint8_t * )(&payload)),
sizeof(payload)); // create zigbee transmit class
    //se ha transformado nuestro vector de 16bits en uno se 8bits --> se ha
doblado la longitud del vector
    unsigned long txTime = _xbee->sendTwo(_zbTx,false,true); // transmit with
time stamp
#ifdef VERBOSE
    Serial << _MEM(PSTR("Transmit time: ")) << txTime << endl;
#endif
}
```

2. Funciones relacionadas con el algoritmo *Share Flag.*

- **SahreFeasibleFlowFlag**

```
float OAgent_OPF::ShareFeasibleFlowFlag(uint8_t iterations, uint16_t period, bool
feasibleFlowFlag ) {
    srand(analogRead(7));                    //moved this instruction here from
fairSplitRatioConsensus() - Sammy
    OLocalVertex * s = _G->getLocalVertex();
    uint8_t nodeID = s->getID();

    float gamma = 0;

    if(isLeader())
    {
        gamma = leaderShareFeasibleFlowFlag(iterations,period,feasibleFlowFlag );
    }
    else
    {
        gamma =
nonleaderShareFeasibleFlowFlag(iterations,period,feasibleFlowFlag);
    }
        //Serial<<"Sup bro?! "<<getbufferdata(0)<<"\n";

       return gamma;
}
```

- **Leader/nonleadesFeasibleFlowFlag**

```
float OAgent_OPF::leaderShareFeasibleFlowFlag(uint8_t iterations, uint16_t
period, bool feasibleFlowFlag) {
    unsigned long t0 = myMillis();
    unsigned long startTime = t0 + RC_DELAY;
    OLocalVertex * s = _G->getLocalVertex();
    float gamma = 0;
    bool scheduled = _waitForChildSchedulePacketRC(startTime, iterations,
period);

    if (!scheduled)
    {
        Serial << "Flag scheduling was a FAIL"<<endl;
        gamma = -1;
    }
    else
    {
        Serial << "Flag scheduling was a SUCCESS"<<endl;
        if(_waitToFinishSchedule(startTime,true,10000))
        {
```

```
            Serial << "Correct Startime is " <<startTime<< ". My startime is "<<
myMillis() <<endl;
            gamma = ShareFlag(iterations,period,feasibleFlowFlag);
        }
    }
    return gamma;
}
```

```
float OAgent_OPF::nonleaderShareFeasibleFlowFlag(uint8_t iterations, uint16_t
period, bool feasibleFlowFlag) {
    unsigned long startTime = 0;
    //delay(50);
    float gamma = 0;
    bool scheduled = _waitForParentSchedulePacketRC(startTime, iterations,
period);

    if(scheduled)
    {
        Serial << "Flag scheduling was a SUCCESS"<<endl;
        if(_waitToFinishSchedule(startTime,true,10000))
        {
            Serial << "Correct Startime is " <<startTime<< ". My startime is "<<
myMillis() <<endl;
            gamma = ShareFlag(iterations,period,feasibleFlowFlag);
        }
    }
    else
    {
        Serial << "Flag scheduling was a FAIL"<<endl;
        gamma = -1;
    }
    return gamma;
}
```

- **<u>ShareFlag</u>**

```
bool OAgent_OPF::ShareFlag(uint8_t iterations, uint16_t period, bool
feasibleFlowFlag )
{
    OLocalVertex * s = _G->getLocalVertex();
    float Dout = float(s->getOutDegree() + 1);
    ORemoteVertex * n = _G->getRemoteVertex(1);
    LinkedList * l = _G->getLinkedList();

     uint8_t neighborID;
    bool receivedPacket;
    bool Flag;
    unsigned long start;
    bool txDone;
    uint16_t txTime;
    int iter = 0;
    int node_check[NUM_REMOTE_VERTICES];
```

```
    uint32_t aLsb;

    if(feasibleFlowFlag){

        s->setFeasibleFlowFlag(true);
        Serial<<"node flag:"<<s->getFeasibleFlowFlag()<<endl;

    }else{

         s->setFeasibleFlowFlag(false);
        Serial<<"node flag:"<<s->getFeasibleFlowFlag()<<endl;
    }


    for(int i=0; i < NUM_REMOTE_VERTICES; i++)
    {
        node_check[i] = 0;
    }
    int frame = 30;

    do
    {
        srand(analogRead(0));
        txTime =  (rand() % (period - 2*frame)) + frame;
        txDone = false;     // initialize toggle to keep track of broadcasts
        start = millis();   // initialize time
        uint8_t i;
        while(uint16_t(millis()-start) < period)
        { receivedPacket =
_waitForNeighborPacket(neighborID,FAIR_SPLITTING_HEADER,true,100);
            Serial<<"menasje de nodo:"<<neighborID<<endl;
            if(_getFeasivbleFlowFlagfromPackage(s))
                    {
                        Serial<<"recieved positive flag"<<endl;
                        s->setFeasibleFlowFlag(true);
                    }

            if((int((millis() - start)) >= txTime) && !txDone) {
                txDone = true;
                Serial<<"broadcast flag"<<endl;
                _broadcastFeasibleFlowFlag(s);
            }
        }
        if(!_quiet) {

            delay(10);
        } else {
            delay(25);
        }

        iter++;// increase the iteration count

    }while(iter < iterations); //we need to implement here the max consensus
```

```
    if(s->getFeasibleFlowFlag()){
         Flag = true;
    }else{
        Flag = false;
    }

    return Flag;
}
```

- **broadcastFeasibleFlowFlag**

```
void OAgent_OPF::_broadcastFeasibleFlowFlag(OLocalVertex * s) {
    uint8_t payload[3];

    uint8_t Flag;

    if(s->getFeasibleFlowFlag()){

        Flag = 1;

    }else{

        Flag = 0;
    }

    payload[0] = FAIR_SPLITTING_HEADER;
    payload[1] = FAIR_SPLITTING_HEADER>>8;
    payload[2] = Flag;


    _zbTx = ZBTxRequest(_broadcastAddress, ((uint8_t * )(&payload)),
sizeof(payload)); // create zigbee transmit class
    unsigned long txTime = _xbee->sendTwo(_zbTx,false,true); // transmit with
time stamp
#ifdef VERBOSE
    Serial << _MEM(PSTR("Transmit time: ")) << txTime << endl;
#endif
}
```

- **_getFeasibleFlowFlagFromPackage**

```
bool OAgent_OPF::_getFeasivbleFlowFlagfromPackage(OLocalVertex * s)
{
    uint8_t ptr = 2;
    long Flag = _getUint8_tFromPacket(ptr);
    Serial<<"flag recived : "<<Flag<<endl;
    if(s->getFeasibleFlowFlag())
    {
        Serial<<"flag aready true"<<endl;
```

```
        return true;
    }else
    {
        if(Flag == 0)
        {
            Serial<<"recieved false flag form package"<<endl;
            return false;  //there needs to be 2nd stage
        }else
        {
            Serial<<"recieved true flag form package"<<endl;
            return true;  //there is no need for 2nd stage
        }

    }
}
```

1. Código Arduino (Control Secundario de frequencia).

```
#include <Streaming.h>
#include <XBee.h>
//#include <Dyno.h>
#include <OGraph_OPF.h>
#include <OAgent_OPF.h>
#include <MgsModbus.h>
#include <SPI.h>
#include <Ethernet.h>
//Node 9

long base = 10000;  // use base to increase precision of results
uint8_t i=1;//number of inneighbors

XBee xbee = XBee();
ZBRxResponse rx = ZBRxResponse();
OLocalVertex s = OLocalVertex(0x415786E1,9,-0.707,0.707,0,0,i, base);
LinkedList l = LinkedList();  //#NODE
OGraph_OPF g = OGraph_OPF(&s,&l);
OAgent_LinkedList al = OAgent_LinkedList();  //#NODE
OAgent_OPF a = OAgent_OPF(&xbee,&rx,&g,&al,true,true); // argument rx?


uint8_t sPin = 7;      // synced led
uint8_t cPin = 48;     // coordination enabled led pin

//variables for node sync check
boolean de = false;

//AFE and controller variables
float f_error0;         // variable to store the read value
float v_error0;         // variable to store the read value
float f_error1;         // ratio consensus result for average frequency error
```

```
float load0;
float feasibleflowFlag0;
bool Flag =0;

float error = 0;
float u_f =0;
float u_v =0;
float u_set = 0;
//Modbus Communication
MgsModbus Mb;
int val;

byte mac[] = {0x90, 0xA2, 0xDA, 0x0E, 0x94, 0xB9 };
IPAddress ip(192, 168, 2, 9); // What are these addresses
IPAddress gateway(192, 168, 2, 20);
IPAddress subnet(255, 255,255, 0);

uint16_t state_high;
uint16_t state_low;
uint8_t Ref_high;
uint8_t Ref_low;
uint8_t Count_high;
uint8_t Count_low;
uint8_t Pos_high;
uint8_t Pos_low;
int fc;
int ref;
int count;
int pos;

float eps_f = 0.001;
float eps_v = 0.001;
float D = 1;

void setup()  {
  Serial.begin(38400);
  Serial3.begin(38400);
  pinMode(cPin, OUTPUT);
  pinMode(sPin, OUTPUT);
  digitalWrite(cPin,HIGH);
  digitalWrite(sPin,HIGH);

  xbee.setSerial(Serial3); //Specify the serial port for xbee

    g.addInNeighbor(0x415786A9,12,0,0,0.26,0.035); // node 12

  g.configureLinkedList();

int NUMNEIGH = s.getOutDegree();//como se hace esto

  digitalWrite(cPin,LOW);
  digitalWrite(sPin,LOW);
```

```
// initialize the ethernet device
 Ethernet.begin(mac, ip, gateway, subnet);   // start etehrnet interface
 for (int i=0;i<12;i++) {
    Mb.MbData[i] = 0;
 }

   if(!(a.isLeader()))
   {
     Serial.println("Still trying to sync");
     if(a.sync())
     {
       Serial.println("Communication Link established");
       Serial.println("c");
       digitalWrite(sPin,HIGH);
     }

   }
   if (a.isLeader())
   {
     Serial.println("Send letter s(r) to sync(resync)");
     while (Serial.available() == 0)
     {
       //simply makes the arduino wait until commputer sends signal
     }
     if(Serial.available())
     {
       Serial.println("got some letter");
       uint8_t b = Serial.read(); //enter the character 's'
       Serial.println(b);
       if (b == 'r')
         {
           a.setLeader(0);
         }
       if ((b == 's')||(b == 'r'))
       {
          Serial.println("got the s and about to sync");
          while(a.sync() == 0)
          {

          }
          Serial.println("Communication Link established");
          Serial.println("c");
          digitalWrite(sPin,HIGH);
          //ce = true;
       }
     }
   }
      receiveTyphoonData();
   bool feasibleflowFlag = Mb.MbData[4]*((-2*Mb.MbData[5])+1);
   int load = Mb.MbData[8]*((-2*Mb.MbData[9])+1);
   int f_error =  Mb.MbData[0]*((-2*Mb.MbData[1])+1);
     f_error0 =  float(f_error);
     load0 = float(load);
```

```
      load0 = load0/(200*base);
      f_error0 =  f_error0/base;

      Serial.print("f error: ");
      Serial.println(float(f_error0),4);

      Serial.print("feasibleflowFlag: ");
      Serial.println(feasibleflowFlag);

      Serial.print("load value: ");
      Serial.println(float(load0),4);

      if (a.isLeader())
      {
          Serial.println("Begin feasible flow? (y/n)");
          while (Serial.read() != 'y')
          {
            //simply makes the arduino wait until commputer sends signal
          }
         Serial.println("got the y");
         u_f = a.feasibleFlowAlgorithm_RSL(1,30,600,0
         Serial.println("Gi result");
         Serial.println(u_f,4);
      }
      if (!(a.isLeader()))
      {
         u_f = a.feasibleFlowAlgorithm_RSL(1,30,600,0
         Serial.println("Gi result");
            Serial.println(u_f,4);
      }

        if (u_f<0)
      {
        Mb.MbData[0]=1;
      }
      else
      {
        Mb.MbData[0]=0;
      }

       Mb.MbData[1]=base*abs(u_f);
       sendConsensusResults();
      a.resync();
}

void loop() {
      receiveTyphoonData();//recibimos información de typhon

    bool feasibleflowFlag = Mb.MbData[4]*((-2*Mb.MbData[5])+1);
    int load = Mb.MbData[8]*((-2*Mb.MbData[9])+1);
    int f_error =  Mb.MbData[0]*((-2*Mb.MbData[1])+1);

      f_error0 =  float(f_error);
```

```
      load0 = float(load);

      load0 = load0/base;
      f_error0 =  f_error0/base;
      load0 = 0;


      Serial.print("f error: ");
      Serial.println(float(f_error0),4);

       Serial.print("feasibleflowFlag: ");
      Serial.println(feasibleflowFlag0);

       Serial.print("load value: ");
      Serial.println(float(load0),4);

      Serial.print("D: ");
      Serial.println(D,4);

      delay(100);

   if (a.isLeader())
      {
        Serial.println("Begin feasible flow? (y/n)");
        while (Serial.read() != 'y')
        {
          //simply makes the arduino wait until commputer sends signal
        }
           Flag = a.ShareFeasibleFlowFlag(10,200, feasibleflowFlag);
           Serial.println("Flag result");
           Serial.println(bool(Flag));
      }
      if (!(a.isLeader()))
      {
          Flag = a.ShareFeasibleFlowFlag(10,200, feasibleflowFlag);
           Serial.println("Flag result");
          Serial.println(bool(Flag));
      }

 if(Flag)
      {
           if (a.isLeader())
          {
            Serial.println("Begin feasible flow? (y/n)");
            while (Serial.read() != 'y')
            {
           //simply makes the arduino wait until commputer sends signal
            }
               u_set = a.feasibleFlowAlgorithm_RSL(1,30,300,0);
               u_f = u_set;
              Serial.print("G9: ");
              Serial.println(float(u_f),4);
```

```
          }
          if (!(a.isLeader()))
          {
          u_set = a.feasibleFlowAlgorithm_RSL(1,30,300,0);
            u_f = u_set;
            Serial.print("G9: ");
            Serial.println(float(u_f),4);
          }
      }else
      {
           if (a.isLeader())
          {
            Serial.println("Begin feasible flow? (y/n)");
            while (Serial.read() != 'y')
            {
          //simply makes the arduino wait until commputer sends signal
            }
              f_error1 = a.ratioConsensusAlgorithm(f_error0,D,30,300);

          }
          if (!(a.isLeader()))
          {

             f_error1 = a.ratioConsensusAlgorithm(f_error0,D,30,300);
          }
          Serial.println("ratio consensus result");
          Serial.println(f_error1,4);
          delay(100);
          if(abs(f_error1) > eps_f & load0 == 0)
          {
            error=error + -1*0.707*f_error1;
             u_f=u_set+0.7071*error;
             //Serial.println(u,4);
          }
      }

      //Sending data

      if (u_f<0) // se analiza el signo de este valor para poder enviarlo
      {
        Mb.MbData[0]=1;
      }
      else
      {
        Mb.MbData[0]=0;
      }

       Mb.MbData[1]=base*abs(u_f);
      sendConsensusResults(); // se envian todos los resultados de MBData
      a.resync();
}

void sendConsensusResults()
```

```
{

  fc = 16;
  ref = 0;
  Ref_high = uint8_t(ref >> 8 && 0x00FF);
  Ref_low = uint8_t(ref & 0x0FF);
  count = 2;
  Count_high = uint8_t(count >> 8 && 0x00FF);
  Count_low = uint8_t(count & 0x0FF);
  pos = 0;
  Pos_high = uint8_t(pos >> 8 && 0x00FF);
  Pos_low = uint8_t(pos & 0x0FF);

  int node9_ip = 69;
  Mb.Req(MB_FC_WRITE_MULTIPLE_REGISTERS,0,4,0,node9_ip);
  Mb.MbmRun();

}

void receiveTyphoonData()
{
  int node9_ip = 69; //part of ip address for node 9 on the HIL side
  Mb.Req(MB_FC_READ_INPUT_REGISTER,0,10,0,node9_ip);
  Mb.MbmRun();
}
```

1. Función *Feasible Flow* sistema centralizado de 3 nodos .

```
2.  #include <iostream>
3.  #include <math.h>
4.  using namespace std;
5.
6.  void Powerflow(float bi[],float g[], int locvectorgen[],int load[],float
    f[][3],int numvecgen[]);
7.  int main(void)
8.  {
9.      #define Np 3 // totalnumber of nodes in the system
10.     #define Npg 2// number of generators
11.     #define Npl 1 // number of loads
12.
13. //definimos variables
14.  int i;// numero del nudo
15.  int j;
16.  int k;
17.  int wi[Np]={2,2,3};
18.  int gimax[Np];
19.  int gimin[Np];
20.  float g[Np];
21.  float fijmax[Np][Np];//matrix of the maximum power trhough the branches
22.  float fijmin[Np][Np];
23.  float bi[Np];// flow balance vector
```

```
24.  float f[Np][Np];//estimate maintained by i for the flow to each j at
   iteration k;
25. int A[Np][Np]={{1,0,-1},{0,1,-1}};//incidence matrix
26.  int locvectorgen[Npg]={0,1};
27.  int numvecgen[Np]={1,1,0};
28.  int vectorcargas[Npl]={2};
29.  int load[Np]={0,0,3};// this is the vectors of the demand of each node
30.      gimax[0]=1;
31.      gimin[0]=-1;
32.      gimax[1]=3;
33.      gimin[1]=-3;
34.      gimax[2]=-3;
35.      gimin[2]=-3;
36.
37. // with this loop we fill the max and min power matrices that the lines
   can carry
38.  for(i=0;i<Np;i++){
39.      for(j=0;j<Np;j++){
40.
41.         fijmax[i][j]=0;
42.         fijmin[i][j]=0;
43.      }
44.  }
45.         fijmax[0][2]=2;
46.         fijmax[2][0]=-fijmax[0][2];
47.         fijmin[0][2]=1;
48.         fijmin[2][0]=-fijmin[0][2];
49.         fijmax[1][2]=2;
50.         fijmax[2][1]=-fijmax[1][2];
51.         fijmin[1][2]=0;
52.         fijmin[2][1]=-fijmin[1][2];
53.
54. // print the matrix
55.     for (i = 0; i <Np; i++) {
56.      /* Recorre filas */
57.      for (j = 0; j <Np; j++) {
58.      /* Recorre columnas */
59.       cout<< fijmax[i][j]<<"  ";
60.      }
61.      cout<<"\n";
62.     }
63.     cout<<"\n\n";
64.     for (i = 0; i <Np; i++) {
65.      /* Recorre filas */
66.      for (j = 0; j <Np; j++) {
67.      /* Recorre columnas */
68.       cout<< fijmin[i][j]<<"  ";
69.      }
70.      cout<<"\n";
71.     }
72.     cout<<"\n\n";
73. // Paso1://inicialization
74.  for(i=0;i<Np;i++){
```

```
75.         for(j=0;j<Np;j++){
76.         f[i][j]=0.5*(fijmax[i][j]+fijmin[i][j]);//this is the inicializted
    matrix of the flow trhough the branches
77.          cout<< f[i][j]<<"  ";
78.         }
79.         cout<<"\n";
80.  }
81.  cout<<"\n\n";
82.
83.   for(i=0;i<Np;i++){
84.      g[i]=0.5*(gimax[i]+gimin[i]);//this is the inicializted matrix of the
    flow trhough the branches
85.      cout<< g[i];
86.   }
87.   cout<<"\n\n";
88.//Paso 2: calculo del balance de potencia en cada nodo
89.
90.k=0;
91.while(k<=500){
92.
93.         Powerflow(bi,g,locvectorgen,load,f,numvecgen)
94.
95.         f[0][2]=f[0][2]-bi[2]/(2*wi[2])+bi[0]/(2*wi[0]);
96.         f[2][0]=f[2][0]-bi[0]/(2*wi[0])+bi[2]/(2*wi[2]);
97.         f[1][2]=f[1][2]-bi[2]/(2*wi[2])+bi[1]/(2*wi[1]);
98.         f[2][1]=-f[1][2];
99.
100.                 for(i=0;i<=Npg;i++)
101.                 {
102.                     g[locvectorgen[i]]=g[locvectorgen[i]]-
    bi[locvectorgen[i]]/(2*wi[locvectorgen[i]]);
103.                 }
104.               //Paso4: Debemos comprobar que no hemos sobrepasado los
    límites
105.
106.               for(i=0;i<=Np;i++){
107.
108.                   if(f[i][2]>fijmax[i][2]){
109.                       f[i][2]=fijmax[i][2];
110.                       f[2][i]=-f[i][2];
111.
112.                   }else if(f[i][2]<fijmin[i][2]){
113.                        f[i][2]=fijmin[i][2];
114.                        f[2][i]=-f[i][2];
115.                   }
116.                   if(g[i]>gimax[i]){
117.                       g[i]=gimax[i];
118.                   }else if(g[i]<gimin[i]){
119.                       g[i]=gimin[i];
120.                   }
121.               }
122.           k++;
123.         }
```

```
124.        //mostramos los resultado por pantalla
125.            for (i = 0; i <Np; i++) {
126.                 cout<< bi[i]<<"  ";
127.                }
128.                cout<<"\n\n";
129.
130.            for (i = 0; i <Np; i++) {
131.                 cout<< g[i]<<"  ";
132.                }
133.                cout<<"\n\n";
134.             for (i = 0; i <Np; i++) {
135.            /* Recorre filas */
136.              for (j = 0; j <Np; j++) {
137.            /* Recorre columnas */
138.               cout<< f[i][j]<<"  ";
139.              }
140.              cout<<"\n";
141.              }
142.               cout<<"\n\n";
143.
144.           return 0;
145.           }
146.
147.       void Powerflow(float bi[],float g[], int vectorgen[],int
   load[],float f[][3],int numvecgen[]){
148.
149.       float suma[Np];//definimos una variable para la suma de las columnas
   de la matriz fij
150.       int i;
151.       int j;
152.        //inicializamos el vector suma a cero
153.           for(j=0;j<=Np;j++){
154.               suma[j]=0;
155.               bi[j]=0;
156.           }
157.
158.               for(i=0;i<=Np;i++){
159.                   // si en nudo i es un generador entramos en este bucle
160.                   if(numvecgen[i]==1){
161.                       //sumamos la columna i del vector fij
162.                       suma[i]=+f[i][2];
163.                       bi[i]=-suma[i]+g[i]+bi[i];
164.                   }else{
165.                       suma[i]=+f[2][0]+f[2][1];
166.                       bi[i]=-suma[i]+bi[i]-load[i];
167.                   }
168.               }
169.       }
```