IE3048 Electrónica Digital 3 – Proyecto Final

SCADA

Introducción:

Este proyecto consistió en la realización de un sistema de Supervisión, Control y Adquisición de Datos (SCADA por sus siglas en inglés) constituido por tres Raspberry Pi y un módulo de internet de las cosas con un Arduino Nano 33 IoT. En este caso, dos Raspberry Pi 3B+ fueron utilizadas para como los RTUs (*remote terminal units*) y una Raspberry Pi 4B fue utilizada como historiador y HMI *(human-machine interface*). El objetivo de este sistema SCADA es almacenar múltiples eventos detectados por cada RTU en un historiador y desplegarlos visualmente para poder supervisar qué está sucediendo. Estos eventos pueden darse por cambios en sensores digitales o analógicos o por el módulo de internet de las cosas antes mencionado; cada RTU contó con cuatro sensores digitales (dos botones y dos interruptores), una analógica (la salida de un divisor de voltaje dado por un potenciómetro que representa el nivel de voltaje de una línea de transmisión), dos salidas digitales (dos LEDs) y su módulo IoT. Los RTUs tuvieron el propósito de enviar notificaciones periódicas con todos los eventos que llegaron a suceder en ese período al historiador, el cual se encargó de procesar esa información, guardarla y desplegarla en una interfaz gráfica amigable para el usuario.

Los eventos existentes son: uno de los botones se presionó, uno de los interruptores cambió de estado, la línea de transmisión se salió de sus valores de funcionamiento normal (de 0.5v a 2.5v), la línea de transmisión regresó a sus valores de funcionamiento normal, el módulo IoT mandó una señal y que el historiador mandó un comando manual para encender una señal digital.

Descripción:

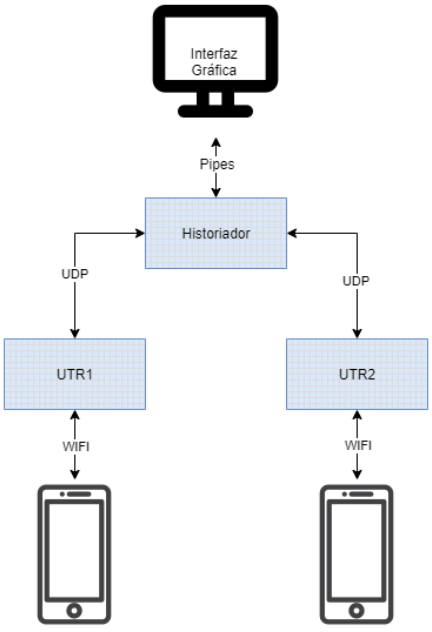

Figura 1: Diagrama general del proyecto

- Historiador

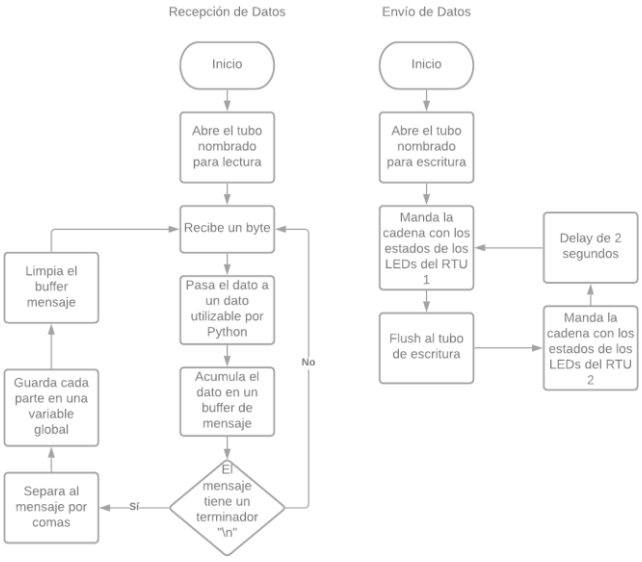

Figura 2: Diagrama flujo de los hilos del programa de Python

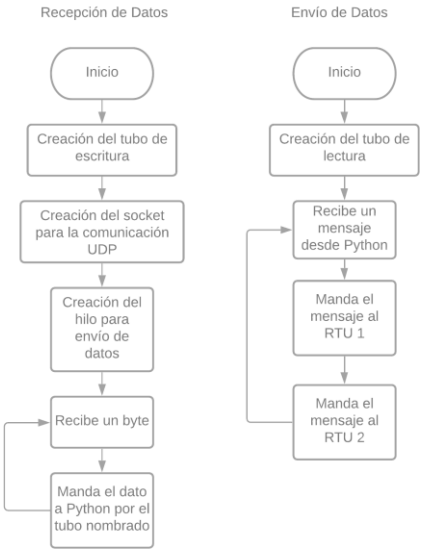

Figura 3: Diagrama flujo de los hilos del programa en C

Para el historiador se utilizó únicamente una Raspberry Pi 4B y un monitor para desplegar la interfaz conectados entre sí a través de un cable micro HDMI. Como se mencionó antes, en este caso, el historiador también hace la función de HMI. La función de historiador de recepción de datos y envío de comandos manuales se realizó en un código de C, el cual se comunicó con las RTUs a través del protocolo UDP. Para realizar esto se utilizaron dos hilos, uno para el envío de datos y otro para la recepción de datos. En el hilo principal se realizó la recepción de datos, los cuales iban siendo almacenados en un buffer para luego enviarlos a través de un tubo nombrado hacia un programa de Python. De forma similar, el segundo hilo iba recibiendo los comandos manuales desde Python a través de un tubo nombrado, los cuales luego eran enviados hacia las RTUs. Este programa de Python fue el que le dio al historiador su funcionalidad de HMI, puesto que se realizó de tal forma que desplegara una interfaz gráfica con la información más importante de forma ordenada y amigable con el usuario. La interfaz fue diseñada para mostrar el estado de los interruptores y del módulo IoT con sliders, el voltaje con un display LCD, la comparación de los voltajes de ambas RTUs en una gráfica de

tiempo real y los eventos con sus respectivas horas en un espacio desplazable. Para esta interfaz se utilizaron los módulos PyQt5 con su herramienta de QtDesigner para facilitar el diseño y pyqtgraph para implementar las gráficas de tiempo real. Finalmente, se realizó un código en QSS para darle estilo a la interfaz, el cual es un lenguaje similar a CSS, pero específico para Qt.

Puesto que el programa de Python no solo hacía el despliegue de datos, sino que tenía que recibir a través de tubos nombrados dichos datos desde el programa de C, fue necesario implementar múltiples hilos. En este caso, se hizo uso de tres hilos: el principal para desplegar la interfaz, un segundo hilo para recibir los datos de forma continua por un tubo de lectura y un tercer hilo para hacer el envío de los comandos solicitados por el usuario en la interfaz a través de un tubo de escritura. Asimismo, para lograr que la gráfica se actualizara en tiempo real, se usó la funcionalidad de temporizadores en Python para que cada segundo se actualizara la información.

Debido a que Python y C son lenguajes distintos que manejan sus estructuras de datos de forma distinta, fue necesario hacer un ajuste en Python al recibir y enviar los datos para que la información fuera útil en ambos programas. Para esto, se hizo uso del módulo struct de Python, el cual se encarga de hacer las conversiones necesarias entre tipos de datos de C a Python y viceversa. De no realizar estas conversiones, los programas recibirían la información de los tubos, pero no podrían desplegarla o desplegarían cosas sin sentido. Al mismo tiempo, esta diferencia entre los programas provocó que el archivo de Python únicamente pudiera leer byte por byte en lugar de hacer una lectura hasta encontrar un terminador de línea. Por esto, fue necesario tener una variable que acumulara la información que iba llegando hasta que la información recibida y traducida a estructuras de Python contuviera un terminador de línea. Este mensaje fue enviado desde una RTU con un formato especial, el cual consistía en mandar cada dato de cada sensor, el número de identificación de la RTU que mandó el mensaje y la hora en que se produjo el evento (cada dato separado por comas entre sí). Una vez se tenía este mensaje ya completo, el programa de Python se encargaba de separar la información (ayudándose de que los datos estaban separados por comas). Estos datos ya separados los almacenaba en variables globales para que la interfaz pudiera acceder a ellas y usarlas en el despliegue de datos. Finalmente, la interfaz utilizaba botones que, al ser presionados, se conectaban a una función que cambiaba el estado de una variable global (similar al funcionamiento de una interrupción del tipo *interrupt on change*). Esta variable global era luego accedida por el hilo de envío de datos y mandada a través del tubo al programa en C.

- UTR

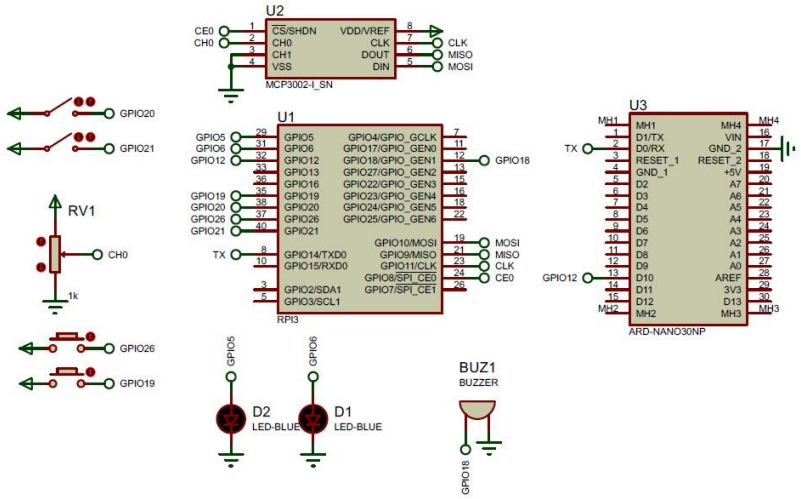

Figura 4: Esquemático de las UTR

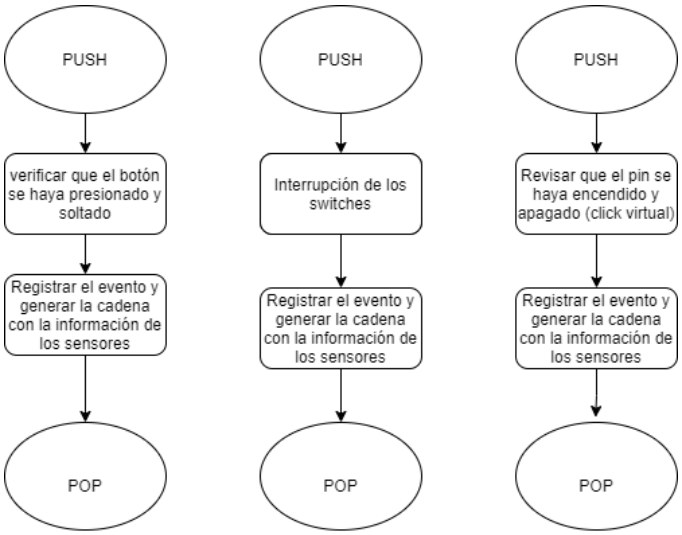

Figura 5: Diagrama de las interrupciones utilizadas en las UTR

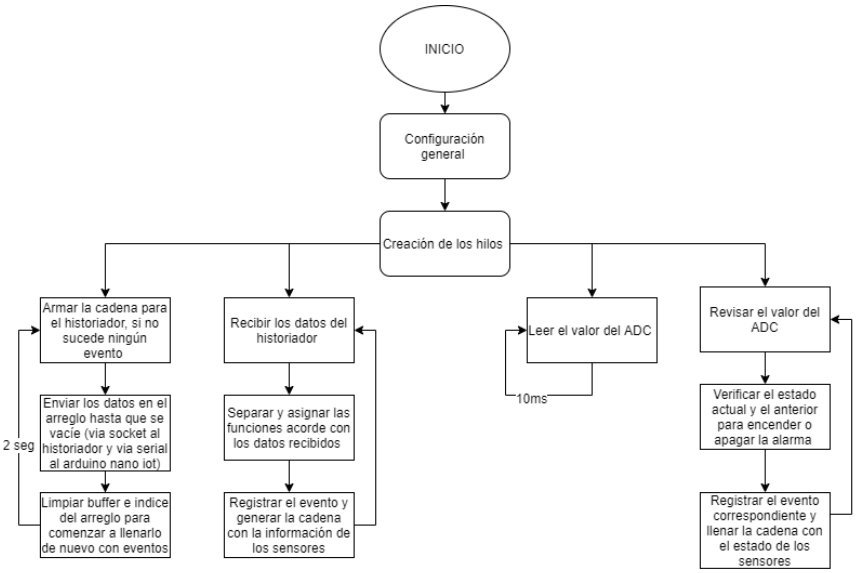

Figura 6: Diagrama de los hilos utilizados en las UTR

Como se puede observar en la Figura 2, cada UTR cuenta con 2 botones, 2 LEDs, 2 switches, un integrado MCP3002, un potenciómetro, un Arduino nano 33 IoT y una Raspberry Pi 3B+. El código principal fue realizado en la Raspberry, siendo esta donde se leen los valores de cada sensor y se envían al historiador a través de sockets. Para leer el estado de los botones, el estado de los switches y los cambios generados por el botón virtual, se utilizaron las interrupciones del puerto GPIO de la Raspberry. La razón de usar interrupciones es por el hecho que los puertos de la Raspberry están optimizados para esto y la implementación es sencilla y eficiente. Por otro lado, el programa cuenta con cuatro hilos. El main, en el cual se envían los datos, un hilo que únicamente se encarga de leer el valor del ADC, un hilo en el que se reciben los datos del historiador, y un hilo en el que se comparan los valores del ADC, para registrar el evento correspondiente y encender o apagar la alarma.

Para leer los eventos de los botones, en las funciones de interrupción se implementó un antirebote para que la variable indicadora únicamente fuera modificada cuando el botón fuera presionado y soltado, para leer el estado de los switches únicamente se guarda en la variable indicadora el estado del pin. Para el pin manejado por el Arduino, se realizó algo muy similar que, con los botones, con la diferencia que la variable indicadora funciona como un *toggle* cuando el botón es

presionado. En el caso de los eventos en el ADC se tuvo que implementar una rutina que únicamente registra como evento el cambio de región en el ADC, cuando el valor del ADC sale del rango de operación o cuando regresara al rango de operación. Para lograr ello, se implementó una especie de antirebote en el que se leía el estado actual y anterior del ADC para que el evento se registrar únicamente cuando estos fueran diferentes. Asimismo, para registrar el evento de recepción de datos se realizó una rutina similar a la mencionada anteriormente, dado que, por la implementación de la interfaz gráfica, se enviaba un flujo contante de datos al UTR, la cual únicamente tenía que registrar un evento cuando el valor actual recibido fuera distinto al anterior.

Para registrar y enviar los eventos se utilizó la siguiente metodología. Los estados de cada uno de los sensores se colocaron en variables globales, para que cualquier hilo o interrupción pudiera tener acceso a ellos y registrarlos en el momento en el que sucediera un evento. Se implemento un buffer común, al cual cada rutina de interrupción e hilo tenía acceso, para que cuando se detecte algún evento, se construya una cadena con el estado de todos los sensores en ese momento y se guarde en alguna posición del arreglo global. Luego, se incrementaba una variable global que indica el índice del arreglo en el que se guardaran los datos, con la finalidad de que cuando otro evento sucediera, la cadena con todos los valores se guarde en la posición siguiente.

Para enviar los datos, se utilizó un ciclo for, en el que se revisaba el valor de la variable global del índice del arreglo, para realizar un numero de envíos que corresponda exactamente al número de datos guardados en el arreglo, que correspondía al número de eventos que sucedieron antes que se venciera el periodo de envió.

Por el hecho de que la Raspberry pi no cuenta con un convertidor analógico a digital (ADC), para la lectura del potenciómetro se utilizó el integrado MCP3002, el cual funciona como modulo ADC externo, con la capacidad de enviar el valor de la conversión en 10 bits, a través del protocolo de comunicación SPI.

Para comunicar el Arduino 33 IoT con la Raspberry pi se utilizaron 2 tipos de comunicación diferentes. Para el botón, se envió un bit en paralelo del Arduino a la Raspberry, donde este pin era manejado directamente a través de la interfaz virtual en el celular y comunicación serial para enviar al Arduino todo lo que era enviado vía UDP al historiador, luego en el Arduino se leía el puerto serial y la lectura era enviada vía WIFI a una terminal virtual en el celular, para poder ver desde ahí el estatus de todos los sensores.

Experimentos:

Con la finalidad de garantizar que el proyecto funcionara adecuadamente, la implementación se segmento en bloques, en las que se verificaba el correcto funcionamiento de cada bloque antes de añadir uno nuevo. En el caso de la UTR, por el hecho que fue un tema que no se había trabajado previamente en laboratorios, primero se implementó la interrupción para un botón, una vez la interrupción funciono correctamente, se replicó la misma rutina para cada uno de los sensores que se fueran a través de las interrupciones del puerto GPIO. Seguido de esto, en un programa separado, se probó la lectura del módulo del ADC, por el hecho que también era la primera vez que se tenía contacto con este módulo y con la comunicación SPI en la Raspberry Pi. Al momento de validar que el módulo estuviera leyendo apropiadamente el divisor de voltaje en el potenciómetro, se procedió a realizar el experimento propuesto en la guía, el cual fue satisfactorio.

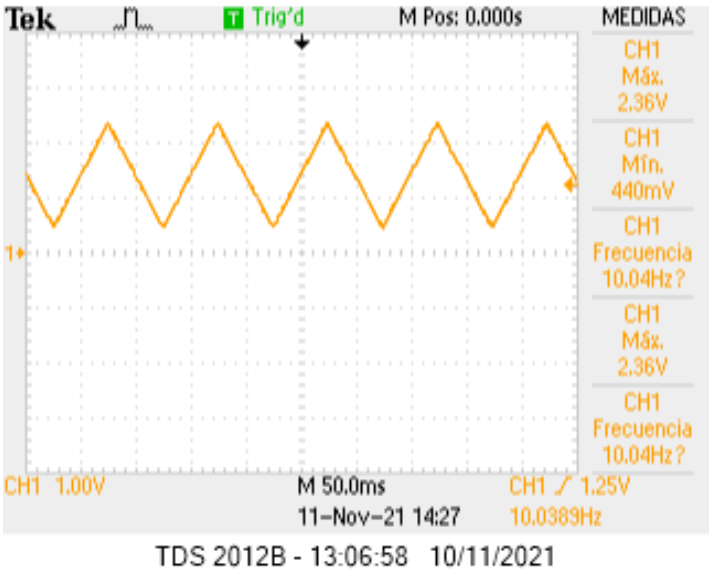

Figura 4: Señal triangular del generador de funciones

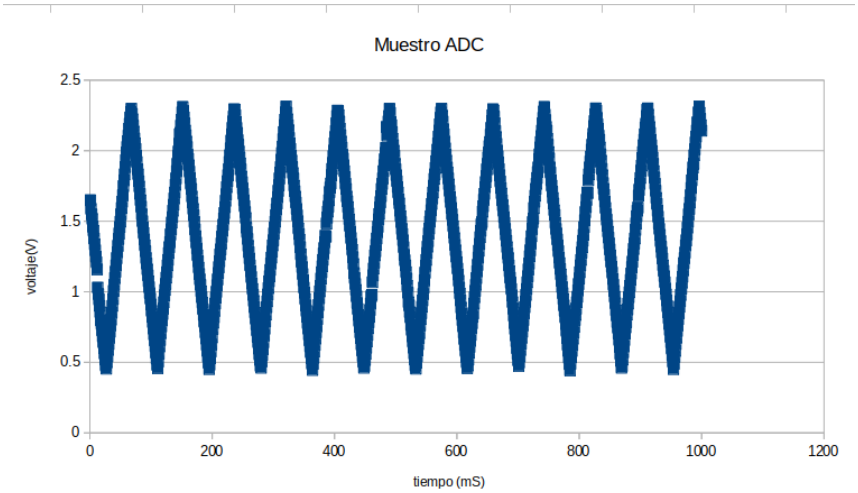

Figura 5: Grafica de las lecturas del ADC

Una vez, ambos programas estuvieron listos, se juntaron ambos programas en un mismo código, verificando que las interrupciones no afectaran las lecturas del ADC y viceversa. Luego de ello, se procedió a implementar la rutina de comparación con los valores del ADC y la alarma. Ya con esto se dedicó un tiempo en cambiar el estado del potenciómetro para verificar que la alarma sonara en el momento correcto. Ya con los sensores y el ADC funcionando correctamente, se agregó la parte para el envío y recepción de datos. Inicialmente, con la ayuda de un programa brindado por el profesor para un laboratorio (Lab8_cliente_RPI3B+). Una vez se enviaron distintos caracteres, de distintos tamaños y a diferente periodo de envió, verificando que todas las iteraciones hayan sido exitosas, se realizó lo mismo para el hilo de recepción, al que se le enviaron cadenas de diferente extensión y a diferente periodo, verificando que todos los datos fueran recibidos correctamente. Seguido de esto, se implementó el botón controlado vía wifi por el Arduino nano IoT, para el cual se encendía un pin en el Arduino que estaba conectado de forma paralela a la Raspberry. Para este segmento se utilizó una interrupción similar a la de los botones, en la que se verificó que el botón virtual en el celular fuera leído independientemente de la velocidad a la que fuera presionado, o tiempo que durara la

presión. Es importante mencionar que las pruebas fueron exitosas utilizando una red WIFI con una velocidad alta, sin embargo, si se tiene una red WIFI más lenta (como la de la universidad), si se dio el caso en el que algunas veces no se detectaran los cambios en el botón. Ya con todos los sensores funcionando en conjunto, se implementó el buffer común de registro de datos, en el que, vía un ciclo for en el main, se imprimían cada 2 segundos (simulando el envío al historiador). Luego de verificar que todos los eventos fueran detectados con el *time stamp* correcto, accionando varios sensores en menos de dos segundos para ver si todos eran registrados y probar al menos 20 veces cada sensor y tipo de evento, se concluyó que la UTR ya estaba lista para enlazarse con el historiador.

De forma paralela, se realizaron todas las pruebas pertinentes en el historiador, antes de enlazarlo con la RTU. Por el hecho que desde un principio se tuvo la idea de implementar una interfaz creada en Python para mostrar los datos del RTU y enviar los comandos. Lo primero que se probó fue el envío de datos a través de tubos nombrados desde un proceso creado en C a un proceso en Python. Acá se realizó una gran variedad de pruebas, dado que no se sabía con certeza como Python interpretaba los datos que eran enviados desde el programa en C. Inicialmente se intentó mandar un mensaje simple e imprimirlo para poder descifrar la forma en que los datos llegaban de un lugar a otro. El proceso de envió se realizó las suficientes veces como para estar seguros de que lo que se estaba recibiendo en Python era exactamente lo que se estaba enviando en el programa de C. Una vez esto quedó funcional, se implementó la comunicación por sockets para enviar y recibir datos a las RTUs, esta se validó utilizando el mismo programa que en la RTU, para que independiente de la extensión del mensaje y el periodo de envió o recepción, la comunicación fuera correcta. Ya con esto listo, se estableció la comunicación con las UTRs y se verificó que estuvieran llegando datos al historiador. Luego, se verificaron los *time stamps* para verificar que el protocolo NTP estuviera funcionando apropiadamente entre las UTRs. Ya con la comunicación establecida, se probaron al menos 20 veces en cada UTR cada uno de los eventos posibles, así como el registro de más de un evento antes del periodo de envió. Finalmente, se verifico que las UTRs recibieran bien los datos, verificando que leyeran apropiadamente el mensaje codificado y ejecutaran las instrucciones pertinentes. De este modo, se dio por finalizado el proyecto con los requerimientos solicitados y se procedió a la implementación de los puntos extras.

Inicialmente se implementó para cada RTU comunicación serial para que cada cadena que era enviada al historiador fuera enviada al Arduino nano 33 IoT, de modo que este la enviara vía WIFI a una terminal en el celular. Para que se pudiera ver que era lo que estaba sucediendo en la RTU, en el historiador, pero también en el celular.

Y, por último, como la parte más compleja del proyecto se implementó la interfaz gráfica, para lo cual fue necesario realizar múltiples experimentos. Primero, se realizó una interfaz simple con sliders que cambiaran de valor con algún tipo de señal de la terminal y botones que al presionarlos imprimieran algo en la terminal. Asimismo, se realizaron pruebas para la gráfica en tiempo real; primero se revisó que la gráfica se actualizara con datos aleatorios, luego se revisó que la gráfica sí se actualizara con los datos del ADC y finalmente se colocaron ambas gráficas en el mismo plano con títulos en los ejes y una leyenda pertinente. Luego, se juntaron ambas pruebas para verificar que funcionaran en conjunto.

Debido a que se usaron múltiples hilos, temporizadores y una interfaz gráfica en un proceso y otro proceso también con multi hilos, el tiempo de respuesta del historiador a los mensajes de las RTUs y el envío de comandos manuales era lento por momentos. Además, esta respuesta también era lenta por momentos o incluso llegaba a perderse según el nivel de saturación de la red que se estaba utilizando. Sin embargo, en una red con solo las RTUs, el historiador, los Arduino IoT y un par de

celulares conectados, se logró observar que no había una pérdida de datos notoria entre dispositivos, es decir, si se presionaban los botones 10 veces, las 10 notificaciones llegaban siempre al historiador.

Resultados:

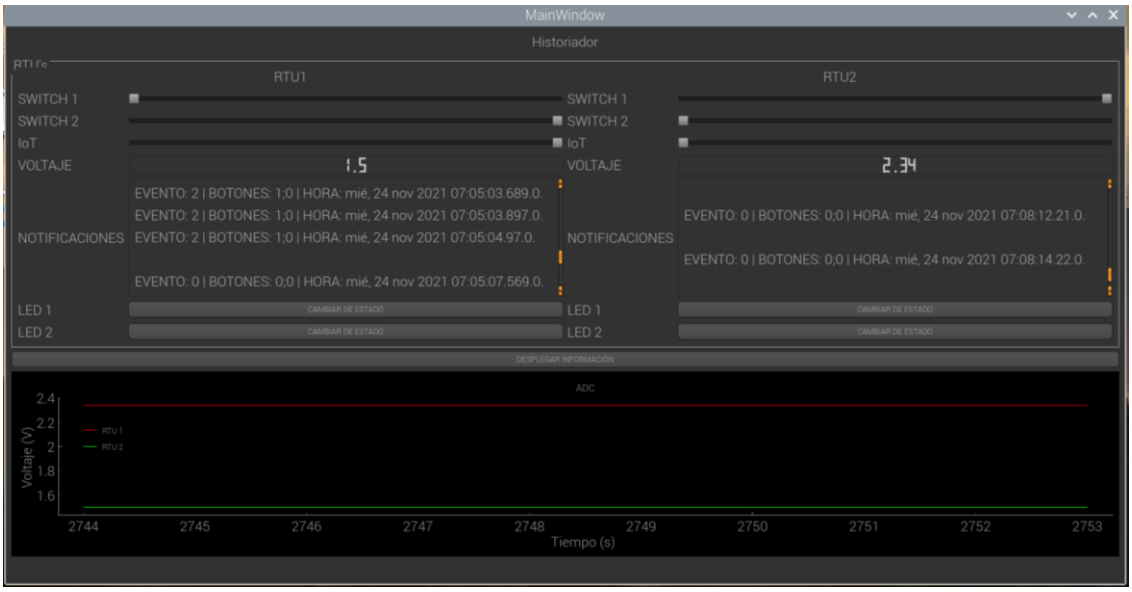

Figura 6. Interfaz Gráfica de Usuario (GUI) del Historiador

Para mostrar el funcionamiento del proyecto y los resultados, se grabó un vídeo y se subió a YouTube. Este puede encontrarse en el siguiente link: [Proyecto Final - sistema SCADA]

Discusión:

Al momento de finalizar el proyecto y en cada una de las partes realizadas del mismo, se obtuvieron los resultados esperados. Es importante mencionar que antes de comenzar la implementación, se realizó una investigación acerca de cada una de las funciones a utilizarse, en el caso que fueran vistas en clase, se consultaron los documentos del curso y si no, se investigó en otras

fuentes. Esto para tener presente la forma correcta para llevar a cabo cada parte de la implementación y poder comprender en su totalidad lo que estuviera sucediendo en el programa. Esto no solo fue de utilidad para optimizar el tiempo de implementación, sino también el hecho de poder predecir los resultados aumenta las probabilidades de tener éxito.

En la práctica, siempre existen impedimentos que muchas veces dificultan la experimentación y hacen que los resultados varíen respecto del modelo teórico. Respecto de la RTU, por el hecho de que gran parte de lo que se tenía que realizar fue tema explicado en clase y laboratorio, no se tuvieron mayores impedimentos en su implementación. Respecto de los temas nuevos, como era el uso del integrado MCP3002, a la comunicación SPI, el Arduino nano 33 IoT y la comunicación UART de las Raspberry, si se tuvieron algunos problemas mínimos. En el caso del integrado MCP3002 y la comunicación SPI, aunque no fueron temas vistos en clase, se nos brindaron las herramientas suficientes para implementarlo sin presentar mayores dificultades. Otro aspecto que fue de mucha ayuda fue la librería para SPI de wiringPi, por el hecho que las funciones en esa librería ahorran considerablemente el trabajo manual para establecer dicha comunicación. Para el caso del uso del Arduino IoT 33, si se experimentaron algunos problemas que llevaron bastante tiempo resolver, la gran mayoría fueron causados por descuidos en la interconexión del Hardware. Además, se utilizó la aplicación de blynk para controlar al Arduino nano IoT 33, por el hecho de ser un tema nuevo, tomo bastante tiempo su implementación y presento algunos problemas a la hora de implementar. Sin embargo, todos los problemas se resolvieron investigando de una manera más profunda para comprender cómo funcionaba todo apropiadamente.

De parte del historiador, la mayor cantidad de problemas se dieron en la parte de la interfaz gráfica y de la comunicación entre procesos a través de tubos nombrados. El primer problema fue que los tipos de datos de C no se traducían bien a los tipos de datos en Python, lo cual se resolvió investigando sobre comunicación entre estos lenguajes, lo cual llevó a encontrar la librería struct, la cual se enfrentaba y solucionaba el problema efectivamente. Luego, con la interfaz, se tuvo el problema que algunos de los paquetes a utilizar ya no estaban siendo actualizados para Raspberry por las compañías desarrolladoras, por lo que la descarga e instalación de los paquetes más recientes de Qt para el funcionamiento de la librería pyqtgraph tuvo que hacerse de forma manual, lo que llevó a investigar sobre cómo acceder a archivos en github desde la terminal, a descomprimir dichos archivos y a cambiar las variables y ruta de entorno modificando el archivo bashrc. Luego de solucionar esto, se encontró el problema de colocar múltiples gráficas en el mismo plano para que ambos voltajes se graficaran en tiempo real, para lo cual fue necesario leer una gran parte de la documentación de la librería pyqtgraph para entender el funcionamiento de la misma. Luego, se encontró con el problema de que los datos desde Python no se estaban enviando correctamente hacia C, lo cual se solucionó al agregar la instrucción flush a la escritura de datos y colocando un delay. Por otro lado, existía el problema que los datos llegaban en grupos cada dos segundos y no de forma continua, por lo que fue necesario utilizar un buffer global para que cuando se actualizaran los datos no se perdiera ninguno y todas las notificaciones llegaran. Finalmente, existía el problema que, puesto que la actualización de los datos en las gráficas y las notificaciones se daba cada segundo y la recepción de datos cada dos, los datos se desplegaban dos veces, lo cual se solucionó haciendo uso de *regular expressions*, con las cuales se verificó que un evento no se colocara si su *time stamp* era exactamente igual a uno mostrado anteriormente.

Este proyecto represento la unificación de todos los conceptos vistos a lo largo del curso, de modo que nos permitió aplicar en un modelo de la vida real todo lo aprendido y terminar de comprender la razón del curso sea parte de nuestro pensum y la importancia que tiene a nivel profesional. Algunos conceptos, como las interrupciones o aplicación del internet de las cosas, habían

sido tratados en cursos anteriores y este proyecto nos permitió llevar esos conocimientos a otro nivel. Asi mismo, fue de mucha ayuda para comprender en su totalidad los últimos temas vistos durante el curso, como lo fueron los pipes y la comunicación por sockets. A pesar de que la elaboración fue tardada, el sentimiento de finalizarlo fue increíble, no solo por el hecho de haber terminado la tarea sino por el logro académico que implicaba finalizar con éxito un proyecto de tal magnitud.

A pesar de que como grupo estamos muy satisfechos con el proyecto, con la forma en la que se nos solicitó implementarlo y con las cosas que este debería incluir, nos parece que algunas cosas se hubieran podio hacer de una mejor manera. En el caso de los sensores utilizados UTR, consideramos que se hubiesen podido implementar sensores un poco más complejos, aprovechando más protocolos de comunicación de la Raspberry, como hubiese sido el protocolo bluetooth, I2C y wifi. Por otro lado, consideramos que tomando en cuenta la herramienta del Arduino, se hubiera podido explotar a otro nivel con aplicaciones que nos permitieran desarrollar algo mucho más complejo e interesante. Finalmente, consideramos que la interfaz gráfica en el historiador fue un factor que le dio auge a nuestro proyecto, pero por el hecho de que no se tenían una gran cantidad de conceptos previos, lograr implementarla significo una gran cantidad de tiempo, de modo que si se hubieran trabajado herramientas en clase o algo por el estilo, hubiera ayudado a que los grupos que no realizaron una interfaz, se sintieran motivados a hacerlo y a los que sí, pudiéramos realizar algo mucho más semejante a algún modelo en la vida real. Como grupo comprendemos que, por las limitaciones de tiempo en el curso, el diseño del proyecto fue bastante bueno, sin embargo, quedan las recomendaciones para futuras oportunidades en las que se tenga más tiempo para su implementación.

Conclusiones:

- El proyecto nos sirvió como herramienta para validar conceptos vistos en cursos anteriores y comprender en su totalidad los conceptos vistos en este curso.
- Se logró implementar correctamente cada uno de los requerimientos solicitados y requerimientos adicionales significativos para ser acreedores a puntos extras.

Referencias:

Campagnola, L. (n.d.). *pyqtgraph*. Retrieved from Plotting in pyqtgraph: https://pyqtgraph.readthedocs.io/en/latest/index.html

Python. (n.d.). *struct — Interpret bytes as packed binary data*. Retrieved from https://docs.python.org/3/library/struct.html#struct.calcsize

StackOverflow. (n.d.). Retrieved from Interprocess communication using pipes between C and python - only newlines are printing: https://stackoverflow.com/questions/65582623/interprocess-communication-using-pipes-between-c-and-python-only-newlines-are

The Qt Company. (n.d.). *Qt Documentation*. Retrieved from https://doc.qt.io

www.unixtimer.com. (n.d.). *Unix & Epoch Timestamp Conversion Tools*. Retrieved from https://www.unixtimer.com/?gclid=Cj0KCQiAhMOMBhDhARIsAPVml-GAELdHgy8vWK9kz3ozN-ydBG8xmk5uYs0fWHQ13s5JihpA6THKUUQaAmCaEALw_wcB