CENTRO DE INGENIERÍA Y DESARROLLO INDUSTRIAL

REPORTE DE PROYECTO INDUSTRIAL

"Interfaz gráfica y electrónica para visualizar señal ultrasónica en A-scan en Raspberry Pi"

QUE PARA OBTENER

LA ESPECIALIDAD TECNÓLOGO EN MECATRÓNICA

PRESENTA

Ing. Antonio de Jesús Aguilar Mota

Tutor Académico

Dr. Jorge Alberto Soto Cajiga

Tutor Planta

Ing. Luis Alberto Carmona Martínez

QUERETARO, QRO. AGOSTO 2017.

Agradecimientos

Agradezco a mi madre por seguirme apoyando, aunque estemos distantes.

Agradezco a mi padre por confiar en mi.

Agradezco a todas las personas que me apoyaron en el desarrollo de este proyecto, haciendo mención de:

- Dr. Jorge Alberto Soto Cajiga, por su apoyo y ser mi asesor.
- Ing. Luis Alberto Carmona Martínez, por su apoyo y ser mi asesor.
- Ing. Juan Pedro Max Muñoz Bustos, por su instrucción en Python.
- Dr. Leonardo Barriga Rodríguez, por su instrucción en Python.
- Mtro. Noé Amir Rodríguez Olivares, por sus revisiones.
- Mtro. Luis Antonio Díaz Jiménez, por sus revisiones.
- Mtro. Alejandro Gómez Hernández, por su instrucción en Altium.

AUTORIZACIÓN
PUBLICACIÓN EN FORMATO ELECTRÓNICO DE TESIS

Fecha: 28/08/17

El que suscribe Antonio de Jesús Aguilar Mota
Alumno (a) ……
AUMA939524HVZGTN00 789060
CURP …… CVU……

0000-0003-2974-6407
ORCID ……

Correo electrónico (opcional) ajm3000@outlook.com
……

Egresado (a) de Centro de Ingeniería y Desarrollo Indsutrial

……

Autor de la Tesis titulada
Interfaz gráfica y electrónica para visualizar señal ultrasónica en A-Scan en
……
Raspberry Pi
……

Por medio del presente documento autorizo[1] en forma gratuita y permanente a que la Tesis arriba citada sea divulgada y reproducida para publicarla mediante almacenamiento electrónico que permita el acceso al público a leerla y conocerla visualmente, así como a comunicarla públicamente en Página Web.
La única contraprestación que condiciona la presente autorización es la del reconocimiento del nombre del autor en la publicación que se haga de la misma.

Atentamente

Antonio de Jesús Aguilar Mota

[1] Ley Federal de Derechos de Autor
Para obtener tu ORCID regístrate en: https://orcid.org/register

Índice

Índice de figuras

Índice de Códigos

Resumen

Este trabajo consiste en el desarrollo de una interfaz gráfica para representar señales en 2D recibidas por un puerto serial en la plataforma Raspberry Pi, enviadas desde un módulo MSP430G2553 Launchpad, teniendo en mente una implementación a futuro, en la cual un dispositivo FPGA(Field Programmable Gate Array) capture señales de naturaleza ultrasónica y sean los valores capturados emitidos por dicho puerto serial. Este proyecto viene derivado del interés por parte de CIDESI en querer desarrollar un sistema de inspección por ultrasonido portátil. Es importante destacar que el presente trabajo no está limitado a esta implementación, y puede servir de base para otros proyectos que requieran una interfaz gráfica.

En este trabajo se define paso a paso cómo desarrollar el programa de interfaz gráfica en el lenguaje de programación Python, el cual es de fácil aprendizaje, rápida implementación y buena integración con el paradigma de programación orientada a objetos. Se recomienda al lector tener conocimiento intermedio de Python, un conocimiento básico de Linux y de las librerías para desarrollo de interfaces gráficas Tkinter.

En el presente trabajo se tienen las consideraciones siguientes:

- Configuración del entorno de desarrollo.
- Desarrollo paso a paso de los distintos elementos de la interfaz gráfica.
- Pruebas de recepción serial.

1. Introducción

Muchos materiales empleados en la industria se encuentran expuestos, dependiendo su localización y el trato que se les de, a las inmediaciones del tiempo y al uso rudo; haciendo que dichos materiales alcancen niveles críticos de estrés que comprometen la integridad de los mismos. Esto es un riesgo aún mayor si dichos materiales sirven para formar parte de estructuras con las que interactúen personas frecuentemente. Es preciso asegurar la integridad de estos materiales y que estos estén dentro de los márgenes permitidos para su empleo.

Para determinar el estado de un material se hace uso de los denominados ensayos no destructivos, que se define como cualquier tipo de prueba practicada a un material que no altere de forma permanente sus propiedades físicas, químicas, mecánicas o dimensionales.[1]. Un ejemplo de estos es haciendo uso de señales ultrasónicas, denominado ultrasonido industrial. Este un método muy utilizado ya que es de tipo volumétrico; esto es, permite examinar todo el volumen o elemento que se desea inspeccionar.

La aplicación de ultrasonidos se conoce sobre todo en el ámbito del diagnóstico médico, cuado se realizan fotografías de bebés que aún no han nacido. En la industria, los ultrasonidos se utilizan en numerosos procesos, por ejemplo, para limpiar, soldar plásticos y metales, cortar, conformar, comprobar materiales, separar, mezclar, desgasificar, pulverizar, localizar, medir y mucho más.

Los ultrasonidos son sonidos imperceptibles para el oído humano. El sonido representa la expansión de oscilaciones mínimas de presión y densidad en un medio elástico (gases, líquidos o sólidos). El número de oscilaciones en el espacio de un segundo se denomina frecuencia, cuya unidad es el hercio (Hz). El rango de frecuencia perceptible por el oído humano se sitúa entre los 16 y los 20 000 Hz. El rango superior a 20 000 Hz se denomina ultrasonido.[2]

La inspección por ultrasonido se define como un procedimiento de inspección no destructivo de tipo mecánico, y su funcionamiento se basa en la impedancia acústica, la que se manifiesta como el producto de la velocidad máxima de propagación del sonido y la densidad del material. Cuando se inventó este procedimiento, se medía la disminución de intensidad de energía acústica cuando se hacían viajar ondas supersónicas en un material, requiriéndose el empleo de un emisor y un receptor. Actualmente se utiliza un único aparato que funciona como emisor y receptor, basándose en la propiedad característica del sonido de reflejarse al alcanzar una interfase acústica.[3]

Un ejemplo de esto es el robot de inspección de ductos desarrollado en CIDESI por el departamento de Energía y cuya principal función es reportar de todas las variaciones que presente la superficie interna del ducto en el que sea ingresado a lo largo de su trayecto.[4]. De la misma manera en CIDESI, se pretende desarrollar un dispositivo de inspección ultrasónica portátil, y es la causa principal de la necesidad de este proyecto; pues este requiere de una interfaz gráfica para visualizar las señales recibidas. Esto se pretende mediante el empleo de una Raspberry Pi[5], la cual cuenta con periféricos adecuados para el desarrollo de este trabajo.

1.1. Propósito

El propósito de este documento es ayudar al desarrollador de aplicaciones que supongan interfaces gráficas en Raspberry Pi, a implementar de forma sencilla y rápida una plantilla desarrollada en el lenguaje programación Python. Se hace énfasis en el hecho de que este proyecto está enfocado para ser futuramente implementado en un dispositivo de inspección por ultrasonido portátil; y es por ello que el diseño final tendrá un panel de control similar al desplegado por estos dispositivos.

1.2. Convenciones del documento

- Este documento está estructurado de forma que el desarrollador siga una implementación de forma secuencial.
- Se trata de explicar cada paso llevado de la forma más detallada posible.
- Se hace una descripción general del proyecto del cuál se desprende este manual.
- Se explica y detalla la forma correcta para la configuración del entorno de desarrollo y las plataformas involucradas.
- En las secciones que se explica la programación de la interfaz, se muestra la forma recomendada por el autor para el desarrollo de la aplicación.

  Primero se definen las librerías.

  Posteriormente el código principal se detalla y se explican los elementos del mismo.

  Los elementos del código se representan en forma jerárquica, de forma que primero se detallan los elementos correspondientes de mayor importancia y posteriormente sus respectivas dependencias.

1.3. Audiencia y recomendaciones de lectura

Este documento está enfocado a desarrolladores que quieran implementar aplicaciones con interfaces gráficas para la plataforma Raspberry Pi, desarrolladas en el lenguaje de programación Python 2.7. Es preciso que posean un conocimiento intermedio del lenguaje de programación y un conocimiento básico de la librería para aplicaciones gráficas Tkinter. De igual manera, se incluyen en las referencias toda la bibliografía empleada para asistencia.

Así mismo, se tiene en consideración a interesados e involucrados en desarrollos de proyectos que supongan un sistema de ultrasonido portátil, pues la plataforma Raspberry Pi, por su tamaño y capacidad de procesamiento se presta par esta tarea, ya que cuenta con periféricos a los cuales se les pueden conectar aditamentos para desplegar elementos visuales, ejemplo de esto es la pantalla de 7.^{en} la que se realizó este proyecto.

Se recomienda leer secuencialmente este documento para llevar un seguimiento de los pasos realizados.

1.4. Alcance del documento

Este manual tiene como objetivo servir a un proyecto de representación gráfica para visualizar una señal ultrasónica en A-Scan. Este proyecto; a su vez, tiene como propósito servir de interfaz gráfica para valores recibidos por medio de comunicación serial desde otros dispositivos compatibles. No obstante, no se está limitado a esto ya que el dispositivo empleado, la Raspberry Pi, cuenta con distintos periféricos que permiten otras formas de comunicación. Así mismo, se da una idea de como realizar una comunicación por medio del puerto serial. El actual proyecto se enfoca a la comunicación con un módulo de Texas Instruments MSP430G2553 Launchpad[6] para probar la comunicación; para realizar la comunicación con los periféricos que soporten este tipo de comunicación, es preciso consultar sus respectivos documentos y/o manuales .

1.5. Planteamiento del problema

Actualmente; en CIDESI, se han desarrollado sistemas ultrasónicos[7] que se encuentran dirigidos a la inspección de ductos.[8] A su vez, se está desarrollando una arquitectura para el procesamiento y visualización de señales ultrasónicas. Este desarrollo se apoya en la plataforma Raspberry Pi, para poder visualizar dichas señales.

Ya que se carece de una interfaz gráfica en el desarrollo previamente mencionado en el cual se puedan desplegar las señales ultrasónicas, el propósito de este proyecto es establecer las bases de programación que permitan la representación de estas señales, empleando las librerías gráficas del lenguaje de programación Python[10] y una pantalla táctil, compatible con Raspberry Pi. Se requiere la representación gráfica de una señal adquirida por un FPGA(Field Programmable Gate Array), mediante un transductor. La plataforma en donde se requiere la visualización de la señal es un Raspberry Pi conectada a una pantalla LCD de 7 pulgadas.

La complejidad del proyecto radica en que una sola persona deba desarrollar tanto la arquitectura electrónica como la configuración del FPGA, y a su vez la visualización de la señal en Raspberry Pi, debiendo completar 3 sistemas distintos a la vez. Es por ello que se recomienda la asignación de cada tarea a personas distintas, en este aspecto se cubre el desarrollo de la interfaz en Raspberry Pi, quedando pendientes los sistemas de la FPGA y la arquitectura electrónica.

1.6. Justificación

La realización de este proyecto asentará las bases para seguir el desarrollo del sistema de procesamiento y visualización de señales ultrasónicas, que se desarrolla actualmente en CIDESI. La plataforma de computación Rasperry Pi es una solución de bajo costo que ofrece recursos de cómputo Linux[9]; lo cual permite emplearla como una computadora con todo y sistema operativo. Por si eso no fuera suficiente, la misma cuenta con pines de propósito general para conexión a otros dispositivos electrónicos. Además de esto, se cuenta con puertos para conexión a dispositivos como la cámara o la pantalla LCD de 7” compatibles, por lo que no hay que adquirir un hardware no dedicado y acoplarlo al sistema.

Este proyecto surge de la necesidad de representar gráficamente los datos enviados por un dispositivo FPGA, el cuál está conectado a un transductor palpador de superficies metálicas. Es probable que se crea que el mismo FPGA pueda cumplir las funciones de sensor y graficador a la vez; sin embargo, se trata de un sistema de captura de datos en tiempo real, por lo que es imperativo que el FPGA tenga lecturas

sin interrupciones. Es por ello la necesidad de este proyecto, una interfaz gráfica desarrollada en Python para la plataforma Raspberry Pi que; auxiliándose de la pantalla LCD de 7" y una comunicación serial, da pauta a una infinidad de implementaciones que van más allá de la lectura de datos.

La elección de la Raspberry Pi supone una ventaja, pues aunado con las características previamente descritas, cuenta nativamente con el lenguaje de programación Python de forma nativa, por lo que está completamente soportado. Los programas desarrollados en este lenguaje son de fácil escalabilidad, su curva de aprendizaje no es elevada y es adecuado para la programación orientada a objetos.

1.7. Objetivo general

El objetivo general de este proyecto es diseñar una interfaz gráfica que permita al usuario/desarrollador visualizar señales dibujadas en una pantalla; empleando librerías gráficas.

1.7.1. Objetivos específicos

1. Diseñar una interfaz gráfica en la herramienta Tkinter[17], incluida en el lenguaje de programación Python.
2. Implementar una comunicación serial entre Raspberry Pi y un dispositivo externo que cuente con esta misma característica. Para propósito de pruebas se emplea el módulo MSP430G2553 Launchpad.
3. Escribir un documento técnico sobre el proyecto realizado.

1.8. Alcances

Desarrollar una interfaz gráfica capaz de mostrar una gráfica en 2D, que pueda ser ejecutada en un programa desarrollado en el lenguaje de programación Python, que a su vez esté dentro del entorno Raspbian y que pueda comunicarse externamente con dispositivos seriales. Este trabajo se considera terminado con la conclusión del código para la implementación de la interfaz gráfica capaz de recibir la señal por el puerto serial, habiendo realizado el despliegue de una señal que haya sido recibida por el puerto serial desde un MSP430G2553 Launchpad. Así también se entrega el código de este y un esquemático para mostrar la conexión.

1.9. Limitaciones

Lo que no se tiene en cuenta:

- El prototipo final; si bien la idea original es implementarlo en un módulo de sensado ultrasónico basado en FPGA, este proyecto queda como una plantilla para interfaces de proyectos que requieran interacción humana y; por lo tanto, su utilización final queda abierta a las necesidades del desarrollador.
- La fuente de energía. Para este proyecto se utilizó únicamente la fuente de alimentación de un Cubo convertidor de AC a 5V DC; que es la energía a la que se alimenta la Raspberry Pi. Es importante destacar que la corriente mínima de alimentación sí debe ser de al menos 2A como mínimo, al combinar el consumo de la Raspberry Pi con su módulo LCD; de no seguirse esta regla, el módulo regulador integrado en la Raspberry Pi puede sufrir alteraciones en su funcionamiento o llegar a un desperfecto permanente.

- Conexión a internet por Wi-Fi. La Raspberry Pi a lo largo del desarrollo de este proyecto se conectó a la internet mediante una computadora que compartía su conexión Wi-Fi a través de Ethernet; sin embargo, nunca se probaron las capacidades de la conexión inalámbricas autónomas de la Raspberry Pi, principalmente por el entorno de red tan restringido en el que se desarrolló.

Otras limitantes:

- Se está restringido a trabajar en el sistema operativo Debian, por lo que sólo se podrán ejecutar aplicaciones para sistemas basados en Linux.
- Al estar en un entorno basado en Linux, es muy dificíl la implementación de aplicaciones de ejecución nativa en Windows, por lo que se recomienda buscar una herramienta alternativa
- La Raspberry Pi cuenta con una memoria Ram de 1Gb, y está a su vez es compartida con la memoria de gráficos, por lo que es un elemento muy importante a considerar para el desarrollo de aplicaciones.

1.10. Cronograma de Actividades

Mayo	Junio	Julio	Agosto
Investigación del lenguaje de programación Python.	Programación de interfaz gráfica	Programación de comunicación serial	Pruebas y Documentación

1.11. Materiales Utilizados

1. Hardware:

   Raspberry Pi 3.

   Pantalla LCD 7”.

   Conversor AC/DC 5v USB.

   Cable ethernet.

   Cable microusb.

   Cables “Dupont”.

   Laptop.

   FTDI.

   MSP430G2553 Launchpad.

2. Software

   Raspbian (Debian).

   Python.

   Librerías especiales (definidas en el desarrollo de este proyecto y en este documento).

   Putty.

   Windows 7.

   Code Composer Studio.

2. Marco teórico

Los ensayos no destructivos son técnicas de inspección que se utilizan para verificar la sanidad interna y externa de piezas, componentes y materiales sin deteriorarlos ni alterar o afectar en forma permanente sus propiedades físicas, químicas y mecánicas. Los ensayos no destructivos pueden ser usados en la materia prima, durante el proceso de fabricación o en el producto final, así como para encontrar discontinuidades que se generan durante el servicio, lo cual permite la remoción previa de la pieza dañada y prevenir así un desastre. Los ensayos no destructivos son usados para producir productos más confiables y seguros. Por ejemplo, en la producción de aeronaves, en la tecnología nuclear, en la exploración espacial, en tanques y recipientes a presión, donde es esencial tener una máxima confiabilidad.[11]

El ultrasonido industrial, mostrado en la figura 1, es un método de inspección volumétrico, ya que se puede examinar todo el volumen o elemento que se desea inspeccionar. Los ensayos superficiales, como partículas magnéticas, sólo pueden detectar discontinuidades abiertas a la superficie o muy cercanas a ella. El ultrasonido son vibraciones mecánicas que se transmiten en el material por medio de ondas de la misma naturaleza que el sonido, pero con frecuencia mayor a los 20000 ciclos/segundo (Hz), es decir fuera del rango audible del oído humano.[11]

Figura 1: Prueba de Ultrasonido Industrial.[12]

El origen de la técnica de inspección ultrasónica se remonta al conocido ensayo de percusión de la muestra con un martillo y la percepción del sonido emitido. Por muchos años, fue común observar a los empleados del ferrocarril golpeando las ruedas de los vagones con un martillo ligero, con el fin de encontrar discontinuidades, sin embargo, los ensayos de sonoridad son muy simples para la detección de discontinuidades pequeñas en la forma precisa.[11]

El uso de ondas ultrasónicas para encontrar defectos en objetos de metal fue propuesto primeramente por Sokolov en 1929. En 1930 se reconoció el uso del ultrasonido como ensayo no destructivo. Investigadores rusos y alemanes desarrollaron el método de inspección ultrasónica con el cual se pueden detectar discontinuidades, pero una de las limitaciones que tenían estos primeros equipos era que tanto la superficie frontal como la posterior debían ser accesibles. Un acceso muy significativo se hizo cuando, casi al mismo tiempo, Firestone en América y Sproule en Inglaterra, introdujeron primeramente un detector de fallas pulso - eco, el cual requiere el acceso sólo por un lado. El sistema pulso eco, sigue siendo el más popular en ensayos no destructivos con ultrasonido.[11]

2.1. Aplicaciones

La inspección ultrasónica es uno de los métodos de inspección no destructiva más ampliamente usados y encuentra aplicaciones en los siguientes ramos industriales:

- Biología: Pregerminación de semillas, homogeneización de leche, putrefacción interna de troncos de árboles, medición de capas de tocino y muslos en porcino vivo.
- Comunicaciones: Señales submarinas, radar y otros sistemas de mensajes.
- Química: Preparación de reacciones y floculación.
- Fotografía: Preparación de emulasiones.
- Medicina: Diagnósticos y exploraciones del cuerpo humano.
- Navegación y pesca: Navegación marina y sondas de profundidad.
- Industria química: Preparación de coloides, desagasificación de líquidos, medición de espesores en polímeros.
- Construcción: Inspección de concreto, inspección de barras de esfuerzo, fatiga de grúas con canasta.
- Metal- Mecánica: Control de calidad en productos fundidos, maquinados, forjados, laminados, uniones soldadas, inspección de recipientes a presión, sistemas de tuberías, intercambiadores de calor. tanques, reactores, estructuras, plataformas y barcos, medición de espesores.
- Automotriz: Inspección de puntos de soldaduras, medición de espesores y controlde calidad de materiales.
- Aeronáutica: Medición de espesores, control de calidad en materiales metálicos y compuestos.

La inspección ultrasónica de metales se utiliza para la detección de discontinuidades. Este método puede utilizarse para detectar fallas internas en la mayoría de los metales y aleaciones de ingeniería. Las discontinuidades a ser detectadas incluyen poros, huecos, inclusiones, laminaciones, sopladuras, grietas, etc. Se han desarrollado procedimientos de inspección y documentación relacionada. Ejemplos de estos documentos son el código ASME, código AWS, normas API, normas ASTM.[11]

2.2. Ventajas del uso del ultrasonido

- Gran velocidad de prueba: La operación electrónica proporciona indicaciones prácticamente instantáneas de la presencia de discontinuidades.
- Mayor exactitud: En comparación con los demás métodos no destructivos, la determinación de la posición de discontinuidades internas, estimando sus tamaños, orientaciones, forma y profundidad es más exacta.
- Alta sensibilidad: Permite la detección de discontinuidades a una gran profundidad.
- Buena resolución: Siendo esta característica la que determina que puedan detectarse los ecos procedentes de discontnuidades próximas a la superficie o que permita la discriminación de ecos provenientes de dos discontinuidades muy cercanas entre sí.

- Acceso: La técnica pulso - eco requiere el acceso por una sola cara del material.
- Permite La interpretación inmediata, la automatización y el control del proceso de fabricación.
- No utiliza radiaciones perjudiciales para el organismo humano y no tiene efectos sobre el material inspeccionado.
- Puede dejar registro permanente de las inspecciones realizadas y evaluaciones a través de computadora.

2.3. Desventajas del uso del ultrasonido

- La inspección manual requiere mucha atención y concentración de técnicos experimentados.
- Se requiere un gran conocimiento técnico para el desarrollo de los procedimientos de inspección.
- Las piezas de geometría compleja, rugosas, de grano, grueso, porosas, demasiado ásperas, muy pequeñas, muy delgadas o no homogéneas son difíciles de inspeccionar.
- Se necesita usar patrones de referencia, tanto para calibrar el equipo como para caracterizar las discontinuidades.
- El patrón de referencia debe ser el mismo material o acústicamente equivalente al material de la pieza que se va a inspeccionar.
- Alto costo de equipo y accesorios.
- Convencionalmente, la transmisión del ultrasonido a la pieza se realiza por medio de un acoplante líquido o pastoso. La transmisión del ultrasonido a la pieza también puede realizarse a través del aire por medio de láser, presión o generando el ultrasonido con un transductor acústico electromecánico.

2.4. Instrumento ultrasónico

Existe una gran variedad de equipos ultrasónicos de diferentes marcas, modelos, tamaños, forma, presentación de resultados, etc. Un ejemplo es el mostrado en la figura 2. La selección del equipo deberá ser hecha de acuerdo con las necesidades de la inspección y el sistema de transmisión apropiado. Sin embargo, el sistema de transmisión pulso - eco es el más utilizado en la actualidad.[11]

Para llevar a acabo la transmisión de ultrasonido existen 4 sistemas básicos:

- Pulso - eco
- Transmisión a través
- Resonancia
- Dual

Figura 2: Instrumento de ultrasonido industrial.[13]

Enfocándose en el sistema de pulso - eco, como el mostrado en la figura 3 consiste en hacer incidir impulsos cortos en vibraciones ultrasónicas sobre un cuerpo, de tal forma que la energía reflejada en las discontinuidades o en la pared posterior proporcionen una buena base para poder valorar el tiempo transcurrido en ida y vuelta del impulso, y permita así determinar la distancia a la cual se encuentran las discontinuidades desde la superficie o pared posterior.[11]

La técnica pulso - eco para el ensayo ultrasónico utiliza 2 parámetros simultáneamente:

- La amplitud de las señales obtenidas de discontinuidades internas.
- El tiempo requerido por el haz para viajar entre superficies específicas y las discontinuidades.

2.4.1. Componentes del equipo ultrásonico

La mayoría de los sistemas de inspección ultrasónica están constituidos por el siguiente equipo básico:

- Generador de los impulsos eléctricos que llegan al transductor.
- Un transductor que recibe los pulsos eléctricos y los convierte en ondas mecánicas.
- Un transductor que recibe las ondas mecánicas y las convierte a pulsos eléctricos. (Puede ser el mismo transductor que se usa para convertir los pulsos eléctricos a mecánicos.)
- Un dispositivo para amplificar y si es necesario, modificar las señales del transductor.
- Un dispositivo de despliegue para indicar las características de salida de la pieza de prueba, el dispositivo puede ser un tubo de rayos catódicos (TRC) o pantalla de cristal líquido LCD.
- Un reloj electrónico o contador (timer) para controlar la operación de los componentes del sistema completo, el cual sirve como punto de referencia primario y proporciona la coordinación del sistema completo.

Figura 3: Instrumento de ultrasonido mostrando la técnica de pulso-eco.[14]

2.5. Controles electrónicos

Casi todos los equipos usados en los ensayos por ultrasonido, aún los más sofisticados, tienen circuitos electrónicos similares y proporcionan funciones básicas comunes. Cada equipo consta de las siguientes funciones básicas esenciales:

- Tipos de forma de onda: Las señales de ultrasonido son representadas en la línea base horizontal en función de su amplitud, creando una forma de onda. Esta forma de onda se conoce como onda de radiofrecuencia y posee toda la información de la señal (picos positivos y negativos). Los equipos de ultrasonido proporcionan la posibilidad de rectificar la onda de radiofrecuencia para obtener la forma de onda que mejor se ajuste a una inspección en particular. La onda completa (FW) muestra los picos positivos y negativos, pero los picos negativos son mostrados como positivos. La media onda positiva (HWP) muestra sólo los picos positivos. La media onda negativa (HWN) muestra sólo los picos negativos.

- Fuente de corriente: Los circuitos encargados de alimentar la corriente para el funcionamiento forman la fuente de corriente eléctrica. Algunos equipos son alimentados directamente de la línea de corriente mientras que otros son aliemntados a través de baterías.

- Desplazamiento del impulso: También conocido como desplazamiento del punto 0, el ajuste de este control desliza el impulso (ecos recibidos) sobre la línea horizontal de la gráfica.

- Conmutador del servicio dual: Este conmutador conecta o desconecta las entradas para el servicio como emisor-receptor o emisor y receptor.

- Amplificador del receptor: Esta unidad tiene la función de transformar las señales recibidas y, por medio de dispositivos periféricos, ellas puedan ser ajustadas a una forma conveniente para su interpretación.

- Velocidad: El control de velocidad permite al inspector proporcionar al equipo la velocidad de propagación de ondas ultrasónicas del material bajo inspección.

- Rango: La pantalla del equipo está provista de 50 divisiones, el control del rango permite al operador proporcionar una valor específico a cada división.
- Zero offset: El control Zero offset es un control de retardo fino usado para compensar el desgaste de la cara del transductor o el tiempo requerido para el viaje del sonido en una línea de retardo o en una zapata para haz angular. Esencialmente permite al inspector fijar un tiempo cero para el cálculo electrónico de distancia en el punto exacto donde el pulso de sonido entra a la pieza a inspeccionar.
- Frecuencia: El control de frecuencia permite al inspector seleccionar la frecuencia que corresponde al valor del transductor.
- Compuertas: Una compuerta es un dispositivo electrónico que permite al inspector monitorear discontinuidades en zonas específicas de la pieza. La compuerta puede ser ajustada para que se emita una alarma visual o sonora cuando una señal la cruce. Para extraer información de una señal (eco) desplegada en la pantalla es necesario posicionar la compuerta sobre esta señal. Los controles de la compuerta permiten modificar su posición en la pantalla.
- Longitud del pulso (Damping): El control de la longitud del pulso o damping es usado para ajustar la duración del pico de voltaje más alto aplicado al transductor. Los valores más altos de damping proporcionan mayor resolución en la superficie de entrada. Los pulsos largos proporcionan mayor penetración.
- Voltaje de pulso: Este control determina la amplitud del pulso inicial generado. Algunos instrumentos proporcionan rangos de 40 a 400 volts en incrementos de 5V. Otros instrumentos proporcionan ajustes de sólo bajo, medio y alto voltaje.
- Ancho de pulso: Algunos instrumentos generan pulsos cuadrados y pulsos en forma de picos. El control del ancho del pulso cuadrado y es usualmente en nanosegundos. El efecto del ancho del pulso es similar al efecto de la longitud del pulso.
- Ganancia: El control de ganancia es usado para ajustar la altura(amplitud) de la señal en la pantalla. Un incremento positivo en el control de ganancia incrementará la amplitud de la señal. Algunos instrumentos cuentan con atenuadores, un incremento positivo de atenuación proporcionará un decremento en la amplitud de la señal.
- Frecuencia de repetición del pulso (FRP): La frecuencia del pulso es el número de pulsos disparados por segundo y es controlado por un reloj electrónico. Las frecuencias típicas son de 300 a 2000 pulso por segundo. Una frecuencia de repetición de pulsos mayor permitirá una mayor velocidad de escaneo manteniendo la sensibilidad requerida.

2.6. Presentación de señales

La presentación de señales depende del equipo ultrasónico utilizado. Entendiéndose por presentación de señales la forma como las señales recibidas son procesadas y mostradas en la pantalla para tener un medio de evaluación. Existen 3 tipos de presentación estandarizadas:

- A-scan
- B-scan
- C-scan

Enfocándose en A-scan, mostrada en la figura 4, es una representación muy usada en sistemas ultrasónicos de prueba. La línea base horizontal en el osciloscopio indica el tiempo transcurrido o distancia recorrida por el haz ultrasónico (de izquierda a derecha) y la deflexión vertical muestra la amplitud de la señal (ECO).

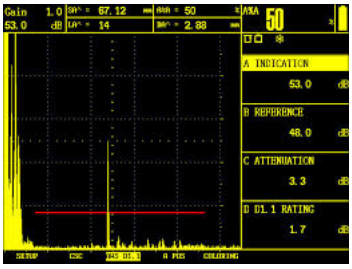

Figura 4: Interfaz gráfica A-Scan para pruebas de ultrasonido.[15]

Cuando se conocen las dimensiones del objeto de prueba, el ultrasonido puede utilizarse para medir las velocidades ultrasónicas y calcular el módulo elástico de un material. La amplitud de la señal representa la intensidad de la energía ultrasónica recibida por el equipo. Esto puede relacionarse con el tamaño de discontinuidades, atenuación en el objeto de prueba, dispersión del haz y otros factores. En ensayos industriales, la sensibilidad de instrumentos A-scan típicamente es adecuada para detectar discontinuidades pequeñas (incluyendo fisuras, inclusiones y porosidad), no obstante que pueden existir otras limitaciones como la resolución.[11]

2.7. Raspberry Pi

Raspberry Pi es un computador de placa reducida, como el que se muestra en la figura 5, computador de placa única o computador de placa simple (SBC) de bajo costo desarrollado en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas.[16]

Sacada a la luz en el año 2016, renueva procesador, una vez más de la compañía Broadcom, una vez más un Quad-Core, pero pasa de 900MHz a 1.20GHz. Mantiene la RAM en 1GB. Su mayor novedad fué la inclusión de Wi-Fi y Bluetooth (4.1 Low Energy) sin necesidad de adaptadores. El Raspberry Pi usa mayoritariamente sistemas operativos GNU/Linux. Raspbian, una distribución derivada de Debian que

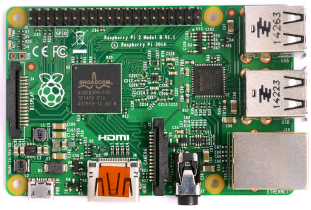

Figura 5: Raspberry Pi.[16]

está optimizada para el hardware de Raspberry Pi, se lanzó durante julio de 2012 y es la distribución recomendada por la fundación para iniciarse.[16]

A la GPU se accede mediante una imagen del firmware de código cerrado (un blob binario, en inglés), que se carga dentro de la GPU al arrancar desde la tarjeta SD. El archivo está asociado a los controladores del núcleo Linux que también son de código cerrado. Las aplicaciones hacen llamadas a las bibliotecas de tiempo de ejecución que son de código abierto, y las mismas hacen llamadas a unos controladores de código abierto en el núcleo Linux. La API del controlador del núcleo es específica para estas bibliotecas. Las aplicaciones que usan vídeo hacen uso de OpenMAX, las aplicaciones tridimensionales usan OpenGL ES y las aplicaciones 2D usan OpenVG; OpenGL ES y OpenVG hacen uso de EGL y este último, del controlador de código abierto del núcleo.[16]

SoC	Broadcom BCM2837
CPU	4x ARM Cortex-A53, 1.2GHz gráfica
GPU	Broadcom VideoCore IV
RAM	1GB LPDDR2 (900 MHz)
Networking	10/100 Ethernet, 2.4GHz 802.11n inalámbrico
Bluetooth	Bluetooth 4.1 Classic, Bluetooth Low Energy
Almacenamiento	microSD
GPIO	40 pines de propósito general
Puertos	4x HDMI, 3.5mm jack de audio-video 3.5mm, 4x USB 2.0, Ethernet, Interfaz de cámara serial, Intefraz de display serial (DSI)

3. Desarrollo

Las funciones principales que se requieren lograr en el desarrollo de este proyecto son :

- Representar gráficamente los valores contenidos de un arreglo.
- Mostrar opciones de configuración, a gusto del desarrollador.
- Permitir una comunicación con los disitintos periféricos de la Raspberry Pi, para este caso, serial

3.1. Ambiente de operación

- Se empleó la plataforma Raspberry Pi 3, la cual cuenta con una pantalla LCD de 7". Además cuenta con una variedad de puertos de comunicación que permiten la escalabilidad en comunicación de proyectos.
- El ambiente de desarrollo es totalmente implementado en Raspbian; sistema operativo basado en Linux, rama Debian[47], especialmente optimizado para ser ejecutado en microcontroladores basados en chips Broadcomm, que es el CPU de las Raspberry Pi.
- Se implementó en el lenguaje de programación Python 2.7[10], el cual es muy sencillo de utilizar; su documentación es de fácil acceso y existen varios ejemplos que sirven de referencia en la red.
- Se emplea la comunicación serial para recibir los datos de un dispositivo externo. Como ya se mencionó, la Raspberry Pi cuenta con varios periféricos, pero para motivos de simplicidad se empleó este tipo de comunicación.

3.2. Documentación del usuario

Por el momento sólo se tiene en consideración este manual para el desarrollo de proyectos similares; si se desea apoyo en la red, se pueden consultar los elementos en Referencias.

3.3. Requerimentos de interfaz externa

A continuación se describe que es lo que incluirá el proyecto de interfaz realizado, en lo que respecta a como interactúa con el usuario, la forma en que los elementos de hardware que se tienen previstos influirán en la Raspberry Pi, y como se comunicará esta misma por el puerto serial.

3.3.1. Interfaces de usuario

Como se puede apreciar en la figura 6, la interacción general se lleva a cabo por medio de interfaces gráficas empleando una pantalla táctil para seleccionar los comandos. Se emplean *Button's* rectangulares[36], que son los que vienen por defecto en la librería Tkinter[17]. A su vez se hace referencia a CIDESI empleando una imagen PNG, con transparencia.

Figura 6: Interfaz Gráfica Preeliminar

Se emplea una pantalla LCD de 7”de capacidades táctiles, como la que se muestra en la figura. Esta tiene una resolución 800*480 pixeles, y tiene un cable Ribbon Flex especial para conectarse a la Raspberry Pi, que posee un puerto dedicado.

Figura 7: Pantalla LCD 7”

Asimismo la Raspberry Pi cuenta con pines de entrada y salida de propósito general (GPIO), mismos

que proveen una variedad de periféricos de comunicación; entre ellos, el de comunicación serial. En la figura 8 se aprecian los pines correspondientes a la comunicación serial son GPIO14 y GPIO15, los cuales fueron los empleados para el presente proyecto.

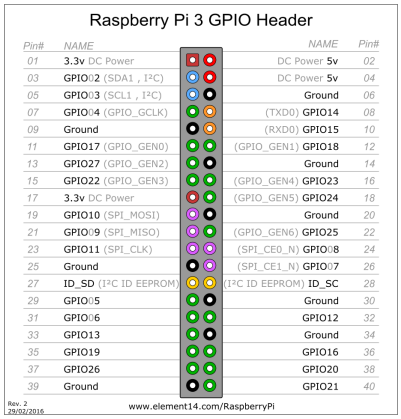

Figura 8: Pines de la Raspberry Pi.[18]

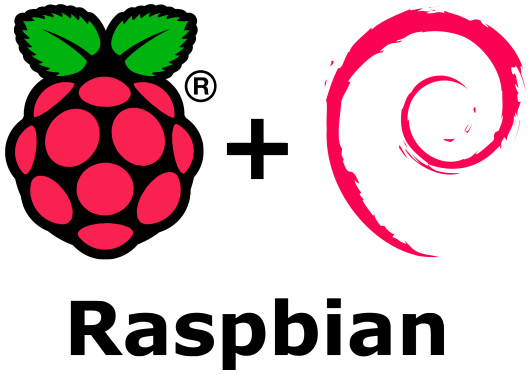

Figura 9: Logotipo de sistema Operativo Raspberry Pi.[19]

Figura 10: Logotipo de lenguaje de programación Python.[20]

3.3.2. Interfaces de comunicación

Se emplea la comunicación serial. Como se había mencionado, para fines de pruebas se empleó una MSP430G2553 Launchpad, que contiene la señal cargada en un vector y este último es enviado con un protocol de comunicación UART.[45]

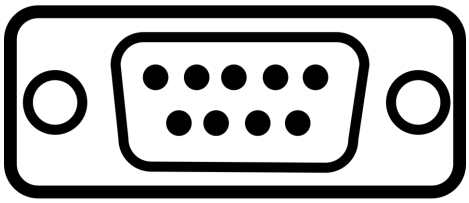

Figura 11: Comunicación Serial.[21]

4. Configuración del entorno de desarrollo

Para desarrollar aplicaciones en la Raspberry Pi, existe la posibilidad de emplear una PC normal. Esto es para comodidad del desarrollador, pues si bien la Raspberry Pi tiene entradas HDMI y USB para conectárseles una Pantalla y un teclado respectivamente; esto implica más espacio en el entorno de trabajo físico. Por lo tanto el siguiente capítulo sugiere los pasos para configurar el entorno de programación para comunicarse con una computadora corriendo Windows.

4.1. Comunicación entre Raspberry Pi y computadora a través del puerto Ethernet

4.1.1. Establecer configuraciones de conexión por Ethernet en la Raspberry Pi

Es importante asignar una dirección IP estática a la Raspberry Pi en la red local que se creará; que incluirá a la Raspberry Pi y la PC. Para ello se debe proceder de la siguiente manera:

- En la Raspberry Pi en la que se vaya a trabajar. Se accede a una terminal virtual y se ingresa el comando sudo nano /etc/dhcpcd.conf
- Se agrega al final del documento lo siguiente:

  interface eth0
  static ip_address=192.168.137.12/24
  static routers=192.168.137.1
  static domain_name_servers=192.168.137.1

Para este ejemplo se empleó la dirección .137.12; sin embargo, se puede emplear cualquier dirección que esté dentro del rango de la máscara de subred que se establece en el siguiente paso.

(Nota: Este paso requiere forzosamente que la Raspberry Pi se encuentre conectada a una pantalla y a un teclado; pero solo se debe realizar una vez.)

4.1.2. Configurar la computadora desde donde se vaya a trabajar en la Raspberry Pi.

En la computadora desde donde se trabajará en la Raspberry Pi, se tienen que establecer ciertas configuraciones que permiten el acceso desde una Terminal SSH o por escritorio remoto.

- Se conecta la Raspberry Pi a la computadora mediante el cable Ethernet.
- Se ingresa al comando ejecutar (tecla Windows + R) y se ingresa el comando ncpa.cpl

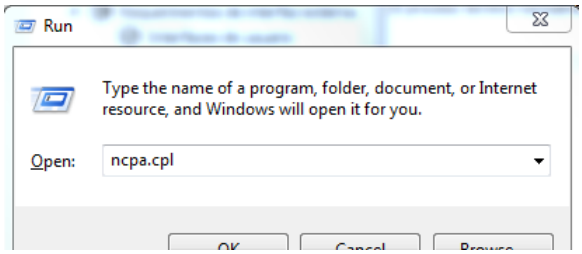

Figura 13: Ventana ejecutar

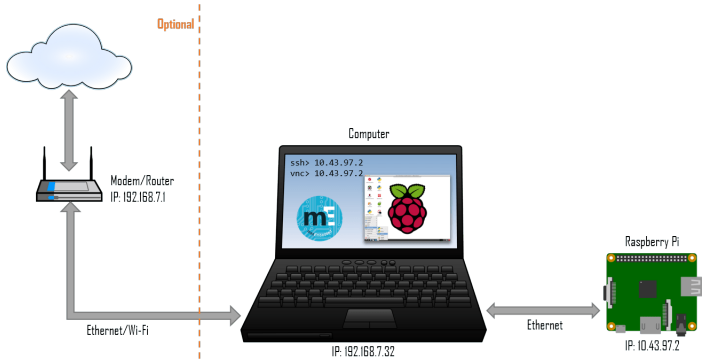

Figura 12: Conexión entre Raspberry Pi y PC a través de Ethernet

- Deberá aparecer una conexión de red extra; se le da click derecho y Propiedades. Una vez dentro de la ventana de diálogo, se selecciona la opción correspondiente a IPv4.

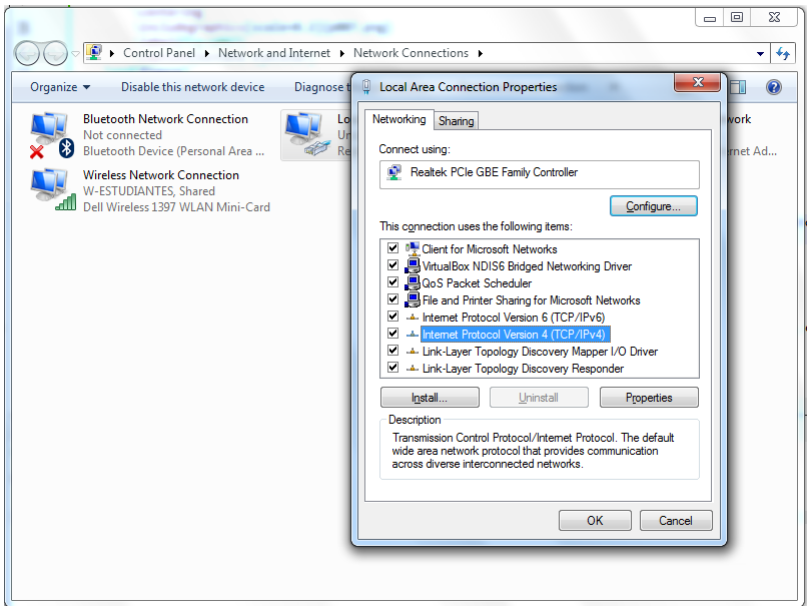

Figura 14: Propiedades de conexión

- Aplicar doble click en la mencionada opción y llenar los campos de la siguiente manera.

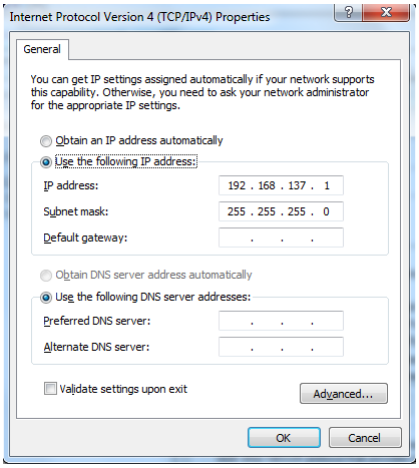

Figura 15: Configuración de red

Con esto bastará para comunicarse con la Raspberry Pi desde Windows.

4.1.3. Probar la conexión

Descargar la aplicación Putty para Windows e instalarla. Seleccionar los parámetros como en la imagen 16 e ingresar.

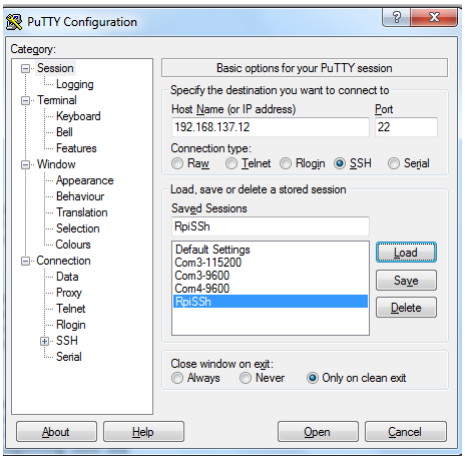

Figura 16: Configuración de red

4.1.4. Opcional: Configurar escritorio remoto

Es posible visualizar el escritorio de la Raspberry Pi en la PC, esto facilita el desarrollo de las aplicaciones simulando el entorno nativo en el que se va a implementar. Para ello:

- Se debe ingresar a una terminal en Raspberry Pi; ya sea dentro de la misma Raspberry o por SSH a través de Puttty, e ingresar el sig. código: sudo apt-get install xrdp.
- Posteriormente; en Windows, se selecciona del menú Inicio, submenú Accesorios, la opción Conexión a Escritorio Remoto.
- Se ingresan las características del dispositivo al que se quiere conectar, en este caso, la Raspberry Pi.

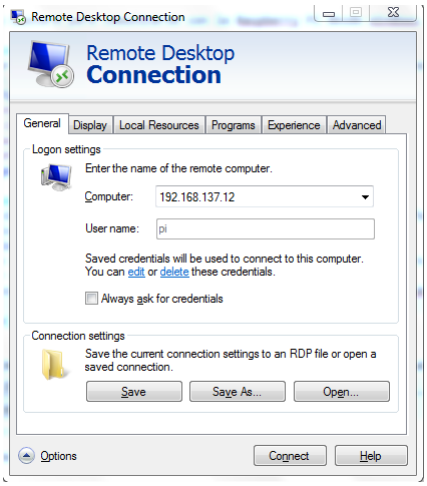

Figura 17: Configuración de escritorio remoto.

- De tener éxito, se debe visualizar el escritorio nativo de Raspbian en una ventana maximizada.

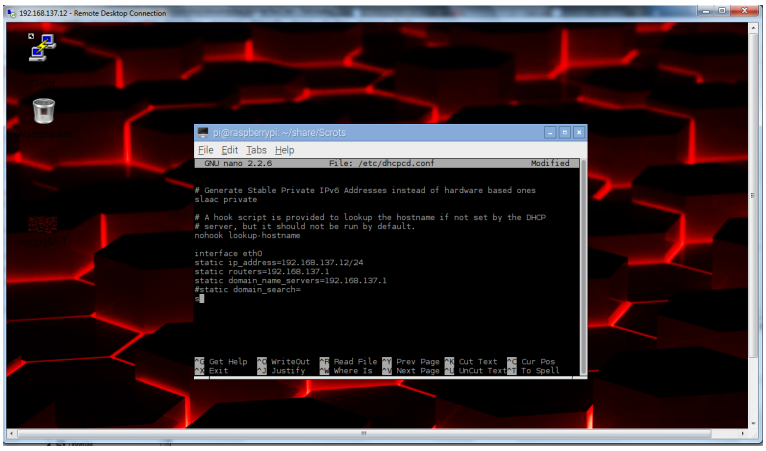

Figura 18: Escritorio Remoto correctamente conectado

4.2. Habilitar conexión a Internet compartida desde Wi-Fi a través de Ethernet

Es bien sabido que los entornos basados en Linux son altamente dependientes del internet; por lo tanto, es recomendable que si se va a seguir este método de trabajo, se comparta la conexión de la siguiente forma:

- Se vuelve al menú de conexiones de red accedido en el segundo paso de la sección anterior, y se selecciona la red que está recibiendo el internet activamente.

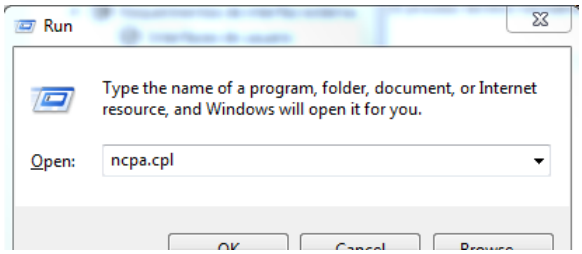

Figura 19: Ventana comando ejecutar

- Se selecciona la pestaña compartir y se pestañean las opciones mostradas; en el submenú de la primera opción se pestañean todos los servicios para compartir a través de la red.

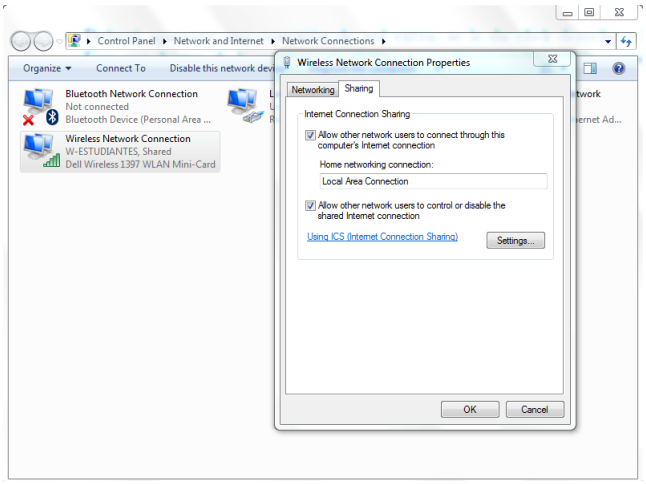

Figura 20: Red activa conectada al internet

Con esto deberá ser suficiente para acceder a internet desde la Raspberry Pi.

4.3. Instalación de librerías

Es primordial saber cómo instalar las librerías que serán requeridas. La instalación de Python incluida en Raspbian incluye algunas librerías por defecto. Sin embargo, en dado caso que llegarán a faltar algunas; el interpretador de Python mostrará un error y desconocerá las declaraciones de estas en el código. En este caso se deberá instalar la librería adecuada.

4.3.1. En Windows:

Si se desea emplear el desarrollo en Windows es necesario:

- Abrir una ventana del símbolo del sistema.
- Escribir el comando pip install (nombre de la librería)
- Es probable que para algunas librería, se deban desccargar herramientas de compilación, para ello de internet se deberán descargar Python Development Tools del siguiente hipervínculo: `https://www.visualstudio.com/vs/python/`.[43]
- Es sabido que los paquetes correspondientes a Matplotlib y a numpy no son del todo compilables en instancias de Windows de 64 bits. Es por ello que se recomienda descargar paquetes precompilados del siguiente repositorio: `http://www.lfd.uci.edu/~gohlke/pythonlibs/`.[42] Python emplea un archivo especial para guardar las librerías llamados wheel.[46] Una vez descargado este archivo, se abre una instancia del símbolo del sistema, se dirige uno al directorio donde resida el archivo y finalmente se ejecuta el comando pip install (nombre del archivo). Si todo sale bien, aparacerá un mensaje indicando la instalación del módulo exitosa.

4.3.2. En Raspbian:

- Se abre una instancia de la terminal en Raspbian, se ejecuta el comando: sudo pip install (nombre de la librería).
- Alternativamente; habrá algunas librerías que puedan instalarse directamente con el administrador de paquetes de Raspbian, mediante el comando: sudo apt-get install (nombre de la librería).

5. Programación de la interfaz

La interfaz está compuesta de elementos cuya jerarquía es descendente:

1. Librerías
2. Programa Raíz.
3. Ventana Principal
4. Subventanas
5. Widgets

5.1. Librerías

5.1.1. Descripción y prioridad

Python ya tiene definido en sí mismo ciertas funciones que facilitan al usuario el desarrollo de programas. Sin embargo hay algunos elementos externos que no se encuentan dentro de Python y por lo tanto deben ser instalados en el entorno de desarrollo.

Código 1: Librerías

```
from Tkinter import *#Librería gráfica
import Tkinter as Tk
from PIL import ImageTk, Image#Importar libreria de imágenes

import numpy as np
import matplotlib
matplotlib.use('TkAgg')
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.pyplot as plt
import time
import sys
import sm
from sm import lec as rd
```

En en el código 1 se muestran las librerías empleadas en este ejemplo; los cuales de describen a continuación:

5.1.2. TkInter

La librería Tkinter [17] es el paquete incluido por defecto en Python para realizar aplicaciones con interfaces gráficas. Tkinter consiste en una serie de módulos; siendo el más importante Tkinter mismo.
El módulo Tkinter se encarga de importar las clases de los widgets y constantes asociadas. La forma de incluirlo en el programa es:

- import Tkinter
- from Tkinter import *

Como se puede apreciar en 1; en el programa está incluido de 2 formas. Primero se manda a llamar la librería completa, cargando todos sus elementos asociados; en seguida se llama de nuevo pero ahora asignándole el nombre de Tk, dándole una identidad a todos los elementos asociados. En la siguiente sección se profundizará un poco más sobre esto.

5.1.3. Python Image Library

Python Image Library es una librería que agrega capacidades de procesamiento de imagen al interpretador de Python. Ofrece un soporte de formatos extenso, una representación externa eficiente, y capacidades de procesamiento de imagen muy poderosas.
La librería principal está diseñada para rápido acceso a datos almacenados en algunos formatos de pixeles básicos. Algunos usos posibles de esta libería:

- Almacenamiento de imágenes.
- Representación de imágenes.
- Procesamiento de imágenes.

Como se muestra en el código 1; de la librería se incluyen sólo los elementos ImageTk e Image[39], pues son los únicos que se necesitan para leer las imágenes de los logotipos.

5.1.4. numpy

numpy es un paquete fundamental para computación científica implementada en Python. Contiene entre otras cosas:

- Un poderoso manipulador de arreglos de N-dimensiones
- Herramientas de integración con código de C, C++ y Fortran
- Capacidades útiles de álgebra lineal, transformada de Fourier y números aleatorios

En el código 1 se incluye simplemente para cargar sus funciones.

5.1.5. Matplotlib

Es una librería de graficado en 2D para Python que produce figuras de calidad de publicación, en una variedad de formatos y ambientes multiplataforma. Matplotlib tiene una base de código extensa que puede ser abrumadora para principiantes, sin embargo, puede ser entendido con un marco conceptual básico y conocimiento de algunos puntos importantes. El graficado requiere configuración en un rango de niveles; desde lo más general a lo más específico. El propósito de esta librería es asistir en visualizar los datos fácilmente.

Todo en Matplotlib está organizado en una jerarquía. En la parte más alta de esta se encuentra el módulo matplotlib.pyplot. En este nivel, funciones simples son usadas para agregar elementos. En el siguiente nivel, se encuentra la interfaz orientada a objetos, en la cual pyplot es usada para algunas funciones de creación de figuras, y el usuario explícitamente crea y sigue elementos que comprenden la figura a graficar.

Para representar gráficamente sus elementos; Matplotlib se comunica con elementos de más bajo nivel que determinan y compatibilizan el entorno gráfico en el que se está trabajando. Para controlar este parámetro se emplea el comando use(), colocando dentro del paréntesis la palabra clave del entorno gráfico correspondiente.

En el código 1, se incluye de una forma distinta a las vistas anteriormente.

- Primero se importa matplotlib para incluir todos sus elementos.
- Se especifica que se trabajará con la librería gráfica Tkinter, por lo tanto se parametriza con TkAgg.
- También es necesario especificar el canvas, que es el elemento en donde se dibujan las figuras. Para ello, de la librería matplotlib, de la clase backends, de la subclase conrrespondiente a TkInter, se parametriza para trabajar con este último, con"FigureCanvasTkAgg".
- Por último, se especifica el objeto que servirá para crea los elementos de matplotlib, en este caso se emplea "plt".

5.1.6. sys

sys provee acceso a algunas variables usadas o mantenidas por el interpretador de Python y funciones que interactúan fuertemente con él. Se emplea para terminar la interacción entre usuario y programa.

5.1.7. sm

Finalmente se utiliza la propiedad de importaciones de Python para cargar módulos externos. Estos módulos son scripts de código externos al programa principal que pueden ser integrados; de forma que su funcionalidad pueda ser incluida sin ofuscar el contenido del código, pues ambos módulos son editables en archivos aparte.

Este módulo será explicado en la sección "Módulo de comunicación serial".

5.2. Variables globales

Son variables que son creadas desde el inicio del programa y están disponibles para todos los módulos que lo componen.

5.2.1. Descripción y prioridad

Las variables bandera no requieren mucha explicación, sólo se emplearon para determinar cambios en las ventanas y determinar si estaban abiertas o no; sin embargo, no se eliminaron debido a que pueden ayudar a futuros desarrolladores que sigan este manual a apreciar el comportamiento de estas variables a lo largo del programa.

Por otro lado, las variables windowHeight y windowWidth, que a su vez definen la variable String universal Geometry, son primordiales para el programa pues definen las dimensiones que tendrán las ventanas de la aplicación. En este caso los valores están fijados a 480 y 800, respectivamente, pues es la resolución de la pantalla de 7 pulgadas a la que está conectada la Raspberry Pi. Sin embargo, puede ser modificada por el desarrollador de modo que satisfaga las necesidades de la plataforma en que esté trabajando.

Código 2: Variables Globales

```
bA=0#Bandera subventana
bOp1=0#Bandera subventana opción 1
bOp2=0#Bandera subventana opción 2
bOp3=0#Bandera subventana opción 3
bOp4=0#Bandera subventana opción 1
```

```
bOp5=0#Bandera subventana opción 1
bOp6=0#Bandera subventana opción 1
bOp7=0#Bandera subventana opción 1
bOp8=0#Bandera subventana opción 1
bOp9=0#Bandera subventana opción 9
bOp10=0#Bandera subventana opción 10
##buttonWidth=10#Altura del boton
windowHeight=480#Altura ventana
windowWidth=800#Anchura Ventana
universalGeometry ='%dx%d' %(windowWidth,windowHeight)#Delimitador ventana
```

5.3. Programa raíz

5.3.1. Descripción y prioridad

El principal elemento del programa, pues es aquí donde se indica el ciclo del mismo. Aquí se define el tipo del objeto raíz, la ventana principal. De no existir este, el programa no se ejecutaría.

Código 3: Programa Raíz

```
#-----------------------------------------------------------------------
if __name__ == "__main__":
    root = Tk.Tk()#Instanciaicón de las librerías gráficas
    root.geometry(universalGeometry)#Delimtador de ventana
    app = MyApp(root)#Instanciación de la ventana principal
    root.mainloop()#Ciclo infinito
```

En el código 3 se muestra un ejemplo de este. A continuación se describen los elementos que lo componen:

- La primer parte del código; el condicional, se asegura de ejecutar esta parte del código si, y sólo si, este es el primer módulo que se ordena ejecutar desde el inicio del programa. Esto quiere decir, que de haber sido incluido como un módulo externo, como si fuera una librería, este segmento no se ejecutaría.
- Como ya se definió; Tkinter es la librería encargada de proveer las propiedades gráficas a Python. Para inicializar Tkinter, se debe crear un widget Tk raíz. Sólo se debe crear un widget raíz para cada programa, y debe ser creado **antes** de crear otros Widgets. Esto se logra con root = Tk.Tk()
- Se especifica la geometría que deberá tener la ventana, haciendo referencia a la variable global declarada al principio del programa. Si bien el objeto root no se desplegará como tal en pantalla, es necesario hacer esto para pasar sus atributos a la subventana inmediata.
- Se crea una instancia de la clase MyApp, la cual tiene como padre al Widget raíz, este puede servir para otorgar algunos de sus parámetros, como las dimensiones de la ventana.
- Finalmente se le especifica al objeto raíz que se ejecute cíclicamente hasta el infinito.

5.4. Clase MyApp

5.4.1. Descripción y prioridad

Esta clase incluye la ventana principal; la que da la bienvenida al usuario en cuanto inicia la aplicación. En el código 4 se muestra su estructura.

Código 4: Clase MyApp

```
class MyApp(Frame):

    #-----------------------------------------------------------------------
    def __init__(self, parent):
        """Constructor"""
        Frame.__init__(self, parent, height=windowHeight, width=windowWidth)# Inicializar
                # con parámetros por defecto
        self.root = parent
        self.root.title("Bienvenido")
##        self.root.config(="black")
##        self.frame = Tk.Frame(parent)
##        self.frame.pack()
##        self.attributes("-toolwindow", 1)
        self.pack()
        self.create_widgets()

    def create_widgets(self):#Contenido de la ventana
        """Botones con características específicas"""
        strBtn = Button(self, text="A-scan", command=self.openFrame,height=3,width=30)
        stBtn = Button(self, text="Configuraciones", command=self.openFrame2,height=3,width=30)
        abBtn = Button(self, text="Acerca",command=self.openFrame3,height=3,width=30)
        exBtn = Button(self, text="Salir",command=self.Salir,height=3,width=30)
        strBtn.place(relx=0.01, rely=0.03)
        stBtn.place(relx=0.01, rely=0.29)
        abBtn.place(relx=0.01, rely=0.58)
        exBtn.place(relx=0.01, rely=0.85)

        #Imagen
        im=Image.open("cidesi.png")
        size = 375,271
        im.thumbnail(size)
        self.img = ImageTk.PhotoImage(im)
        panel = Label(self, image = self.img)
        panel.place(relx=0.45,rely=0.3)

    #-----------------------------------------------------------------------
    def hide(self):#Función prara occultar ventana
        """"""
        self.root.withdraw()

    #-----------------------------------------------------------------------
    def openFrame(self):#Función para abrir ventana de A-scan
        """"""
        self.hide()
        subFrame = scanFrame(self)

    #-----------------------------------------------------------------------
    def openFrame2(self):#Función para abrir ventana de configuración
        """"""
        self.hide()
        subFrame = confFrame(self)
    #-----------------------------------------------------------------------
    def openFrame3(self):#Función para abrir ventana de información
        """"""
##        state=
        global bA
        state = bA
        if state==0:
            subFrame=aboutFrame(self)
```

```
        else:
            pass
#-----------------------------------------------------------------------
def show(self):#Función para mostrar la ventana
    """"""
    self.root.update()
    self.root.deiconify()
#-----------------------------------------------------------------------
def Salir(self):#Función para salir del programa
    self.root.destroy()
```

- Se empieza por nombrar la clase; en este caso MyApp. Entre paréntesis se determina el tipo de objeto que se requiere, Frame establece que va a ser una ventana.
- Se define la primera función, que es el constructor del objeto, **la ventana**. Hay que recordar que Python es un lenguaje de programación de declaración explícita, esto explica la presencia del parámetro self[40]; esto es una referencia a la instancia actual de la clase. self referencia al objeto mismo y parent al objeto padre, el cual fue definido en el Programa Raíz, como parámetro de la instancia App.
- Frame.__init__ es el método de inicialización, los argumentos que conlleva hacen; de nuevo, referencia a la instancia de la clase, el objeto padre, y en este caso la altura y anchura. Más adelante se explicará el porque de estos 2 últimos.
- self.root = parent es para hacer referencia al objeto del cuál se proviene en una variable; que casualmente comparten el mismo nombre en ambas instancias.
- Pack() sirve para desplegar la ventana.
- create widgets es una función personalizada que permite la creación de Widgets en la ventana principal. Aquí están definidos los distintos Button's y etiquetas a emplear. En este ejemplo; primero se definen los Button's, y posteriormente su ubicación en la ventana. Como se está empleando el administrador de geometría Place[31], es necesario haber definido en la creación del objeto la altura y anchura del objeto; ya que si bien estos datos son heredados del objeto padre, sólo sirven para determinar las dimensiones de la ventana mas no para la posición relativa de los objetos con respecto del origen.
- Para detalles ornamentales se escogió un logo de esta institución. Para manipularla es necesaria la librería Python Image Library. Se hace una instancia de la imagen al abrirla con Image.open(), se emplea una variable para establecer una tamaño específico y se reajusta su tamaño. Se hace una instancia de la imagen con referencia al objeto ventana actual y se coloca en un Widget de tipo Label (etiqueta) y se despliega.
- La función hide sirve para ocultar la ventana actual.[32]
- La función openFrame sirve para abrir un submenú que contiene la visualización de la gráfica; que se tiene planeada para la visualización de la señal A-Scan. Esta función oculta la ventana actual y desliega el mencionada menú.
- La función openFrame2 funciona para abrir un segundo submenú que contiene opciones de configuración. Primero oculta la ventana actual y despliega este submenú.

- La función openFrame3 despliega una subventana con una pequeña leyenda, asemejando a los submenús Acerca de en los programas, conteniendo los créditos del actual proyecto.
- La función show sirve para desocultar la ventana actual.
- La función Salir destruye la ventana y sale del entorno Python, efectivamente cerrando el programa.

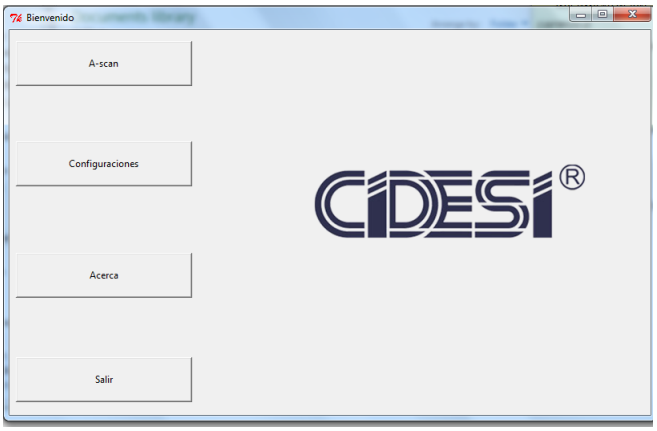

Figura 21: Vista Preeliminar Clase MyApp

5.5. Clase scanFrame

5.5.1. Descripción y prioridad

Esta clase define una Widget ventana que contiene la gráfica donde de se va a representar la señal recibida por la comunicación serial. A su vez muestra Button's para establecer ciertos parámetros de configuración que afecten al despliegue de la señal en la gráfica como filtros, etc.

Código 5: Clase scanFrame

```
#######################################################################
class scanFrame(Tk.Toplevel):#Clase ventana A-Scan
    """"""

    #-----------------------------------------------------------------
    def __init__(self, original):
        """Constructor"""
        self.original_frame = original
        Tk.Toplevel.__init__(self,height=windowHeight,width=windowWidth)# Inicializar
        self.geometry(universalGeometry)#Delimitador de ventana          con parámetros
        rango_filas = range(0,20)#Configuración filas
        for i in rango_filas:
            self.rowconfigure(i,minsize=24)
        rango_columnas = range(0,11)#COnfiguración columnas
        for j in rango_columnas:
            self.columnconfigure(j,minsize=50)
        self.title("A-scan")
        self.scanWidgets()

    def scanWidgets(self):#Contenido de la ventana
        """Botones"""
        btnEscan = Tk.Button(self, text="Iniciar Escaneo",command=self.grafico)
        btnOp02 = Tk.Button(self, text="Opcion 2",command=self.op2)
        btnOp03 = Tk.Button(self, text="Opcion 3",command=self.op3)
        btnOp04 = Tk.Button(self, text="Opcion 4",command=self.op4)
        btnOp05 = Tk.Button(self, text="Opcion 5",command=self.op5)
        btnOp06 = Tk.Button(self, text="Opcion 6",command=self.op6)
        btnOp07 = Tk.Button(self, text="Opcion 7",command=self.op7)
        btnOp08 = Tk.Button(self, text="Opcion 8",command=self.op8)
        btnOp09 = Tk.Button(self, text="Opcion 9",command=self.op9)
        btnOp10 = Tk.Button(self, text="Opcion 10",command=self.op10)
        btn = Tk.Button(self, text="Cerrar", command=self.onClose)


        """Ubicación de botones"""
        btnEscan.grid(row=14,column=1,sticky=N+S+E+W)
        btnOp02.grid(row=14,column=3,sticky=N+S+E+W)
        btnOp03.grid(row=14,column=5,sticky=N+S+E+W)
        btnOp04.grid(row=14,column=7,sticky=N+S+E+W)
        btnOp05.grid(row=14,column=9,sticky=N+S+E+W)
        btnOp06.grid(row=16,column=1,sticky=N+S+E+W)
        btnOp07.grid(row=16,column=3,sticky=N+S+E+W)
        btnOp08.grid(row=16,column=5,sticky=N+S+E+W)
        btnOp09.grid(row=16,column=7,sticky=N+S+E+W)
        btnOp10.grid(row=16,column=9,sticky=N+S+E+W)
        btn.grid(row=15,column=10,sticky=N+S+E+W)

        #Imagen
##        im=Image.open("cidesi.png")
##        size = 320,271
##        im.thumbnail(size)
##        self.img = ImageTk.PhotoImage(im)
##        panel = Label(self, image = self.img)
##        panel.grid(row=3,column=3,columnspan=4,rowspan=5)

        #Grafico
        self.fig = plt.figure(figsize=(8,4))
```

```
##          self.fig = plt.figure()
        self.fig, self.axes = plt.subplots(figsize=(8,4))
        self.styles = ['r-','g-','y-','m-','k-','c-']
        self.x = np.arange(0,1025)
        self.y=self.x
        self.Figure = FigureCanvasTkAgg(self.fig,master=self)
        self.Figure.get_tk_widget().grid(row=0,column=0,columnspan=11,rowspan=14)
    #-----------------------------------------------------------------------
    def plot(self,ax,style):
        """"""
        return ax.plot(self.x, self.y, style, animated=True)[0]
    #-----------------------------------------------------------------------
    def onClose(self):
        """"""
        self.destroy()
        self.original_frame.show()
    #-----------------------------------------------------------------------
    def grafico(self):
        self.axes.set_title("A-scan")
        self.axes.set_autoscaley_on(False)
        self.axes.set_ylim([-255,255])
        self.axes.set_xlim([0,1025])
        self.fig.canvas.draw()
        self.backgrounds = self.fig.canvas.copy_from_bbox(self.axes.bbox)
        self.lines = self.plot(self.axes,self.styles[0])
##          for i in xrange(1,1000):
##              self.fig.canvas.restore_region(self.backgrounds)
##              self.lines.set_ydata((255*np.sin(self.x + i/10.0))/2 + 128)
##              self.axes.draw_artist(self.lines)
##              self.fig.canvas.blit(self.axes.bbox)
        y= rd()
        self.fig.canvas.restore_region(self.backgrounds)
        self.lines.set_ydata(y)
        self.axes.draw_artist(self.lines)
        self.fig.canvas.blit(self.axes.bbox)
    #-----------------------------------------------------------------------
    def op1(self):#Función abrir opción 1
        """"""
        subFrame=op1Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op2(self):#Función abrir opción 2
        """"""
        subFrame=op2Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op3(self):#Función abrir opción 3
        """"""
        subFrame=op3Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op4(self):#Función abrir opción 4
        """"""
        subFrame=op4Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op5(self):#Función abrir opción 5
        """"""
        subFrame=op5Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op6(self):#Función abrir opción 6
        """"""
        subFrame=op6Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op7(self):#Función abrir opción 7
        """"""
        subFrame=op7Frame(self)
```

```
        self.wait_window(subFrame)
#-----------------------------------------------------------------------
def op8(self):#Función abrir opción 8
    """"""
    subFrame=op8Frame(self)
    self.wait_window(subFrame)
#-----------------------------------------------------------------------
def op9(self):#Función abrir opción 9
    """"""
    subFrame=op9Frame(self)
    self.wait_window(subFrame)
#-----------------------------------------------------------------------
def op10(self):#Función abrir opción 10
    """"""
    subFrame=op10Frame(self)
    self.wait_window(subFrame)
```

- Se inicia por definir la clase scanFrame, la cual va a ser un Widget de tipo Toplevel. Este tipo de Widgets se comportan de manera similar a los de tipo Frame, pero tienen la particularidad de desplegarse en una ventana separada y un nivel encima. Es usada para desplegar ventanas de aplicación extra, diálogos, y otras ventanas "pop-up". En la función de inicialización se cargan los argumentos self; que ya se explicó, y original, para hacer referencia al objeto del cual se proviene.
- Se hace referencia al objeto del cual se proviene en una variable, llamada original_frame
- Como en el Widget anterior, se define el constructor con __init__; cargándolo con los argumentos self, la anchura y la altura de la ventana.
- Como aquí no se heredan las características del padre, se define la geometría de la ventana con la variable global universalGeometry.
- Para esta ventana; se empleará el administrador de geometría Grid[30], es por ello que se debe dar una configuración distinta a esta ventana. Primero; la variable rango_filas sirve para determinar el número de filas en donde se acomodarán los Widgets. En seguida; con ayuda de un ciclo for se da formato a las filas, indicando que se requiere una separación mínima de 24 pixeles entre cada fila. Posteriormente, se da formato a las columnas de igual manera con ayuda de un ciclo for, indicando que se require un espacio mínimo de 50 pixeles entre cada columna(columnconfigure)(rowconfigure).
- Se establece el título de la ventana y se crean los widgets.
- Esta es la ventana con la configuración de Widgets más compleja. Primero se crean los Button's, se definen sus parámetros(command) y se colocan con la función Grid. Se configura el gráfico, este elemento depende la librería Matplotlib.

  Se crea una figura(figure) con un tamaño específico. Los números definen la anchura y la altura respectivamente (este tamaño es obtenido a prueba y error, es recomendable probar al que se ajuste a la aplicación del desarrollador).

  Se definen la cantidad de gráficos que tendrá esta figura dentro de sí. Para este ejemplo se empleará solo 1. Se vuelve a definir el tamaño(axes).

  Se inicializa un vector que contenga los strings que definen los diferentes estilos(styles) disponibles dentro de Matplotlib. Se incluyen 6 porque se sigue una implementación anterior, el desarrollador puede emplear cualquiera de los 6 que más le agrade.

Se inicializa un vector x y un vector y; que son los que contendrán los valores iniciales de la gráfica.

Se especifica que el Canvas en donde se va a dibujar la figura estará dentro de un objeto perteneciente a FigureCanvasTkAgg.

Se coloca la figura.

- La función plot es una función auxiliar de la siguiente función, la cual se carga con parámetros que definen a x y a y en la gráfica, así como el estilo que ocuparán. A su vez, retorna una gráfica representando estos parámetros.

- La función onClose permite cerrar la ventana actual y mostrar la anterior; que es de donde se proviene.

- La función gráfico es para comenzar a desplegar animadamente la señal que se quiere representar.

  Se define el título de la gráfica.

  Se establece que el eje Y va a tener un valor fijo desde el inicio; es decir, no presenta escalabilidad($set_autoescaleony$).

  Se establecen los límites del eje x(set_xlim).

  Se establecen los límites del eje y(set_ylim).

  Se dibuja el canvas, canvas.draw.

  Se guarda el fondo, para permitir una actualización en tiempo real de la gráfica y este sea renovado cada que se actualice la gráfica(backgrounds).[26]

  Se establece que lines va a ser lo ordenado por la función plot, previamente definida(lines).

  Mediante un ciclo for:

    Se restablece el fondo del canvas con el que está guardado en backgrounds, canvas.restore.

    Se actualizan los datos correspondientes a y(set_ydata).

    Se dibujan las líneas(axes.$draw_artist$., canvas.blit).

- Las siguientes funciones no hacen más que abrir ventanas de diálogo que se tienen planeadas para establecer configuraciones. Están configuradas de tal manera que no se tenga acceso a ninguna otra ventana hasta que la ventana de diálogo haya concluido.[41] Primero se crea la subventana y se pone al widget actual en tiempo de espera hasta que haya concluido la interacción mediante la función($wait_window$).

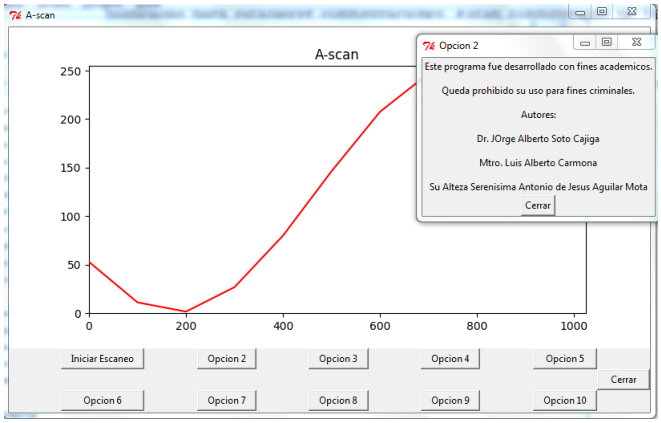

Figura 22: Vista Preeliminar Clase scanFrame

5.6. Clase confFrame

5.6.1. Descripción y prioridad

Esta clase define una Widget ventana que contiene Button's para acceder a submenús que se tienen expectados para acceder configuraciones; como en la ventana anterior. La diferencia es que estas configuraciones se encuentran en una instancia aparte, por lo que no interfieren con la representación gráfica.

Código 6: Clase confFrame

```
class confFrame(Tk.Toplevel):#Clase ventana configuración
    """"""

    #-----------------------------------------------------------------------
    def __init__(self, original):
        """Constructor"""
        self.original_frame = original
        Tk.Toplevel.__init__(self,height=windowHeight,width=windowWidth)# Inicializar
                # con parámetros de altura y anchura
        self.geometry(universalGeometry)#Delimitador de ventana
        rango_filas = range(0,20)#Confiugración de filas
        for i in rango_filas:
            self.rowconfigure(i,minsize=24)#Configuraación de columnas
        rango_columnas = range(0,10)
        for j in rango_columnas:
            self.columnconfigure(j,minsize=80)
        self.title("Configuraciones")
        self.confWidgets()

    def confWidgets(self):#Contenido de la ventana
        """Botones"""
        btnOp01 = Tk.Button(self, text="Opcion 1",command=self.op1)
        btnOp02 = Tk.Button(self, text="Opcion 2",command=self.op2)
        btnOp03 = Tk.Button(self, text="Opcion 3",command=self.op3)
        btnOp04 = Tk.Button(self, text="Opcion 4",command=self.op4)
        btnOp05 = Tk.Button(self, text="Opcion 5",command=self.op5)
        btnOp06 = Tk.Button(self, text="Opcion 6",command=self.op6)
        btnOp07 = Tk.Button(self, text="Opcion 7",command=self.op7)
        btnOp08 = Tk.Button(self, text="Opcion 8",command=self.op8)
        btnOp09 = Tk.Button(self, text="Opcion 9",command=self.op9)
        btnOp10 = Tk.Button(self, text="Opcion 10",command=self.op10)
        btnCerrar1 = Tk.Button(self, text="Cerrar", command=self.onClose)
##        btnCerrar2 = Tk.Button(self, text="Cerrar", command=self.onClose)
##        btnCerrar3 = Tk.Button(self, text="Cerrar", command=self.onClose)
##        btnCerrar4 = Tk.Button(self, text="Cerrar", command=self.onClose)
##        btnCerrar5 = Tk.Button(self, text="Cerrar", command=self.onClose)

        """Ubicación de botones"""
        btnOp01.grid(row=1,column=1,sticky=N+S+E+W)
        btnOp02.grid(row=4,column=1,sticky=N+S+E+W)
        btnOp03.grid(row=7,column=1,sticky=N+S+E+W)
        btnOp04.grid(row=10,column=1,sticky=N+S+E+W)
        btnOp05.grid(row=13,column=1,sticky=N+S+E+W)
        btnOp06.grid(row=1,column=3,sticky=N+S+E+W)
        btnOp07.grid(row=4,column=3,sticky=N+S+E+W)
        btnOp08.grid(row=7,column=3,sticky=N+S+E+W)
        btnOp09.grid(row=10,column=3,sticky=N+S+E+W)
        btnOp10.grid(row=13,column=3,sticky=N+S+E+W)
        btnCerrar1.grid(row=14,column=8,sticky=N+S+E+W)
##        btnCerrar2.grid(row=1,column=4,sticky=N+S+E+W)
##        btnCerrar3.grid(row=1,column=5,sticky=N+S+E+W)
##        btnCerrar4.grid(row=1,column=6,sticky=N+S+E+W)
##        btnCerrar5.grid(row=1,column=7,sticky=N+S+E+W)

        """Posicionar imagen"""
        #Imagen
        im=Image.open("cidesi.png")
        size = 320,271
```

```
        im.thumbnail(size)
        self.img = ImageTk.PhotoImage(im)
        panel = Label(self, image = self.img)
        panel.grid(row=3,column=5,columnspan=4,rowspan=5)

    #-----------------------------------------------------------------------
    def onClose(self):#Función cerrar ventana
        """"""
        self.destroy()
        self.original_frame.show()
#-----------------------------------------------------------------------
    def op1(self):#Función abrir opción 1
        """"""
        subFrame=op1Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op2(self):#Función abrir opción 2
        """"""
        subFrame=op2Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op3(self):#Función abrir opción 3
        """"""
        subFrame=op3Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op4(self):#Función abrir opción 4
        """"""
        subFrame=op4Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op5(self):#Función abrir opción 5
        """"""
        subFrame=op5Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op6(self):#Función abrir opción 6
        """"""
        subFrame=op6Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op7(self):#Función abrir opción 7
        """"""
        subFrame=op7Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op8(self):#Función abrir opción 8
        """"""
        subFrame=op8Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op9(self):#Función abrir opción 9
        """"""
        subFrame=op9Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op10(self):#Función abrir opción 10
        """"""
        subFrame=op10Frame(self)
        self.wait_window(subFrame)
```

A continuación se describen los elementos en el código 6. Como se verá la configuración es similar a la ventana anterior:

- Se crea una clase, de tipo Toplevel.
- La función de inicialización contiene a self y al objeto padre, de donde se proviene.

- Se hace una referencia al objeto padre.
- Se inicializa la clase con argumentos de self, la altura y la anchura.
- Se delimitan las dimensiones de la ventana.
- Se configuran las filas, especificando la cantidad de estas y con un ciclo for se determina un espacio mínimo de 24 pixeles entre cada una.
- Se configuran las columnas, especificando la cantidad de estas y con un ciclo for se determina un espacio mínimo de 80 pixeles entre cada una.
- Se establece un título.
- Se crean los Widgets.

  Primero se crean los Button's, y se determina su ubicación en la ventana empleando el administrador de geometría.

  Como en la ventana de bienvenida, también se emplea una imagen ornamental con el logotipo de esta institución; primero se abre la imagen y se ajusta su tamaño. Posteriormente se hace una referencia a la imagen y se coloca dentro de un Widget etiqueta.
- La función onClose es para cerrar la ventana.
- Las siguientes funciones no hacen más que abrir ventanas de diálogo que se encuentran planeadas para establecer configuraciones. Están configuradas de tal manera que no se tenga acceso a ninguna otra ventana hasta que la ventana de diálogo haya concluido. Primero se crea la subventana y se pone al widget actual en tiempo de espera hasta que haya concluido la interacción mediante la función wait_window.

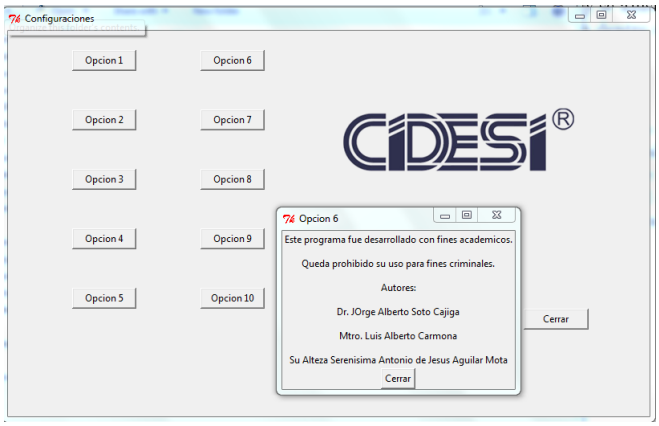

Figura 23: Vista Preeliminar Clase confFrame

5.7. Clases opFrame's

5.7.1. Descripción y prioridad

Los Widgets opFrames son ventanas de tipo Toplevel que se tiene expectado contengan opciones de configuración. Estas contienen una configuración distinta pues tienen un comportamiento de diálogo; es decir, no permiten el control de otra ventana hasta que hayan sido cerradas.

Código 7: Plantilla general para clases opFrame's

```
class op10Frame(Tk.Toplevel):#Clase ventana de opción 10
    """"""

    #-----------------------------------------------------------------------
    def __init__(self, original):
        """Constructor"""
        global bOp10
        bOp10=1#Bandera
        self.original_frame = original
        Tk.Toplevel.__init__(self)
##          self.geometry("400x240")
        self.title("Opcion 10")
        self.wm_attributes("-topmost",1)
        self.grab_set()
        self.op10Widgets()

    def op10Widgets(self):#Contenido de la ventana
        texto="Este programa fue desarrollado con fines academicos...."
        l1 = Label(self,text=texto)
        btn = Tk.Button(self, text="Cerrar", command=self.onClose)
        btn.grid(row=1,column=0)#Posicionamiento de objetos
        l1.grid(row=0,column=0)

    #-----------------------------------------------------------------------
    def onClose(self): #Función para cerrar la ventana.
        """"""
        global bOp10
        bOp10=0
        self.destroy()
        self.original_frame.deiconify()
```

En el código 7 se muestra la plantilla para declarar este tipo de ventanas, es igual para todas. El desarrollador puede agregar widgets o cambiar las dimensiones de la ventana si así le place.

- Se declara la clase opFrameXX; donde XX es el número correspondiente, para este proyecto se han declarado 10 Widgets de este tipo. Esta va a ser de tipo Toplevel.
- Se define la clase constructora, cargándola con parámetros self y la referencia de la ventana de la cual se proviene.
- En una implementación anterior se hacía uso de la variable global, ya no es prioritaria pero se dejó para apreciar su comportamiento en la ejecución del programa.
- Se guarda la referencia del widget de donde se proviene.
- Se inicializa el Widget Toplevel.
- No se especifica el tamaño de la ventana, ya que los widgets contenidos ajustan a la misma, pero si se desea un tamaño específico se puede descomentar.
- Se establece el título.

- Esta configuración es primordial en la ventana para que presente comportamiento de diálogo; se debe especificar dentro de sus atributos que va a estar por encima de todo lo demás.
- Además se especifíca que va a tomar el control total del programa impidiendo que se seleccionen otras ventanas.
- Se crean los Widgets de esta ventana.

  Para esta ventana sólo se crea una etiqueta; sin embargo queda abierto al desarrollador si necesita otro tipo de elementos.
- Se crea una función que destruye esta ventana y muestra la anterior.

5.8. Módulo de comunicación serial para leer datos externos.

5.8.1. Descripción y prioridad

Este módulo consiste en un script de naturaleza externa, para no ofuscar el entendimiento del código principal. En este módulo se encuentra la inicialización de la comunicación serial.

Código 8: Módulo de comunicación serial

```
import serial
import time
import struct
port = serial.Serial("/dev/ttyS0", baudrate=115200, timeout=0.02)
##port = serial.Serial("COM3", baudrate=115200, timeout=12)

def lec():
        while(1):
                start_time = time.time()
                port.write("S")
                rcv = port.read(1025)
                elapsed_time = time.time() - start_time
                print elapsed_time
                l = len(rcv)
                print l
                rango = range(0,l)
                nv=[]
                for i in rango:
                    ncs=struct.unpack("<b",rcv[i])
                    ncs1=ncs[0]
                    nv.append(ncs1)
                return nv
                break


if __name__ == "__main__":
        x=lec()
```

- Se importa el módulo de comunicación serial; es cuál es un módulo de Python que permite hacer interfaz entre el interpretador de Python y los puertos de comuicación serial donde se encuentre. (Nota: Es probable que no se tenga el módulo instalado. Para ello; es preciso consultar la sección de Instalación de librerías, que se encuentra en el capítulo Configuración del entorno de desarrollo.)
- Se empleó el módulo time, el cuál carga instancias para mediciones cronométricas.
- Se emplea el módulo struct; el cuál tiene funciones para conversión de datos.[27]
- Se declara una instancia de la clase Serial del módulo serial. En esta se inicializa la comunicación del puerto serial, se especifica el puerto que se va a emplear, se determina la velocidad de transferencia de datos; y opcionalmente, se establece un tiempo de respuesta en caso de errores para cortar la comunicación si no se tiene respuesta.
- Se determina una función que incluye las funciones de lectura en el puerto.

  Se establece un tiempo de inicio, empleando las librerías de tiempo.

  Hay que recordar que se está empleando una Launchpad para enviar la señal simulada.

  Se especifica que se van a leer 1025 bytes; que es la longitud de datos que mide la señal, y se asigna a la variable rcv.

  Se lee el tiempo transcurrido desde el inicio; asignándose a la variable $elapsed_time$.

Para comprobar que se haya recibido la señal sin errores, se lee la cantidad de bytes recibidos a través del puerto. Si este valor coincide con la cantidad de datos enviados, se tuvo una comunicación exitosa.

Se establece un rango correspondiente a la cantidad de datos leídos.

Se inicializa un arreglo vacío, que contendrá la señal.

En un ciclo for que transcurrirá conforme a lo determinado en el rango:

La funcion struct en este caso permite la conversión del string recibido en datos numéricos; que puedan ser leídos por Python. La b minúscula significa la conversión de datos con signo, y se especifica el valor que será convertido en la posición del string.

El resultado anterior es otorgado en formato Tuple, por lo que sólo se emplerá el primer valor de este.[44]

El valor obtenido se agrega al arreglo inicializado anteriormente.

- Se sale del ciclo for y se devuelve el arreglo para ser usado por funciones externas.
- Se rompe el ciclo while si sólo se require una lectura, eliminado este se pueden hacer varias lecturas. Esta configuración ya depende del desarrollador.
- En ambientes de Python, se acostumbra a hacer las cosas de forma Pythonica para la declaración de módulos. El siguiente elemento del código es similar a lo implementado en la sección "Programa Raíz", se establece un condicional que se asegura que el módulo sea el principal o si es importado. De ser el principal ejecutaría la función previamente descrita, de ser importado simplemente declararía las librerías importadas e implementaría la función declarada.

El código perteneciente al firmware empleado para pruebas en este desarrollo se encuentra en el apartado Anexos.9 De la misma manera, se anexa el diagrama de conexión entre Raspberry Pi y MSP430G2553 Launchpad en la figura 24. Se aprecia como están interconectados el pin TX de la Raspberry Pi con el pin RX de MSP430G2553 Launchpad. Lo mismo ocurre con el pin RX de Raspberry Pi y el pin TX de MSP430G2553 Launchpad

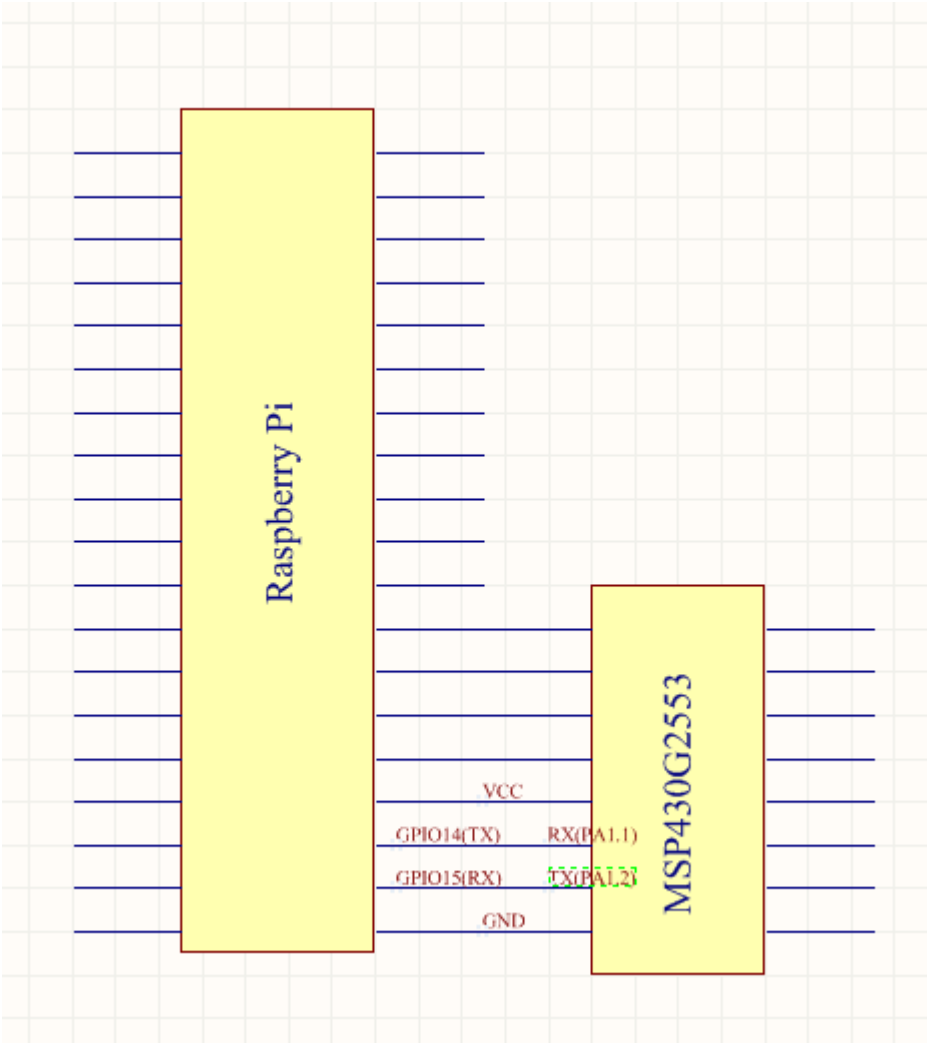

Figura 24: Diagrama de conexión entre Raspberry Pi y MSP430G2553 Launchpad.

6. Resultados

Se realizó el funcionamiento del software ejecutándose en Raspbian.

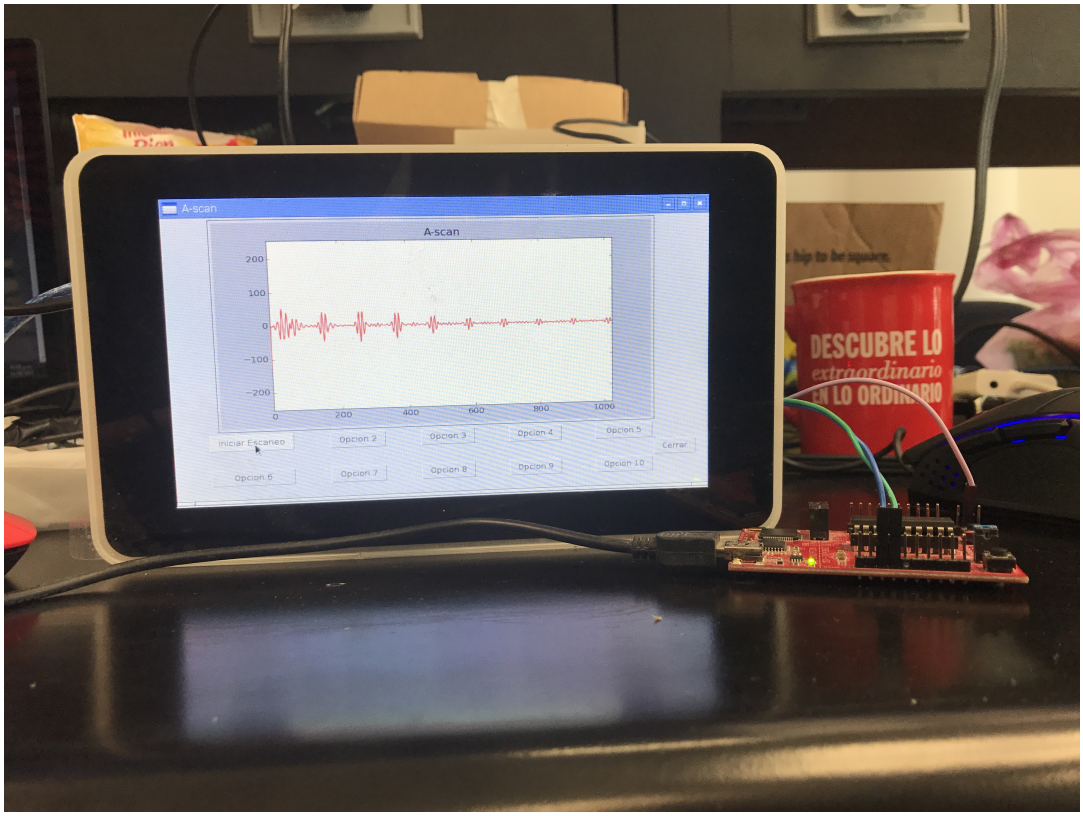

Figura 25: Gráfica final muestreada en Pantalla LCD de Raspberry Pi. La señal es enviada desde la Launchpad a la derecha.

Como se puede observar en la figura 25, hay una comunicación entre la Raspberry Pi. Para esta situación; la MSP430G2553 Launchpad se encuentra cargada con una programa que incluye una señal de muestra de 1025 datos; los cuales son enviados cada vez que recibe un caracter "S" por el mismo puerto serial.

A su vez se muestra la señal recibida en la pantalla de la interfaz; esta se muestra una vez y puede ser mandada a llamar las veces que se requiera.

En cuanto a tiempos; se apreció una transmisión en un lapso de ± 0.071 segundos considerando una tasa de transferencia en el puerto serial (Baudrate) de 115200 baudios. Esto es para una sola transmisión conteniendo 1025 bytes, es preciso realizar pruebas con la interfaz aplicando la transmisión continua de datos desde el FPGA para obtener un mejor veredicto con respecto a la viabilidad de seguir empleando este método o no para comunicarse con la Raspberry Pi. En otras pruebas realizadas, se logra una tasa

de refresco en la pantalla de hasta 200 imágenes por segundo.

Por otro lado, la cantidad de señales recibidas puede variar, dependiendo de la cantidad de datos que se tengan, por lo que es aconsejable diseñar un algoritmo para detección de errores, de modo que ignore las señales recibidas erróneamente. Dentro de las pruebas que se hicieron, se lograba efectivamente recibir una cantidad constante de señales con las configuraciones de comunicación serial previamente descritas, pero llegaban a presentarse ligeros errores durante esta transmisión.

7. Conclusiones

Se sabe que la comunicación serial es un método sencillo para hacer interfaz entre 2 dispositivos electrónicos ya que; ignorando los pines de alimentación, solo se requiere un par de señales que conecten a los dispositivos entre sí. La programación de la Raspberry Pi en el lenguaje de Programación Python es recomendada para desarrollar proyectos; especialmente si es requerido a corto plazo, por las siguientes razones:

- Se incluye por defecto en el sistema operativo Raspbian, lo que elimina el buscar paquetes externamente.
- Es de sintaxis sencilla y permite la implementación de la programación orientada a objetos de forma intuitiva.
- Debido a su popularidad; hay mucho soporte en la red, y se puede obtener apoyo consultándole a la amplia comunidad.
- Ya incluye Tkinter como backend para renderizado de video; y da pauta a ocupar otros, como Qt para presentar una aspecto más moderno.
- Python es considerado un programa de scripting; y similar a Java, el código escrito en un equipo puede ser trasladado a otro y esperar el mismo funcionamiento (siempre y cuando el entorno de desarrollo esté correctamente configurado.)
- Puedes mandar a llamar diferentes elementos del código desde otros archivos; sólamente debe ser importado en las cabeceras.
- En Raspbian, puede comunicarse directamente con los GPIO, por lo que no es necesario programar algún módulo de más bajo nivel.

Sobre el desarrollo electrónico; pues no se trata de nada complejo, simple y sencillamente es la conexión RX(Raspberry Pi a TX(MSP430G2553 Launchpad) y TX(Raspberry Pi) a RX(MSP430G2553 Launchpad), además de la alimentación de MSP430G2553 Launchpad desde Raspberry Pi.

7.1. Observaciones

De ser requerida alguna alternativa al método de comunicación; ya sea porque es muy lento, o porque no hay recepción adecuada de los datos, se recomiendan las siguientes alternativas:

- Emplear una comunicación SPI, pues la comunicación en los relojes de ambos dispositivos será la misma, determinada por el dispositivo determinado como maestro.
- Emplear un módulo de comunicación de más bajo nivel como C. Sin embargo no se recomienda del todo pues podría causar incompatibilidad con el software previamente desarrollado en Python.
- Realizar comunicación por el puerto Ethernet. Previamente se ha realizado un experimento en el cual se enviaban datos por el puerto Ethernet. Sin embargo, para esto habría que convertir o adaptar un Webserver dentro del dispositivo emisor (Microcontrolador o FPGA) y la Raspberry Pi estaría leyendo los datos recibidos. Se propone esto por la tasa de transferencia de datos en estas comunicaciones; pero puede que no sea lo más viable si ya se tiene una arquitectura electrónica fija.

8. Anexos

Glosario

$set_autoescaleony$ Método de conveniencia para vista de eje autoescalada simple.. 42

set_xlim Establece el límite de datos para el eje x. 42

set_ydata Establece los datos de np.array para y. Acepta arreglos de 1 dimernsión.. 42

set_ylim Establece el límite de datos para el eje y. 42

$wait_window$ Espera hasta que la ventana sea destrouida.. 42

axes Axes contiene la mayoría de los elementos de la figura, Ejes, Divisiones, Línea 2D, Polígonos ye stablece el sistema de coordenadas.. 41

axes.$draw_artist$ Este método sólo puede ser usado después de que un draw inicial que cachéa el renderer. Es usado para que actualizar eficientemente los datos de Axes. (axis ticks, labels, etc nos son actualizadas.). 42

backgrounds El fondo que se muestra en la gráfica.. 42

Broadcomm Empresa de circuitos integrados que desarrolla el chip empleado por la Raspberry Pi para procesamiento.. 21

Button El widget Button es un widget estándar de Tkinter, usado para implementar varias clases de Botones. Los botones pueden contener imágenes, texto y puedes asocair una función de Python a cada botón. Cuando el botón es presionado, Tkinter automáticamente llama a un método o na función.. 21, 37, 39, 41, 44, 46

canvas.blit Copia elementos de una parte de la memoria gráfica del canvas en background.. 42

canvas.draw Renderiza la figura. 42

canvas.restore Restaura el contexto de gráficos desde el stack, necesario únicamente para backends que guardan contexto de gráficos en el stack.. 42

CIDESI Es un centro público de investigación, pertenece al Sistema de Centros del Consejo Nacional de Ciencia y Tecnología de México. CIDESI tiene por finalidad contribuir al desarrollo del sector productivo de México mediante proyectos tecnológicos de Investigación e Innovación, además de proveer diversos servicios tecnológicos especializados.. 7, 21

columnconfigure Solicita o establece las propiedades de columna del índice de la columna de la geometría maestra Master.. 41

command En Tkinter, un callback es un código de Python que es llamado cuando algo ocurre. Por ejemplo, el widget Botón provee el callback "command" el cuál es llamado cuando el usuario cliquea el botón.. 41

Debian El proyecto Debian es una asociación de individuos que han hecho una causa común para crear un sistema operativo libre. Los sistemas actuales Debian actualmente usan el núcleo Linux o el núcleo FreeBSD.. 11, 21

dhcpcd.conf Es el archivo de configuración del cliente que provee servicios de DHCP y DHCPv6.. 25

ensayos no destructivos Cualquier tipo de prueba practicada a un material que no altere de forma permanente sus propiedades físicas, químicas, mecánicas o dimensionales.. 7, 13

figure El módulo figura provee el artist de mayor nivel Figure que contiene todos los elemntos de graficado.. 41

FigureCanvasTkAgg Matplotlib provee un objeto FigureCanvasTkAgg como parte del módulo *matplotlib.backends.backend$_t$kagg*. Un objeto FigureCanvasTkAgg encapsula una instancia Figure y se comporta como un objeto Tkinter. 42

FPGA Es un dispositivo programable que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada "in situ" mediante un lenguaje de descripción especializado.. 6, 9, 10, 24, 52, 54

Frame Es una región rectangular en la pantalla. El widget Frame es principalmente usado como maestro de geometría paa otros widgets, o proveer espaciado entre otros widgets.. 37, 41

GPIO GPIO (General Purpose Input/Output, Entrada/Salida de Propósito General) es un pin genérico en un chip, cuyo comportamiento (incluyendo si es un pin de entrada o salida) se puede controlar (programar) por el usuario en tiempo de ejecución.. 22, 23, 54

Grid El administrador de geometría Grid coloca los widgets en una tabla bidimensional. El widget maestro es dividido en un número de filas y columnas, y cada celda en la tabla resultante puede contener un widget.. 41

init Representa el constructor de un objeto/clase en Python.. 37

IPv4 Internet Protocol versión 4 (IPv4) es la cuarta versión del protocolo de el Protocolo de Interet. Es uno de los protocolos principales de los métodos de interconexión de redes basados en estándares., y fué la primera versión lanzadada para producción en la ARÑPANET EN 1983. Aún se encarga de dirigir la mayoría del tráfico de Internet al día de hoy; a pesar del constante lanzamiento del protocolo sucesor, IPv6.. 26

Label El widget Label es un widget estándar de Tkinter, usado para desplegar texto o imagen en pantalla.. 37

lines Este módulo contiene contienen todas las clases de líneas 2D que pueden dibujar una variedad de estilos de línea,, marcadores, y colores.. 42

Linux GNU/Linux es el término empleado para referirse a la combinación del sistema operativo GNU, desarrollado por la Free Software Foundation, y el núcleo(kernel) desarrollado por Linus Torvalds y la Linux Fondation. Su desarrollo es uno de los ejemplos más prominentes de software libre; todo su código libre puede ser utilizado, modificado y redistribuido libremente por cualquiera bajo los términos de la GPL y otra serie de licencias libres.. 11, 21, 29

Matplotlib Es una librería de graficado en 2D que produce figuras de calidad.. 30, 32, 41

MSP430G2553 Launchpad La familia Texas Instruments MSP430 de microncontroladores de Ultra-bajo consumo consiste en una serie de dispositivos que incluyn diferentes grupos de periféricos enfocados a varias aplicaciones. La aqruitectura, combinada con 5 modos de bajo consumo, está opptimizada para alcanzar una vida de batería extendida en aplicaciones de mediación portátiles. El dispositivo un poderoso procesador RISC, registros de 16-bits y generadores que contribuyen a la máxima eficiencia de código.. 5, 9–11, 24, 50–52, 54, 62

ncpa.cpl Comando para acceder a los adaptadores de red y sus configuraciones en Windows.. 25

numpy Es el paquete fundamental para computación científica en Python.. 30, 32

Pack El adminsitrador de geometría Pack muestra widgets en filas y columnas. Se pueden usar la opciones fill, expand y side para controlar este dministrador de geometría.. 37

pip Representa el constructor de un objeto/clase en Python.. 30

Putty Es un cliente de SSH y Telnet, desarrollado originalmente por Simon Tatham para la plataforma de Windows.. 11, 27

Python Lenguaje de programación cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje multiparadigma, ya que soporta programación orientada a objetos, programación imperativa y funcional. Es un lenguaje interpretado, usa un tipado dinámico y multiplataforma.. 8, 10, 11, 21, 31, 32, 35, 37, 38, 54

Python Image Library Librería que agrega capacidades de procesamiento de imagen al iterpretador de Python.. 32

Raspberry Pi Es un computador de placa reducida, computador de placa única o computador de placa simple (SBC) de bajo costo desarrollado en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas.. 3, 5, 7–11, 19–22, 25, 27–29, 33, 50–52, 54

Raspbian Raspbian es un sistema operativo libre, basado en Debian optimizado para la raspberry Pi. Raspbian incluye alrededor de 35000, softaware precompilado en un formato de fácil instalación para Raspberry Pi.. 10, 11, 21, 28, 52, 54

rowconfigure Solicita o establece las propiedades de fila del índice de la fila de la geometría maestra Master.. 41

self La variable self se refiere al objeto o variable ligado a la clase. Esto es por lo que asignar atributos a una instancia, se necesita saber a que instancia hacerlo. La variable self especifica a la instancia que se le aplican los atributos.. 37, 41, 45–47

serial Es el proceso de envío de datos de un bit a la vez, de forma secuencial, sobre un canal de comunicación o un bus.. 9, 10, 21, 23, 24, 39, 49

SSH (Secure SHell, en español: intérprete de órdenes seguro) es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder servidores privados a través de una puerta trasera (también llamada backend). Permite manejar por completo el servidor mediante un intérprete de comandos, y también puede redirigir el tráfico de X (Sistema de Ventanas X) para poder ejecutar programas gráficos si tenemos ejecutando un Servidor X (en sistemas Unix y Windows).. 25, 28

styles Define el color en que se representa la línea dentro de la gráfica.. 41

sys Este módulo provee acceso a algunas variables por el interpretador y a funciones y a funciones que actúan fuertemente con el interpretador.. 33

TkAgg Backend de renderización de AGG para Tkinter. 33

Tkinter Es la interfaz estándar de Python para el toolkit de interfaces gráficas de Tk. Consiste en un número de módulos. La interfaz es provista por de un módulo de extensión binaria llamada $_tkinter$; este módulo contiene la interfaz de bajo nivel de Tk, y nunca debe ser usada por programadores de aplicaciones.. 8, 10, 21, 31, 33, 35, 54

Toplevel El widget Toplevel funciona similarmente al widget Frame, pero es desplegado en una ventana separada de nivel superior. Estas ventanas usualmente tienen barras de título, bordes, y otras ventanas decorativas.. 41, 45, 47

transductor acústico electromecánico (Por sus siglas en inglés, Electromagnetic Acoustic Transducer-EMAT) es un transductor para generación y recepción de sonido sin contacto usando mecanismos electromagnéticos.. 15

Tuple Es una secuencia de objetos inmutables en Python. Similares a las listas; sin embargo, no son modificables como estas últimas.. 50

UART son las siglas en inglés de Universal Asynchronous Receiver-Transmitter, en español: Transmisor-Receptor Asíncrono Universal, es el dispositivo que controla los puertos y dispositivos serie. Se encuentra integrado en la placa base o en la tarjeta adaptadora del dispositivo.. 24

wheel Wheels son el nuevo estándar para distribución de Python. 30

Widgets En informática, un widget o artilugio es una pequeña aplicación o programa, usualmente presentado en archivos o ficheros pequeños que son ejecutados por un motor de widgets o Widget Engine. Entre sus objetivos están dar fácil acceso a funciones frecuentemente usadas y proveer de información visual. Aunque no es condición indispensable, los widgets suelen ser utilizados para ser "empotrados" en otra página web, copiando el código que el mismo widget pone a disposición del usuario. Dado que son pequeñas aplicaciones, los widgets pueden hacer todo lo que la imaginación desee e interactuar con servicios e información distribuida en Internet; pueden ser vistosos relojes en pantalla, notas, calculadoras, calendarios, agendas, juegos, ventanas con información del tiempo en su ciudad, incluso sistemas de tiendas de comercio, etcétera.. 31, 35, 37, 41, 46–48

Referencias

[1] Wikipedia, *Ensayo no destructivos*, obtenido el 23 de agosto de 2017 de https://es.wikipedia.org/wiki/Ensayo_no_destructivo

[2] TELSONIC Ultrasonics, *Ultrasonido industrial: Una breve explicacion*, obtenido el 23 de agosto de 2017 de https://www.telsonic.com/es/ultrasonido-industrial-una-breve-explicacion/

[3] Wikipedia, *Inspeccion por Ultrasonido*, obtenido el 23 de agosot de 2017 de https://es.wikipedia.org/wiki/Inspecci~A{^3}n_por_ultrasonido

[4] CIDESI, *Diablo instrumentado para inspeccion de ductos* obtenido el 23 de agosto de 2017 de http://cidesi.com/wsite/destacados/inspeccion-ductos.php

[5] Raspberry Pi Foundation, *Raspberry Pi*, obtenido el 23 de mayo e 2017 de https://www.raspberrypi.org

[6] Texas Instruments, *MSP430g2553*, obtenido el 23 de agosto de 2017 de http://www.ti.com/product/MSP430G2553

[7] NAVA Balanzar, Luciano, *Diseno de un sistema electronico para la medicion de espesores por ultrasonido*, Queretaro, Qro., Mexico, CIDESI, 2010.

[8] GOMEZ, Alejandro, *Diseno y desarrollo de un sistema electronico para la generacion y adquisicion de senales de ultrasonido.*

[9] Linux.org, *LINUX*, obtenido el 23 de agosto de 2017 de https://www.linux.org

[10] Python Software Foundation, *Python* obtenido el 23 de agosto de 2017 de https://www.python.org

[11] SAGRERO Rivera, Jorge, *Ultrasonido Industrial Nivel I*, Centro de IngenierÃa y Desarrollo Industrial

[12] LLOG S.A de C.V., *Curso de ultrasonido industrial II*, obtenido el 23 de agosto de 2017 de http://www.llogsa.mx/curso-de-ultrasonido-industrial-ii-2/

[13] ZION NDT, *Equipos para ultrasonido industrial*, obtenido el 23 de agosto de 2017 de http://www.zion-ndt.mx/metodo/ultrasonido/equipos-para-ultrasonido-industrial/

[14] MAMAMI, RubÃ©n, *Calibracion En Distancia con equipos de Ultrasonido Industrial con el bloque IIW Cap. 1-2*, obtenido el 23 de agosto de 2017 de https://www.youtube.com/watch?v=8xdlOioW64E

[15] Alibaba, *S60 Portable Ultrasonic Flaw Detector (NDT, ultrasonic, ultrasound, A scan , digital portable ultrasonic flaw detector)*, obtenido el 23 de agosto de 2017 de https://www.alibaba.com/product-detail/S60-Portable-Ultrasonic-Flaw-Detector-NDT_2002943310.html

[16] Wikipedia, *Raspberry Pi*, obtenidoe el 23 de agosto de 2017 de https://es.wikipedia.org/wiki/Raspberry_Pi

[17] Python, *TkInter*, obtenido el 23 de agosto de 2017 de https://wiki.python.org/moin/TkInter

[18] Raspberry para torpes, *Mando de Megadrive en la Raspberry Pi por GPIO* obtenido el 23 de agosto de 2017 de https://raspberryparatorpes.net/proyectos/mando-de-megadrive-en-la-raspberry-pi-por-gpio/

[19] Pistachitos.com, *Comenzando con Raspbian*, obtenido el 23 de agosto de 2017 de http://pistachitos.com/guias/comenzando-con-raspbian/

[20] Python Argentina, *Tutorial de Python (y Django!) en Espanol*, obtenido el 23 de agosto de 2017 de http://docs.python.org.ar/tutorial/

[21] pixabay, obtenido el 23 de agosto de 2017 de https://pixabay.com/en/port-serial-port-plug-connector-150929/

[22] Aendruk, *How can I add a new user as sudoer using the command line?*, 15 de mayo de 2010, obtenido el 3 de julio de 2017 de https://askubuntu.com/questions/7477/how-can-i-add-a-new-user-as-sudoer-using-the-command-line

[23] alex, Enero de 2014, obtenido el 3 de julio de 2017 de http://raspi.tv/2013/rpi-gpio-basics-5-setting-up-and-using-outputs-with-rpi-gpio

[24] O'Reilly Media., *Connecting a Push Switch*, 2013, obtenido el 3 de julio de 2017 de http://razzpisampler.oreilly.com/ch07.html

[25] alex, *RPi.GPIO update and detecting BOTH rising and falling edges* Enero de 2014, obtenido el 3 de julio de 2017 de http://raspi.tv/2014/rpi-gpio-update-and-detecting-both-rising-and-falling-edges

[26] Joe Kington, *why is plotting with Matplotlib so slow?* 21 de Enero de 2012, obtenido el 3 de julio de 2017 de https://stackoverflow.com/questions/8955869/why-is-plotting-with-matplotlib-so-slow

[27] Python Software Foundation, *Interpret strings as packed binary data* obtenido el 3 de julio de 2017 de https://docs.python.org/2/library/struct.html

[28] tutorials Point, *Python Tkinter place() Method* obtenido el 3 de julio de 2017 de https://www.tutorialspoint.com/python/tk_place.htm

[29] Secnetix, *Python: Lambda Functions* obtenido el 3 de julio de 2017 de http://www.secnetix.de/olli/Python/lambda_functions.hawk

[30] Effbot.org, *The Tkinter Grid Geometry Manager* obtenido el 3 de julio de 2017 de http://effbot.org/tkinterbook/grid.htm

[31] Bryan Oakley, *Why does place() not work for widgets in a Frame in a Canvas?* 24 de Junio de 2015, obtenido el 3 de julio de 2017 de https://stackoverflow.com/questions/31028874/why-does-place-not-work-for-widgets-in-a-frame-in-a-canvas

[32] Mike, *Tkinter: How to Show / Hide a Window* 26 de Julio de 2012, obtenido el 3 de julio de 2017 de http://effbot.org/tkinterbook/grid.htm

[33] cdbitesky, *How do I change button size in Python* 9 de Enero de 2013, obtenido el 3 de julio de 2017 de https://stackoverflow.com/questions/14247709/how-do-i-change-button-size-in-python

[34] Sam, *Grid within a frame?* 4 de mayo de 2010, obtenido el 3 de julio de 2017 de https://stackoverflow.com/questions/2763266/grid-within-a-frame

[35] Effbot, *The Tkinter Place Geometry Manager* obtenido el 3 de julio de 2017 de `http://effbot.org/tkinterbook/place.htm`

[36] Effbot, *The Tkinter Button Widget* obtenido el 3 de julio de 2017 de `http://effbot.org/tkinterbook/button.htm`

[37] Devshed, *Thread: Place manager not working in Tkinter* 2 de mayo de 2014, obtenido el 3 de julio de 2017 de `https://stackoverflow.com/questions/7831655/python-classes-in-tkinters-grid-manager`

[38] unutbu, *Python Classes in Tkinter's Grid Manager* 20 de octubre de 2011, obtenido el 3 de julio de 2017 de `http://forums.devshed.com/python-programming-11/manager-tkinter-961337.html`

[39] $tobias_k$, *Python TkInter PhotoImage* 20 de julio de 2013, obtenido el 3 de julio de 2017 de `https://stackoverflow.com/questions/17760871/python-tkinter-photoimage`

[40] Yasoob, *The self variable in Python explained* 7 de agosto de 2013, obtenido el 3 de julio de 2017 de `https://pythontips.com/2013/08/07/the-self-variable-in-python-explained/`

[41] Effbot, *Dialog Windows* obtenido el 3 de julio de 2017 de `http://effbot.org/tkinterbook/tkinter-dialog-windows.htm`

[42] GOHLKE, Christoph; *Unofficial Windows Binaries for Python Extension Packages* obtenido el 12 de julio de 2017 de `http://www.lfd.uci.edu/~gohlke/pythonlibs/`

[43] MICROSOFT, *Python Development Tools* obtenido el 12 de julio de 2017 de `https://www.visualstudio.com/vs/python/`

[44] Tutorials Point, *Tuples Pyton* obtenido el 14 de julio de 2017 de `https://www.tutorialspoint.com/python/python_tuples.htm`

[45] Wikipedia, *Comunicacion Serie* obtenido el 14 de julio de 2017 de `https://es.wikipedia.org/wiki/Comunicaci~A{^3}n_serie`

[46] Python Wheels, *Wheels* obtenido el 14 de julio de 2017 de `http://pythonwheels.com`

[47] Software in the Public Interest, Inc; *Debian, the universal operating system*, obtenido el 4 de agosto de 2017 de `https://www.debian.org/index.es.html`

8.1. Códigos implementados

Código 9: Programación dentro de MSP430G2553 Launchpad

```
/* Example code demonstrating the use of pointers.
 * It uses the hardware UART on the MSP430G2553 to receive
 * and transmit data back to a host computer over the USB connection on the MSP430
 * launchpad.
 * Note: After programming using CCS it is necessary to stop debugging and reset the uC before
 * connecting the terminal program to transmit and receive characters.
 * This demo will transmit the characters AB in response to a S been sent each time S is sent the
 * order of the characters is swaped. If agen any other character is sent and unknown command
 * response is sent.
 */

#include "msp430g2553.h"
void UARTSendArray(char *TxArray, int ArrayLength);
void UARTSendChar(char *TxChar);
void SwapChars(char *a, char *b);
const int vecLen = 1025;
const char vecSeg[vecLen]={129,251,0,0,255,255,255,0,1,3,4,3,255,251,...
...247,247,249,254,4,9,12,10,14,11,4,248,234,222,218,227,249,20,43,53,...
46,25,0,233,220,217,221,229,240,253,12,27,38,40,31,12,246,229,223,230,...
242,0,8,8,2,251,249,253,6,16,22,22,10,255,241,230,227,232,244,2,10,20,...
21,17,10,2,252,247,245,244,245,248,253,2,6,9,10,8,4,0,253,251,251,251,...
252,254,255,1,2,2,3,2,2,1,0,255,254,254,253,254,254,0,1,2,2,2,2,1,255,...
254,254,254,254,0,1,2,2,1,255,253,252,253,0,4,8,11,9,3,250,239,232,232,...
241,2,21,36,40,30,10,240,219,209,216,234,3,25,36,35,24,8,249,240,237,240,...
245,251,255,1,3,5,7,8,8,5,1,253,250,248,249,251,254,1,3,4,4,3,1,0,0,255,254,...
254,254,254,255,0,0,1,1,1,1,1,0,0,255,255,254,254,255,0,1,1,2,1,0,0,255,255,255,...
255,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,254,254,254,0,2,6,9,10,6,254,243,234,...
231,238,253,17,34,41,33,12,240,217,207,214,236,8,32,42,38,21,0,238,230,232,240,251,...
4,9,9,7,4,2,1,1,1,0,254,252,251,252,253,0,2,3,3,2,1,255,254,254,255,1,2,2,1,255,...
253,251,252,254,0,3,5,6,4,1,254,251,251,252,254,0,2,3,3,2,1,255,255,255,255,255,...
255,255,255,255,0,0,1,2,2,2,1,0,254,253,253,253,253,254,0,3,7,9,9,6,254,245,237,234,...
238,251,10,28,35,30,12,245,224,215,220,238,6,25,35,31,17,0,242,235,237,244,253,4,...
7,7,4,1,0,0,0,1,1,0,255,253,253,253,255,0,1,2,1,0,0,0,1,1,2,1,0,254,252,251,252,...
255,2,5,6,5,3,255,252,251,251,252,255,1,3,4,3,1,0,255,254,254,254,255,0,0,0,0,0,...
0,1,1,1,1,0,255,255,254,253,254,254,254,0,1,3,4,5,5,4,1,253,248,244,243,246,254,8,18,...
23,21,10,250,235,227,229,241,1,17,25,24,15,2,246,240,241,246,253,3,6,6,3,1,255,...
255,0,1,2,1,255,253,252,253,255,1,3,3,2,1,255,254,255,0,1,2,1,255,253,252,253,255,...
2,4,5,4,1,254,252,251,252,254,0,2,3,3,3,1,0,254,253,253,254,255,1,2,3,2,1,0,254,...
254,255,255,0,1,1,0,255,254,255,255,1,2,2,3,2,2,0,255,252,250,249,249,252,2,...
9,14,14,10,1,247,239,237,242,252,7,15,17,10,4,252,246,245,248,253,2,4,5,3,1,255,...
255,255,0,1,1,0,255,254,254,255,0,1,1,1,0,255,255,255,1,2,2,1,255,253,253,253,...
255,1,3,4,3,1,255,253,253,253,255,1,2,3,2,1,0,254,253,253,254,255,0,2,2,2,1,0,...
255,254,254,255,0,1,1,1,0,255,254,255,255,0,1,2,2,2,2,1,255,254,251,250,250,252,...
1,6,10,11,9,2,250,244,242,245,252,4,10,12,10,4,254,249,248,250,254,1,3,3,2,0,...
255,255,255,1,1,1,0,255,255,255,255,0,1,1,0,255,255,255,0,1,2,2,1,255,254,253,...
254,255,1,2,3,2,0,255,254,254,254,255,1,2,2,2,1,0,254,253,253,254,255,1,2,3,3,...
1,0,254,253,254,255,0,1,1,1,0,255,255,255,255,0,0,1,1,1,1,1,0,254,252,251,251,...
253,1,6,9,9,6,0,250,246,245,248,254,4,8,9,6,2,254,251,251,253,0,1,1,0,255,254,...
255,0,1,2,2,1,255,254,253,254,255,0,1,1,1,0,255,255,0,1,1,1,0,255,254,254,254,...
255,1,1,1,1,0,255,254,255,0,1,2,2,1,0,254,253,253,253,255,0,2,3,3,2,1,255,254,...
253,253,255,0,2,2,2,1,255,254,254,255,0,1,1,1,1,1,1,1,0,0,254,252,251,252,255,...
3,7,9,7,3,253,248,246,247,251,1,6,8,7,4,0,254,253,253,254,255,0,255,255,255,0,...
1,2,3,2,1,255,253,253,254,255,1,2,2,1,0,255,254,255,0,1,2,1,0,255,255,254,255,...
255,0,0,0,0,0,0,1,2,2,1,0,255,254,253,253,254,0,1,2,2,2,1,0,0,255,254,254,255,...
255,0,1,1,1,1,0,255,255,255,0,0,1,0,0,0,0,1,1,1,255,253,251,251,253,0,4,7,8,6,1,...
252,248,246,248,253,2,6,8,7,4,0,253,251};
//const char vecLen = 5;
//const char vecSeg[vecLen]={48,49,50,51,52};

volatile char data;
volatile char x1;
volatile char x2;
int i;
```

```
void main(void)

{
  WDTCTL = WDTPW + WDTHOLD; // Stop WDT

  P1DIR |= BIT0 + BIT6;      // Set the LEDs on P1.0, P1.6 as outputs
  P1OUT = BIT0;              // Set P1.0

//     BCSCTL1 = CALBC1_1MHZ;    // Set DCO to 1MHz
//   DCOCTL = CALDCO_1MHZ;     // Set DCO to 1MHz

  BCSCTL1 = CALBC1_16MHZ;       //Coloca el DCO a 16 MHz
  DCOCTL = CALDCO_16MHZ;

  /* Configure hardware UART */
  P1SEL = BIT1 + BIT2 ;      // P1.1 = RXD, P1.2=TXD
  P1SEL2 = BIT1 + BIT2 ;     // P1.1 = RXD, P1.2=TXD
  UCA0CTL1 |= UCSSEL_2;      // Use SMCLK

//   UCA0BR0 = 104;            // Set baud rate to 9600 with 1MHz clock (Data Sheet 15.3.13)
//   UCA0BR1 = 0;              // Set baud rate to 9600 with 1MHz clock
//    UCA0MCTL = UCBRS0;        // Modulation UCBRSx = 1

//   UCA0BR0 = 130;                                // 16MHz 9600
//   UCA0BR1 = 6;                                  // 16MHz 9600
//   UCA0MCTL = UCBRS_6;                            // Modulation UCBRSx = 1

  UCA0BR0 = 138;                                // 16MHz 115200
  UCA0BR1 = 0;                                  // 16MHz 115200
  UCA0MCTL = UCBRS_6;                            // Modulation UCBRSx = 1

  UCA0CTL1 &= ~UCSWRST;      // Initialize USCI state machine
  IE2 |= UCA0RXIE;           // Enable USCI_A0 RX interrupt

  x1 = 'A';
  x2 = 'B';
//   while(1)
//   {
//       for(i = 0;i<=vecLen;i++)
//       {
//           UARTSendChar(vecSeg[i]);
////           UARTSendChar(x2);
////           UARTSendArray("\n\r", 2);
//           if (i>=vecLen)
//           {
//               i = 0;
//           }
//       }
//   }

  __bis_SR_register(LPM0_bits + GIE); // Enter LPM0, interrupts enabled
}

//// Echo back RXed character, confirm TX buffer is ready first
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
  data = UCA0RXBUF;
  switch(data){
  case 'S':
    {
//       SwapChars(&x1, &x2);      // Pass the pointers to variables x1 and x2 to the function

//       UARTSendChar(x1);
//       UARTSendChar(x2);
//       UARTSendArray("\n\r", 2);
      UARTSendArray(vecSeg, vecLen);
      P1OUT &= ~BIT6;              // Set P1.0
    }
```

```
    break;
  default:
    {
      UARTSendArray("Unknown Command: ", 17);
      UARTSendChar(data);
      UARTSendArray("\n\r", 2);
    }
    break;
  }
}

void UARTSendArray(char *TxArray, int ArrayLength){
  // Send number of bytes Specified in ArrayLength in the array at using the hardware UART 0
  // Example usage: UARTSendArray("Hello", 5);
  // int data[2]={1023, 235};
  // UARTSendArray(data, 4);  // Note because the UART transmits bytes it is necessary to send
                              // two bytes for each integer hence the data length is twice the
                              // array length
    while(ArrayLength--){              // Loop until StringLength == 0 and post decrement
    while(!(IFG2 & UCA0TXIFG));        // Wait for TX buffer to be ready for new data
    UCA0TXBUF = *TxArray++;            // Write the character at the location specified by the
  }                                    // pointer and post increment
}

void UARTSendChar(char *TxChar){
  // Send number of bytes Specified in ArrayLength in the array at using the hardware UART 0
  // Example usage: UARTSendArray('A');
  while(!(IFG2 & UCA0TXIFG));        // Wait for TX buffer to be ready for new data
  UCA0TXBUF = TxChar;              //Write the character at the location specified py the pointer
}

void SwapChars(char *a, char *b){
  char c = 0;       // initialise temporary variable c
  P1OUT |= BIT6;              // Set P1.0
  c = *a;       // copy the value in memory location a in variable c
  *a = *b;      // copy the value stored in memory location b into memory location a
  *b = c;       // copy the value temporarily stored in c into memory location b

}
```

Código 10: Código completo del programa principal de la Interfaz Gráfica

```
#!/usr/bin/python
# -*- coding: cp1252 -*-
from Tkinter import *#Librería gráfica
import Tkinter as Tk
from PIL import ImageTk, Image#Importar libreria de imágenes

import numpy as np
import matplotlib
matplotlib.use('TkAgg')
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.pyplot as plt
import time
import sys
import sm
from sm import lec as rd

bA=0#Bandera subventana
bOp1=0#Bandera subventana opción 1
bOp2=0#Bandera subventana opción 2
bOp3=0#Bandera subventana opción 3
bOp4=0#Bandera subventana opción 1
bOp5=0#Bandera subventana opción 1
bOp6=0#Bandera subventana opción 1
bOp7=0#Bandera subventana opción 1
bOp8=0#Bandera subventana opción 1
bOp9=0#Bandera subventana opción 9
bOp10=0#Bandera subventana opción 10
##buttonWidth=10#Altura del boton
windowHeight=480#Altura ventana
windowWidth=800#Anchura Ventana
universalGeometry ='%dx%d' %(windowWidth,windowHeight)#Delimitador ventana
#######################################################################
class op10Frame(Tk.Toplevel):#Clase ventana de opción 10
    """"""

    #-------------------------------------------------------------------
    def __init__(self, original):
        """Constructor"""
        global bOp10
        bOp10=1#Bandera
        self.original_frame = original
        Tk.Toplevel.__init__(self)
##        self.geometry("400x240")
        self.title("Opcion 10")
        self.wm_attributes("-topmost",1)
        self.grab_set()
        self.op10Widgets()

    def op10Widgets(self):#Contenido de la ventana
        texto="Este programa fue desarrollado con fines academicos...."
        l1 = Label(self,text=texto)
        btn = Tk.Button(self, text="Cerrar", command=self.onClose)
        btn.grid(row=1,column=0)#Posicionamiento de objetos
        l1.grid(row=0,column=0)

    #-------------------------------------------------------------------
    def onClose(self): #Función para cerrar la ventana.
        """"""
        global bOp10
        bOp10=0
        self.destroy()
        self.original_frame.deiconify()
#######################################################################
class op9Frame(Tk.Toplevel):#Clase ventana de opción 9
    """"""

    #-------------------------------------------------------------------
    def __init__(self, original):
```

```
        """Constructor"""
        global bOp9
        bOp9=1#Bandera
        self.original_frame = original
        Tk.Toplevel.__init__(self)
##          self.geometry("400x240")
        self.title("Opcion 9")
        self.wm_attributes("-topmost",1)
        self.grab_set()
        self.op9Widgets()

    def op9Widgets(self):#Contenido de la ventana
        texto="Este programa fue desarrollado con fines academicos...."
        l1 = Label(self,text=texto)
        btn = Tk.Button(self, text="Cerrar", command=self.onClose)
        btn.grid(row=1,column=0)#Posicionamiento de objetos
        l1.grid(row=0,column=0)

    #-----------------------------------------------------------------------
    def onClose(self): #Función para cerrar la ventana.
        """"""
        global bOp9
        bOp9=0
        self.destroy()
        self.original_frame.deiconify()
#######################################################################
class op8Frame(Tk.Toplevel):#Clase ventana de opción 8
    """"""

    #-----------------------------------------------------------------------
    def __init__(self, original):
        """Constructor"""
        global bOp7
        bOp8=1#Bandera
        self.original_frame = original
        Tk.Toplevel.__init__(self)
##          self.geometry("400x240")
        self.title("Opcion 8")
        self.wm_attributes("-topmost",1)
        self.grab_set()
        self.op8Widgets()

    def op8Widgets(self):#Contenido de la ventana
        texto="Este programa fue desarrollado con fines academicos...."
        l1 = Label(self,text=texto)
        btn = Tk.Button(self, text="Cerrar", command=self.onClose)
        btn.grid(row=1,column=0)#Posicionamiento de objetos
        l1.grid(row=0,column=0)

    #-----------------------------------------------------------------------
    def onClose(self): #Función para cerrar la ventana.
        """"""
        global bOp8
        bOp8=0
        self.destroy()
        self.original_frame.deiconify()
#######################################################################
class op7Frame(Tk.Toplevel):#Clase ventana de opción 7
    """"""

    #-----------------------------------------------------------------------
    def __init__(self, original):
        """Constructor"""
        global bOp7
        bOp7=1#Bandera
        self.original_frame = original
        Tk.Toplevel.__init__(self)
##          self.geometry("400x240")
        self.title("Opcion 7")
```

```
        self.wm_attributes("-topmost",1)
        self.grab_set()
        self.op7Widgets()

    def op7Widgets(self):#Contenido de la ventana
        texto="Este programa fue desarrollado con fines academicos...."
        l1 = Label(self,text=texto)
        btn = Tk.Button(self, text="Cerrar", command=self.onClose)
        btn.grid(row=1,column=0)#Posicionamiento de objetos
        l1.grid(row=0,column=0)

    #-----------------------------------------------------------------------
    def onClose(self): #Función para cerrar la ventana.
        """"""
        global bOp7
        bOp7=0
        self.destroy()
        self.original_frame.deiconify()
#######################################################################
class op6Frame(Tk.Toplevel):#Clase ventana de opción 6
    """"""

    #-----------------------------------------------------------------------
    def __init__(self, original):
        """Constructor"""
        global bOp6
        bOp6=1#Bandera
        self.original_frame = original
        Tk.Toplevel.__init__(self)
##        self.geometry("400x240")
        self.title("Opcion 6")
        self.wm_attributes("-topmost",1)
        self.grab_set()
        self.op6Widgets()

    def op6Widgets(self):#Contenido de la ventana
        texto="Este programa fue desarrollado con fines academicos...."
        l1 = Label(self,text=texto)
        btn = Tk.Button(self, text="Cerrar", command=self.onClose)
        btn.grid(row=1,column=0)#Posicionamiento de objetos
        l1.grid(row=0,column=0)

    #-----------------------------------------------------------------------
    def onClose(self): #Función para cerrar la ventana.
        """"""
        global bOp6
        bOp6=0
        self.destroy()
        self.original_frame.deiconify()
#######################################################################
class op5Frame(Tk.Toplevel):#Clase ventana de opción 5
    """"""

    #-----------------------------------------------------------------------
    def __init__(self, original):
        """Constructor"""
        global bOp5
        bOp5=1#Bandera
        self.original_frame = original
        Tk.Toplevel.__init__(self)
##        self.geometry("400x240")
        self.title("Opcion 5")
        self.wm_attributes("-topmost",1)
        self.grab_set()
        self.op5Widgets()

    def op5Widgets(self):#Contenido de la ventana
        texto="Este programa fue desarrollado con fines academicos...."
        l1 = Label(self,text=texto)
```

```
        btn = Tk.Button(self, text="Cerrar", command=self.onClose)
        btn.grid(row=1,column=0)#Posicionamiento de objetos
        l1.grid(row=0,column=0)

    #-----------------------------------------------------------------------
    def onClose(self): #Función para cerrar la ventana.
        """"""
        global bOp5
        bOp5=0
        self.destroy()
        self.original_frame.deiconify()
#######################################################################
class op4Frame(Tk.Toplevel):#Clase ventana de opción 4
    """"""

    #-----------------------------------------------------------------------
    def __init__(self, original):
        """Constructor"""
        global bOp4
        bOp4=1#Bandera
        self.original_frame = original
        Tk.Toplevel.__init__(self)
##          self.geometry("400x240")
        self.title("Opcion 4")
        self.wm_attributes("-topmost",1)
        self.grab_set()
        self.op4Widgets()

    def op4Widgets(self):#Contenido de la ventana
        texto="Este programa fue desarrollado con fines academicos...."
        l1 = Label(self,text=texto)
        btn = Tk.Button(self, text="Cerrar", command=self.onClose)
        btn.grid(row=1,column=0)#Posicionamiento de objetos
        l1.grid(row=0,column=0)

    #-----------------------------------------------------------------------
    def onClose(self): #Función para cerrar la ventana.
        """"""
        global bOp4
        bOp4=0
        self.destroy()
        self.original_frame.deiconify()
#######################################################################
class op3Frame(Tk.Toplevel):#Clase ventana de opción 3
    """"""

    #-----------------------------------------------------------------------
    def __init__(self, original):
        """Constructor"""
        global bOp3
        bOp3=1#Bandera
        self.original_frame = original
        Tk.Toplevel.__init__(self)
##          self.geometry("400x240")
        self.title("Opcion 3")
        self.wm_attributes("-topmost",1)
        self.grab_set()
        self.op3Widgets()

    def op3Widgets(self):#Contenido de la ventana
        texto="Este programa fue desarrollado con fines academicos...."
        l1 = Label(self,text=texto)
        btn = Tk.Button(self, text="Cerrar", command=self.onClose)
        btn.grid(row=1,column=0)#Posicionamiento de objetos
        l1.grid(row=0,column=0)

    #-----------------------------------------------------------------------
    def onClose(self): #Función para cerrar la ventana.
        """"""
```

```
        global bOp3
        bOp3=0
        self.destroy()
        self.original_frame.deiconify()
#######################################################################
class op2Frame(Tk.Toplevel):#Clase ventana de opción 2
    """"""

    #-------------------------------------------------------------------
    def __init__(self, original):
        """Constructor"""
        global bOp2
        bOp2=1#Bandera
        self.original_frame = original
        Tk.Toplevel.__init__(self)
##          self.geometry("400x240")
        self.title("Opcion 2")
        self.wm_attributes("-topmost",1)
        self.grab_set()
        self.op2Widgets()

    def op2Widgets(self):#Contenido de la ventana
        texto="Este programa fue desarrollado con fines academicos...."
        l1 = Label(self,text=texto)
        btn = Tk.Button(self, text="Cerrar", command=self.onClose)
        btn.grid(row=1,column=0)#Posicionamiento de objetos
        l1.grid(row=0,column=0)

    #-------------------------------------------------------------------
    def onClose(self): #Función para cerrar la ventana.
        """"""
        global bOp2
        bOp2=0
        self.destroy()
        self.original_frame.deiconify()
#######################################################################
class op1Frame(Tk.Toplevel):#Clase ventana de opción 1
    """"""

    #-------------------------------------------------------------------
    def __init__(self, original):
        """Constructor"""
        global bOp1
        bOp1=1#Bandera
        self.original_frame = original
        Tk.Toplevel.__init__(self)
##          self.geometry("400x240")
        self.title("Opcion 1")
        self.wm_attributes("-topmost",1)
        self.grab_set()
        self.op1Widgets()

    def op1Widgets(self):#Contenido de la ventana
        texto="Este programa fue desarrollado con fines academicos...."
        l1 = Label(self,text=texto)
        btn = Tk.Button(self, text="Cerrar", command=self.onClose)
        btn.grid(row=1,column=0)#Posicionamiento de objetos
        l1.grid(row=0,column=0)

    #-------------------------------------------------------------------
    def onClose(self): #Función para cerrar la ventana.
        """"""
        global bOp1
        bOp1=0
        self.destroy()
        self.original_frame.deiconify()
#######################################################################
class aboutFrame(Tk.Toplevel):#Clase ventana de información
    """"""
```

```
    #-----------------------------------------------------------------------
    def __init__(self, original):
        """Constructor"""
        global bA
        bA=1#Bandera
        self.original_frame = original
        Tk.Toplevel.__init__(self)
##        self.geometry("400x240")
        self.title("Acerca de")
        self.wm_attributes("-topmost",1)
        self.grab_set()
        self.abWidgets()

    def abWidgets(self):#Contenido de la ventana
        texto="Este programa fue desarrollado con fines academicos...."
        l1 = Label(self,text=texto)
##        img = ImageTk.PhotoImage(Image.open("cidesi.png"))
##        panel = Label(root, image = img)
##        panel.pack(x=200,y=300)
        btn = Tk.Button(self, text="Cerrar", command=self.onClose)
        btn.grid(row=1,column=0)#Posicionamiento de objetos
        l1.grid(row=0,column=0)

    #-----------------------------------------------------------------------
    def onClose(self): #Funcipón para cerrar la ventana.
        """"""
        global bA
        bA=0
        self.destroy()
        self.original_frame.show()
#######################################################################
class confFrame(Tk.Toplevel):#Clase ventana configuración
    """"""

    #-----------------------------------------------------------------------
    def __init__(self, original):
        """Constructor"""
        self.original_frame = original
        Tk.Toplevel.__init__(self,height=windowHeight,width=windowWidth)# Inicializar
                # con parámetros de altura y anchura
        self.geometry(universalGeometry)#Delimitador de ventana
        rango_filas = range(0,20)#Confiugración de filas
        for i in rango_filas:
            self.rowconfigure(i,minsize=24)#Configuraación de columnas
        rango_columnas = range(0,10)
        for j in rango_columnas:
            self.columnconfigure(j,minsize=80)
        self.title("Configuraciones")
        self.confWidgets()

    def confWidgets(self):#Contenido de la ventana
        """Botones"""
        btnOp01 = Tk.Button(self, text="Opcion 1",command=self.op1)
        btnOp02 = Tk.Button(self, text="Opcion 2",command=self.op2)
        btnOp03 = Tk.Button(self, text="Opcion 3",command=self.op3)
        btnOp04 = Tk.Button(self, text="Opcion 4",command=self.op4)
        btnOp05 = Tk.Button(self, text="Opcion 5",command=self.op5)
        btnOp06 = Tk.Button(self, text="Opcion 6",command=self.op6)
        btnOp07 = Tk.Button(self, text="Opcion 7",command=self.op7)
        btnOp08 = Tk.Button(self, text="Opcion 8",command=self.op8)
        btnOp09 = Tk.Button(self, text="Opcion 9",command=self.op9)
        btnOp10 = Tk.Button(self, text="Opcion 10",command=self.op10)
        btnCerrar1 = Tk.Button(self, text="Cerrar", command=self.onClose)
##        btnCerrar2 = Tk.Button(self, text="Cerrar", command=self.onClose)
##        btnCerrar3 = Tk.Button(self, text="Cerrar", command=self.onClose)
##        btnCerrar4 = Tk.Button(self, text="Cerrar", command=self.onClose)
##        btnCerrar5 = Tk.Button(self, text="Cerrar", command=self.onClose)
```

```
        """Ubicación de botones"""
        btnOp01.grid(row=1,column=1,sticky=N+S+E+W)
        btnOp02.grid(row=4,column=1,sticky=N+S+E+W)
        btnOp03.grid(row=7,column=1,sticky=N+S+E+W)
        btnOp04.grid(row=10,column=1,sticky=N+S+E+W)
        btnOp05.grid(row=13,column=1,sticky=N+S+E+W)
        btnOp06.grid(row=1,column=3,sticky=N+S+E+W)
        btnOp07.grid(row=4,column=3,sticky=N+S+E+W)
        btnOp08.grid(row=7,column=3,sticky=N+S+E+W)
        btnOp09.grid(row=10,column=3,sticky=N+S+E+W)
        btnOp10.grid(row=13,column=3,sticky=N+S+E+W)
        btnCerrar1.grid(row=14,column=8,sticky=N+S+E+W)
##          btnCerrar2.grid(row=1,column=4,sticky=N+S+E+W)
##          btnCerrar3.grid(row=1,column=5,sticky=N+S+E+W)
##          btnCerrar4.grid(row=1,column=6,sticky=N+S+E+W)
##          btnCerrar5.grid(row=1,column=7,sticky=N+S+E+W)

        """Posicionar imagen"""
        #Imagen
        im=Image.open("cidesi.png")
        size = 320,271
        im.thumbnail(size)
        self.img = ImageTk.PhotoImage(im)
        panel = Label(self, image = self.img)
        panel.grid(row=3,column=5,columnspan=4,rowspan=5)

    #-----------------------------------------------------------------------
    def onClose(self):#Función cerrar ventana
        """"""
        self.destroy()
        self.original_frame.show()
#-----------------------------------------------------------------------
    def op1(self):#Función abrir opción 1
        """"""
        subFrame=op1Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op2(self):#Función abrir opción 2
        """"""
        subFrame=op2Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op3(self):#Función abrir opción 3
        """"""
        subFrame=op3Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op4(self):#Función abrir opción 4
        """"""
        subFrame=op4Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op5(self):#Función abrir opción 5
        """"""
        subFrame=op5Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op6(self):#Función abrir opción 6
        """"""
        subFrame=op6Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op7(self):#Función abrir opción 7
        """"""
        subFrame=op7Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op8(self):#Función abrir opción 8
        """"""
```

```
        subFrame=op8Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op9(self):#Función abrir opción 9
        """"""
        subFrame=op9Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op10(self):#Función abrir opción 10
        """"""
        subFrame=op10Frame(self)
        self.wait_window(subFrame)
#######################################################################
class scanFrame(Tk.Toplevel):#Clase ventana A-Scan
    """"""

    #-----------------------------------------------------------------------
    def __init__(self, original):
        """Constructor"""
        self.original_frame = original
        Tk.Toplevel.__init__(self,height=windowHeight,width=windowWidth)# Inicializar
        self.geometry(universalGeometry)#Delimitador de ventana          con parámetros
        rango_filas = range(0,20)#Configuración filas
        for i in rango_filas:
            self.rowconfigure(i,minsize=24)
        rango_columnas = range(0,11)#COnfiguración columnas
        for j in rango_columnas:
            self.columnconfigure(j,minsize=50)
        self.title("A-scan")
        self.scanWidgets()

    def scanWidgets(self):#Contenido de la ventana
        """Botones"""
        btnEscan = Tk.Button(self, text="Iniciar Escaneo",command=self.grafico)
        btnOp02 = Tk.Button(self, text="Opcion 2",command=self.op2)
        btnOp03 = Tk.Button(self, text="Opcion 3",command=self.op3)
        btnOp04 = Tk.Button(self, text="Opcion 4",command=self.op4)
        btnOp05 = Tk.Button(self, text="Opcion 5",command=self.op5)
        btnOp06 = Tk.Button(self, text="Opcion 6",command=self.op6)
        btnOp07 = Tk.Button(self, text="Opcion 7",command=self.op7)
        btnOp08 = Tk.Button(self, text="Opcion 8",command=self.op8)
        btnOp09 = Tk.Button(self, text="Opcion 9",command=self.op9)
        btnOp10 = Tk.Button(self, text="Opcion 10",command=self.op10)
        btn = Tk.Button(self, text="Cerrar", command=self.onClose)


        """Ubicación de botones"""
        btnEscan.grid(row=14,column=1,sticky=N+S+E+W)
        btnOp02.grid(row=14,column=3,sticky=N+S+E+W)
        btnOp03.grid(row=14,column=5,sticky=N+S+E+W)
        btnOp04.grid(row=14,column=7,sticky=N+S+E+W)
        btnOp05.grid(row=14,column=9,sticky=N+S+E+W)
        btnOp06.grid(row=16,column=1,sticky=N+S+E+W)
        btnOp07.grid(row=16,column=3,sticky=N+S+E+W)
        btnOp08.grid(row=16,column=5,sticky=N+S+E+W)
        btnOp09.grid(row=16,column=7,sticky=N+S+E+W)
        btnOp10.grid(row=16,column=9,sticky=N+S+E+W)
        btn.grid(row=15,column=10,sticky=N+S+E+W)

        #Imagen
##          im=Image.open("cidesi.png")
##          size = 320,271
##          im.thumbnail(size)
##          self.img = ImageTk.PhotoImage(im)
##          panel = Label(self, image = self.img)
##          panel.grid(row=3,column=3,columnspan=4,rowspan=5)

        #Grafico
```

```
        self.fig = plt.figure(figsize=(8,4))
##        self.fig = plt.figure()
        self.fig, self.axes = plt.subplots(figsize=(8,4))
        self.styles = ['r-','g-','y-','m-','k-','c-']
        self.x = np.arange(0,1025)
        self.y=self.x
        self.Figure = FigureCanvasTkAgg(self.fig,master=self)
        self.Figure.get_tk_widget().grid(row=0,column=0,columnspan=11,rowspan=14)
    #-----------------------------------------------------------------------
    def plot(self,ax,style):
        """"""
        return ax.plot(self.x, self.y, style, animated=True)[0]
    #-----------------------------------------------------------------------
    def onClose(self):
        """"""
        self.destroy()
        self.original_frame.show()
    #-----------------------------------------------------------------------
    def grafico(self):
        self.axes.set_title("A-scan")
        self.axes.set_autoscaley_on(False)
        self.axes.set_ylim([-255,255])
        self.axes.set_xlim([0,1025])
        self.fig.canvas.draw()
        self.backgrounds = self.fig.canvas.copy_from_bbox(self.axes.bbox)
        self.lines = self.plot(self.axes,self.styles[0])
##        for i in xrange(1,1000):
##            self.fig.canvas.restore_region(self.backgrounds)
##            self.lines.set_ydata((255*np.sin(self.x + i/10.0))/2 + 128)
##            self.axes.draw_artist(self.lines)
##            self.fig.canvas.blit(self.axes.bbox)
        y= rd()
        self.fig.canvas.restore_region(self.backgrounds)
        self.lines.set_ydata(y)
        self.axes.draw_artist(self.lines)
        self.fig.canvas.blit(self.axes.bbox)
    #-----------------------------------------------------------------------
    def op1(self):#Función abrir opción 1
        """"""
        subFrame=op1Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op2(self):#Función abrir opción 2
        """"""
        subFrame=op2Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op3(self):#Función abrir opción 3
        """"""
        subFrame=op3Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op4(self):#Función abrir opción 4
        """"""
        subFrame=op4Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op5(self):#Función abrir opción 5
        """"""
        subFrame=op5Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op6(self):#Función abrir opción 6
        """"""
        subFrame=op6Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op7(self):#Función abrir opción 7
        """"""
```

```
        subFrame=op7Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op8(self):#Función abrir opción 8
        """"""
        subFrame=op8Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op9(self):#Función abrir opción 9
        """"""
        subFrame=op9Frame(self)
        self.wait_window(subFrame)
    #-----------------------------------------------------------------------
    def op10(self):#Función abrir opción 10
        """"""
        subFrame=op10Frame(self)
        self.wait_window(subFrame)

#######################################################################
class MyApp(Frame):

    #-----------------------------------------------------------------------
    def __init__(self, parent):
        """Constructor"""
        Frame.__init__(self, parent, height=windowHeight, width=windowWidth)# Inicializar
                # con parámetros por defecto
        self.root = parent
        self.root.title("Bienvenido")
##          self.root.config(="black")
##          self.frame = Tk.Frame(parent)
##          self.frame.pack()
##          self.attributes("-toolwindow", 1)
        self.pack()
        self.create_widgets()

    def create_widgets(self):#Contenido de la ventana
        """Botones con características específicas"""
        strBtn = Button(self, text="A-scan", command=self.openFrame,height=3,width=30)
        stBtn = Button(self, text="Configuraciones", command=self.openFrame2,height=3,width=30)
        abBtn = Button(self, text="Acerca",command=self.openFrame3,height=3,width=30)
        exBtn = Button(self, text="Salir",command=self.Salir,height=3,width=30)
        strBtn.place(relx=0.01, rely=0.03)
        stBtn.place(relx=0.01, rely=0.29)
        abBtn.place(relx=0.01, rely=0.58)
        exBtn.place(relx=0.01, rely=0.85)

        #Imagen
        im=Image.open("cidesi.png")
        size = 375,271
        im.thumbnail(size)
        self.img = ImageTk.PhotoImage(im)
        panel = Label(self, image = self.img)
        panel.place(relx=0.45,rely=0.3)

    #-----------------------------------------------------------------------
    def hide(self):#Función prara occultar ventana
        """"""
        self.root.withdraw()

    #-----------------------------------------------------------------------
    def openFrame(self):#Función para abrir ventana de A-scan
        """"""
        self.hide()
        subFrame = scanFrame(self)

    #-----------------------------------------------------------------------
    def openFrame2(self):#Función para abrir ventana de configuración
        """"""
        self.hide()
```

```
        subFrame = confFrame(self)
    #-----------------------------------------------------------------------
    def openFrame3(self):#Función para abrir ventana de información
        """"""
##          state=
        global bA
        state = bA
        if state==0:
            subFrame=aboutFrame(self)
        else:
            pass
    #-----------------------------------------------------------------------
    def show(self):#Función para mostrar la ventana
        """"""
        self.root.update()
        self.root.deiconify()
    #-----------------------------------------------------------------------
    def Salir(self):#Función para salir del programa
        self.root.destroy()
        sys.exit(0)

#-----------------------------------------------------------------------
if __name__ == "__main__":
    root = Tk.Tk()#Instanciaicón de las librerías gráficas
    root.geometry(universalGeometry)#Delimtador de ventana
    app = MyApp(root)#Instanciación de la ventana principal
    root.mainloop()#Ciclo infinito
```